MEAutility Documentation

Release 1.2.1

Alessio Buccino

Oct 30, 2024

CONTENTS

Python package for multi-electrode array (MEA) handling and stimulation.

CHAPTER

ONE

INSTALLATION

To install run:

```
pip install MEAutility
```

If you want to install from sources and be updated with the latest development you can install with:

```
git clone https://github.com/alejoe91/MEAutility
cd MEAutility
python setup.py install (or develop)
```

The package can then imported in Python with:

```
import MEAutility as MEA
```

CHAPTER

TWO

REQUIREMENTS

- numpy
- pyyaml
- matplotlib

CHAPTER

THREE

CONTENTS

The following sections will guide you through definitions and handling of MEA models, as well as electrical stimulation and plotting functions.

3.1 MEA definition

This notebook shows how MEA can be using a .yaml file and how MEA models can be added and removed to and from the file system.

```
import MEAutility as MEA
from pprint import pprint
import matplotlib.pylab as plt
```

3.1.1 List available MEAs:

```
MEA.return_mea()
```

```
Available MEA:
 ['SqMEA-6-25um', 'SqMEA-10-15um', 'tetrode', 'Neuroseeker-128', 'SqMEA-5-30um', 'SqMEA-
↪15-10um', 'Neuronexus-32-Kampff', 'Neuronexus-32-cut-30', 'Neuropixels-128',
↪'Neuroseeker-128-Kampff', 'Neuropixels-24', 'SqMEA-7-20um', 'Neuronexus-32',
↪'Neuropixels-384']
```

These MEA are saved during installation. Each MEA corresponds to a .yaml file containing key information for the MEA. Let's take a look at some examples.

3.1.2 Square MEA

```
sqmea_info = MEA.return_mea_info('SqMEA-10-15um')
pprint(sqmea_info)
```

```
{'dim': 10,
 'electrode_name': 'SqMEA-10-15um',
 'pitch': 15,
 'shape': 'square',
 'size': 5,
```

(continues on next page)

(continued from previous page)

```
'sortlist': None,
'type': 'mea'}
```

The returned dictionary corresponds the the .yaml file. For this MEA model `dim` is a single `int` and `pitch` is a single `int` (or `float`). Therefore, a 10x10 Square MEA is instantiated with 15um pitch in the yz direction (if `plane` is not in the yaml file, `yz` is default). The electrodes `shape` is `square`, and half the side length is 5um. Since `sortlist` is `None`, the electrode count starts from the bottom left and it follows the rows up and then goes to the next column (the last index is the electrode on the top right). The `type` `mea` will be used for plotting.

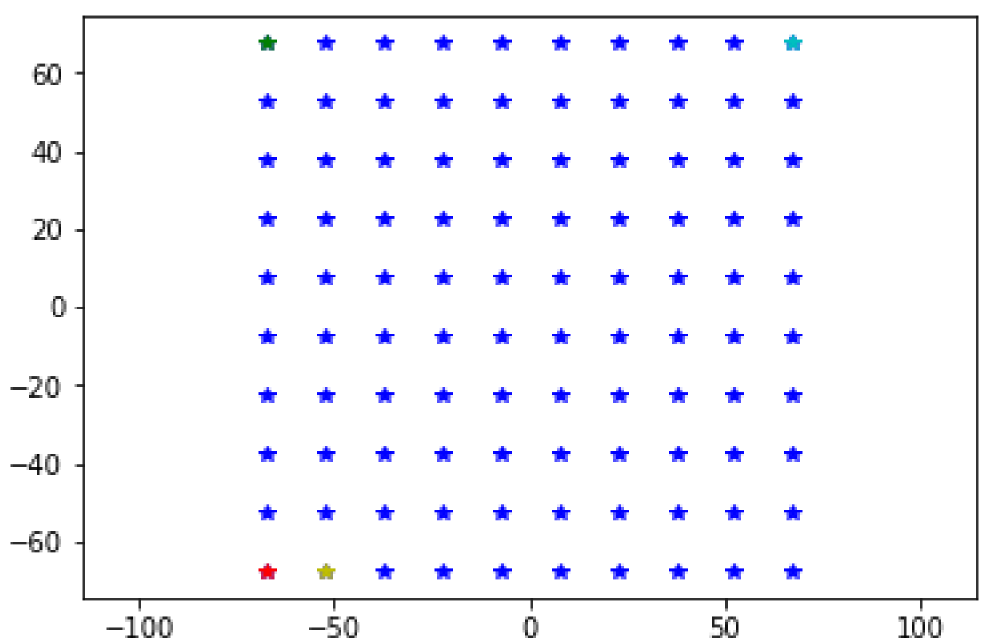
Let's now instantiate a MEA object:

```
sqmea = MEA.return_mea('SqMEA-10-15um')
print(type(sqmea))
print(sqmea.number_electrodes)
print(sqmea.dim)
```

```
'plane' field with 2D dimensions assumed to be 'yz
Model is set to semi
<class 'MEAutility.core.RectMEA'>
100
[10, 10]
```

The MEA is a rectangular MEA with 100 electrodes.

```
plt.plot(sqmea.positions[:, 1], sqmea.positions[:, 2], 'b*')
plt.plot(sqmea.positions[0, 1], sqmea.positions[0, 2], 'r*')
plt.plot(sqmea.positions[9, 1], sqmea.positions[9, 2], 'g*')
plt.plot(sqmea.positions[10, 1], sqmea.positions[10, 2], 'y*')
plt.plot(sqmea.positions[-1, 1], sqmea.positions[-1, 2], 'c*')
_ = plt.axis('equal')
```



Rectangular MEAs can be handled as matrices, where the first inex is the ROW and the second index is the COLUMN:

```
print(sqmea[0][0].position) # electrode 0
print(sqmea[9][0].position) # electrode 9
print(sqmea[0][1].position) # electrode 10
print(sqmea[-1][-1].position) # electrode 99
```

```
[  0.  -67.5 -67.5]
[  0.  -67.5  67.5]
[  0.  -52.5 -67.5]
[ 0.  67.5 67.5]
```

3.1.3 Rectangular MEA

```
neuroseeker_info = MEA.return_mea_info('Neuroseeker-128')
pprint(neuroseeker_info)
```

```
{'dim': [32, 4],
 'electrode_name': 'Neuroseeker-128',
 'pitch': 22.5,
 'shape': 'square',
 'size': 10.0,
 'sortlist': None,
 'type': 'mea'}
```

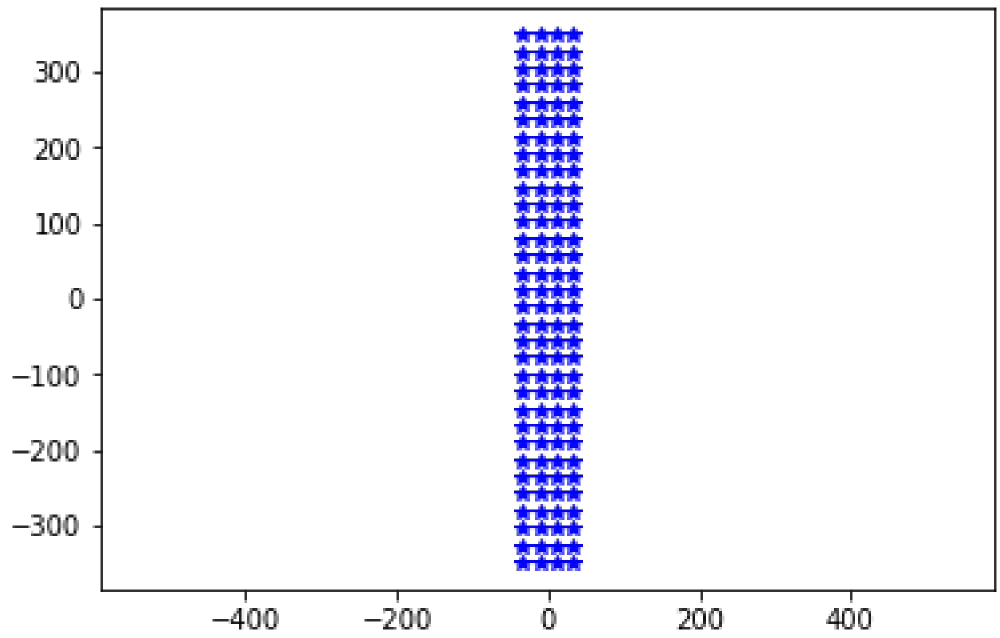
This MEA is rectangular, with 32 rows, 4 columns, and a regular pitch of 22.5um

```
neuroseeker = MEA.return_mea('Neuroseeker-128')
print(type(neuroseeker))
print(neuroseeker.number_electrodes)
print(neuroseeker.dim)
```

```
'plane' field with 2D dimensions assumed to be 'yz
Model is set to semi
<class 'MEAutility.core.RectMEA'>
128
[32, 4]
```

```
plt.plot(neuroseeker.positions[:, 1], neuroseeker.positions[:, 2], 'b*')
_ = plt.axis('equal')
print(neuroseeker[0][0].position) # electrode 0
print(neuroseeker[31][0].position) # electrode 31
print(neuroseeker[1][0].position) # electrode 32
print(neuroseeker[-1][-1].position) # electrode 127
```

```
[   0.    -33.75 -348.75]
[  0.   -33.75 348.75]
[   0.    -33.75 -326.25]
[  0.    33.75 348.75]
```



3.1.4 General MEA

When `dim` and `pitch` is are single `int` (or `float` for `pitch`) or a list of 2 values, a rectangular MEA is created. Some MEA configuration can be different.

```
neuronexus_info = MEA.return_mea_info('Neuronexus-32')
pprint(neuronexus_info)
```

```
{'dim': [10, 12, 10],
 'electrode_name': 'Neuronexus-32',
 'pitch': [25.0, 18.0],
 'shape': 'circle',
 'size': 7.5,
 'sortlist': None,
 'stagger': -12.5,
 'type': 'mea'}
```

For this MEA there are 3 different options: - `dim` has 3 elements - `pitch` hass 2 elements - `stagger` is present

When len(`dim`) > 2, then each element represents the number of rows of each column. In this case, there are 3 columns: the first and third have 10 electrodes, the second one has 12.

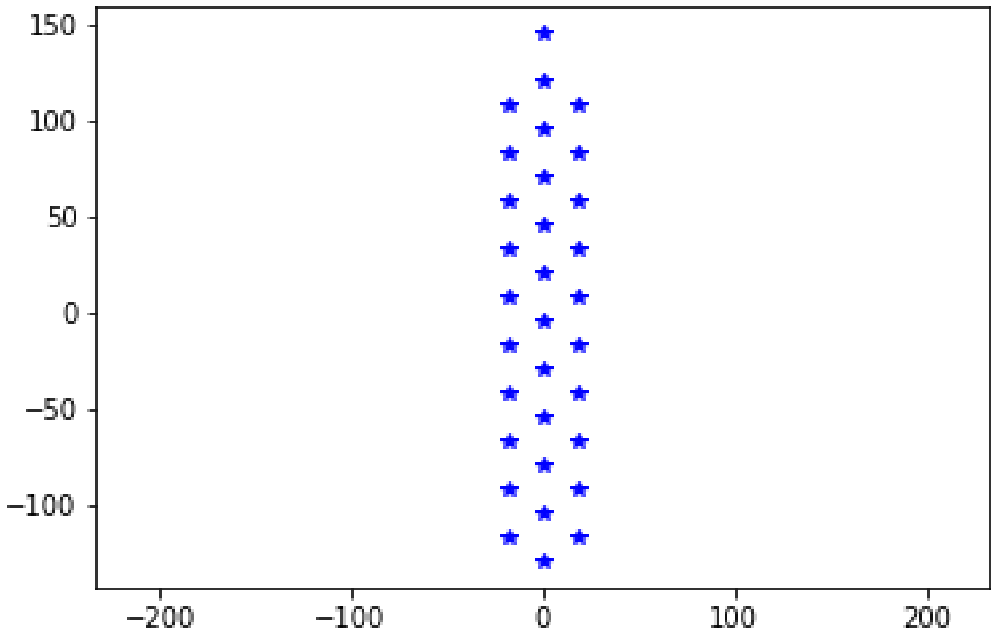
The first value of `pitch` is the inter-row distance (top to bottom). The second value is the inter-column distance (left to right).

The `stagger` key allows the shift colimns. If only one value is given (`int` or `float`) every other column starting from he second one is staggered. Otherwise `stagger` can be a list with the same number of elements of `dim`.

Given this information, we can wxpect how the neuronexus MEA looks like:

```
neuronexus = MEA.return_mea('Neuronexus-32')
plt.plot(neuronexus.positions[:, 1], neuronexus.positions[:, 2], 'b*')
_ = plt.axis('equal')
```

```
'plane' field with 2D dimensions assumed to be 'yz
Model is set to semi
```



Adding and removing MEA models

It is possible to load user-defined yaml files in the MEAutility package, so that they are available from the entire file system.

Let's first create a `user.yaml` file on-the-fly.

```
import yaml, os

user_info = {'dim': [10, 12, 9, 8],
             'electrode_name': 'user',
             'description': "a brief description of the probe",
             'pitch': [10.0, 40.0],
             'shape': 'circle',
             'size': 7.5,
             'sortlist': None,
             'stagger': [0, -12, 30, -22],
             'type': 'mea'}

with open('user.yaml', 'w') as f:
    yaml.dump(user_info, f)

yaml_files = [f for f in os.listdir('.') if f.endswith('.yaml')]
print(yaml_files)
```

```
['user.yaml']
```

Now we can add the newly created yaml file to the MEA package:

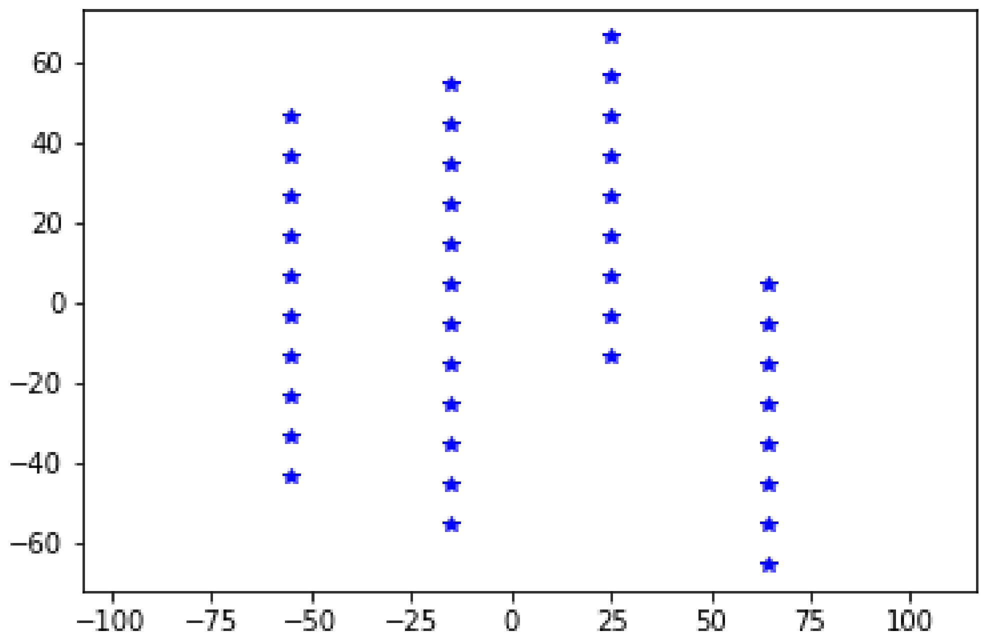
```
MEA.add_mea('user.yaml')
```

```
Available MEA:
 ['SqMEA-6-25um', 'SqMEA-10-15um', 'tetrode', 'Neuroseeker-128', 'SqMEA-5-30um', 'SqMEA-
↪15-10um', 'Neuronexus-32-Kampff', 'Neuronexus-32-cut-30', 'Neuropixels-128',
↪'Neuroseeker-128-Kampff', 'Neuropixels-24', 'SqMEA-7-20um', 'Neuronexus-32', 'user',
↪'Neuropixels-384']
```

and create a user MEA object:

```
usermea = MEA.return_mea('user')
plt.plot(usermea.positions[:, 1], usermea.positions[:, 2], 'b*')
_ = plt.axis('equal')
```

```
'plane' field with 2D dimensions assumed to be 'yz
Model is set to semi
```



If we don't need the user MEA anymore, we can remove it from the MEA package:

```
MEA.remove_mea('user')
```

```
Removed:  /home/alessiob/anaconda3/envs/mearec/lib/python3.6/site-packages/MEAutility/
↪electrodes/user.yaml
Available MEA:
 ['SqMEA-6-25um', 'SqMEA-10-15um', 'tetrode', 'Neuroseeker-128', 'SqMEA-5-30um', 'SqMEA-
↪15-10um', 'Neuronexus-32-Kampff', 'Neuronexus-32-cut-30', 'Neuropixels-128',
↪'Neuroseeker-128-Kampff', 'Neuropixels-24', 'SqMEA-7-20um', 'Neuronexus-32',
↪'Neuropixels-384']
```

3.2 MEA handling

This notebook shows how to handle MEA and electrodes in he 3D space.

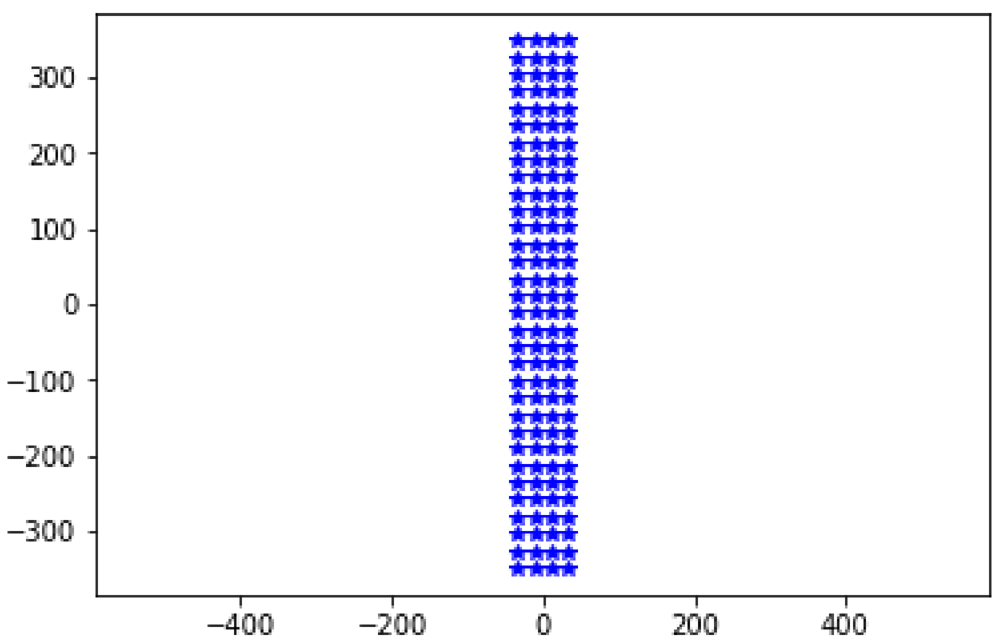
```
import MEAutility as MEA
import matplotlib.pylab as plt
```

First, let's instantiate a MEA object among the available MEAs:

```
MEA.return_mea()
```

```
Available MEA:
 ['SqMEA-6-25um', 'SqMEA-10-15um', 'circle_500', 'tetrode', 'Neuroseeker-128', 'SqMEA-5-
↪30um', 'SqMEA-15-10um', 'Neuronexus-32-Kampff', 'Neuronexus-32-cut-30', 'Neuropixels-
↪128', 'Neuroseeker-128-Kampff', 'Neuropixels-24', 'SqMEA-7-20um', 'Neuronexus-32',
↪'Neuropixels-384']
```

```
neuroseeker = MEA.return_mea('Neuroseeker-128')
plt.plot(neuroseeker.positions[:, 1], neuroseeker.positions[:, 2], 'b*')
_ = plt.axis('equal')
```



By default the MEA is instantiated with it's center of mass at (0,0,0) and electrodes lying in the plane specified in the yaml file (by default `plane` is `yz`)

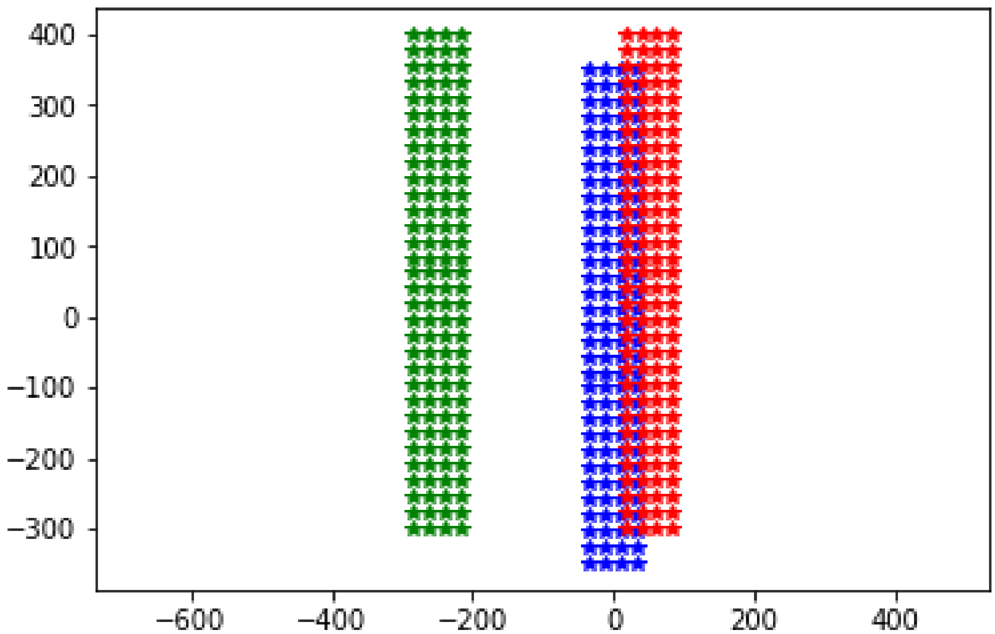
```
neuroseeker.plane
```

```
'yz'
```

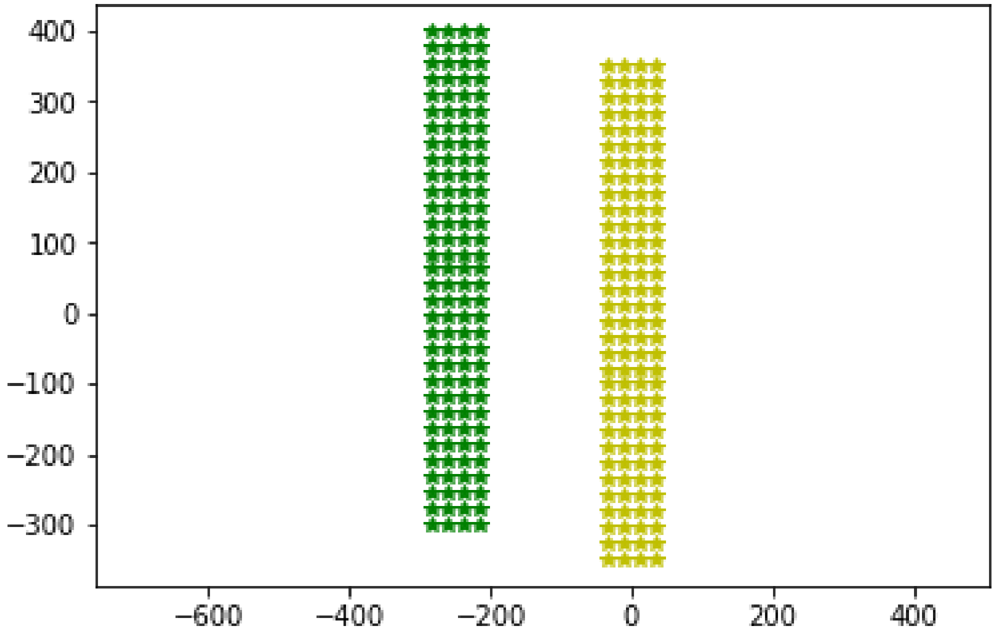
3.2.1 Moving the probe around

The probe can be easily moved with a the `move` and `center` methods:

```
plt.plot(neuroseeker.positions[:, 1], neuroseeker.positions[:, 2], 'b*')
neuroseeker.move([0, 50, 50])
plt.plot(neuroseeker.positions[:, 1], neuroseeker.positions[:, 2], 'r*')
neuroseeker.move([0, -300, 0])
plt.plot(neuroseeker.positions[:, 1], neuroseeker.positions[:, 2], 'g*')
_ = plt.axis('equal')
```




```
plt.plot(neuroseeker.positions[:, 1], neuroseeker.positions[:, 2], 'g*')
neuroseeker.center()
plt.plot(neuroseeker.positions[:, 1], neuroseeker.positions[:, 2], 'y*')
_ = plt.axis('equal')
```



3.2.2 Rotating the probe

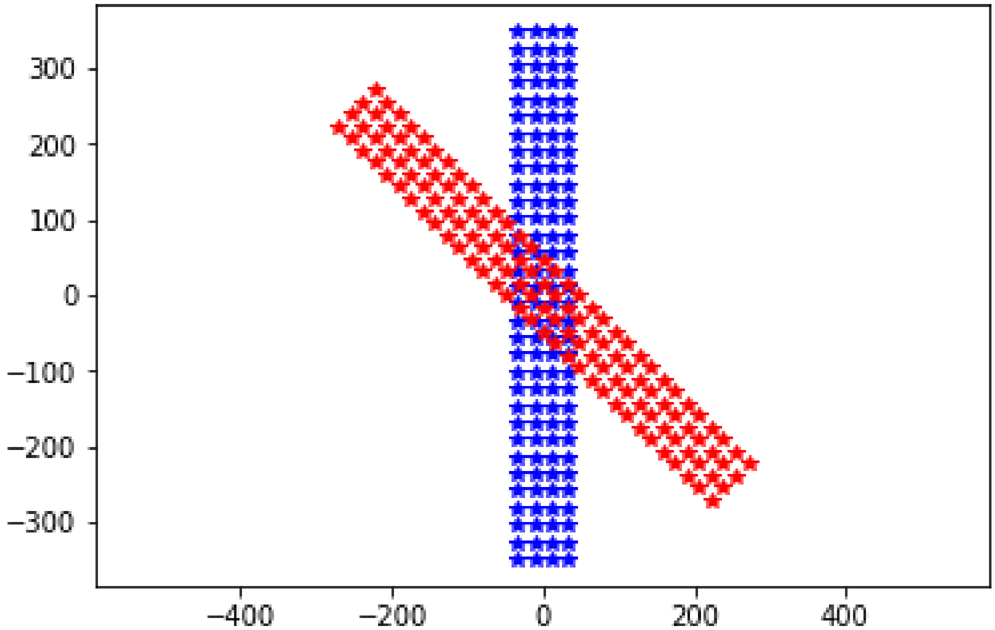
With the `rotate` method, MEA probes can be rotated along any axis by any angle (in degrees). The current plane and orientation of the probe is stored by the variables `main_axes` and `normal`

```
# main_axes indicate the MEA plane
print(neuroseeker.main_axes[0], neuroseeker.main_axes[1])

# normal indicates the axis perpendicular to the electrodes
print(neuroseeker.normal)

# normal axis is also stored by each electrode and could be changed separately
print(type(neuroseeker.electrodes[0]), neuroseeker.electrodes[0].normal)
```

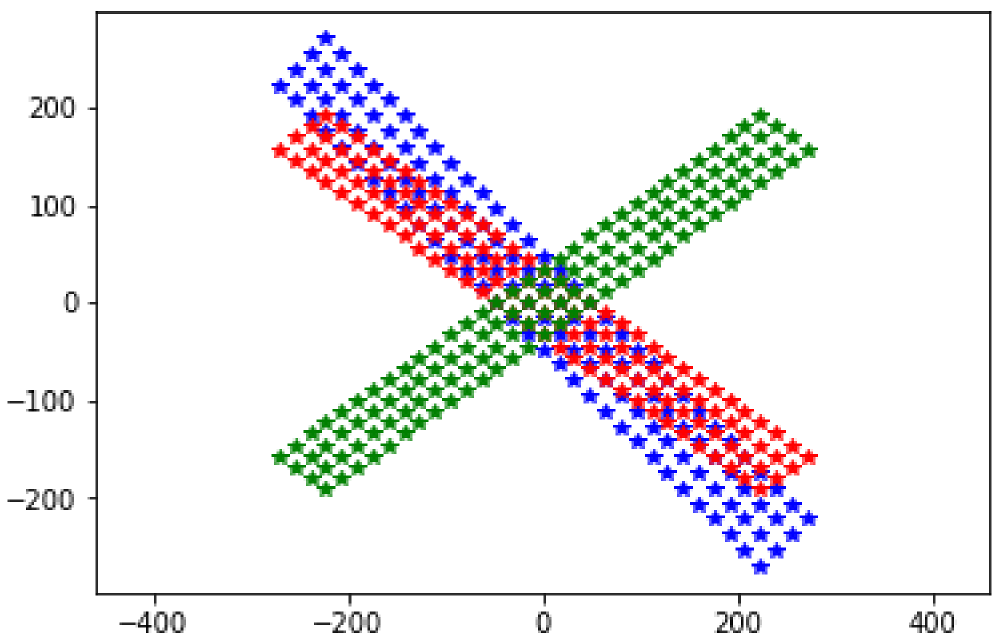
```
[0 1 0] [0 0 1]
[-1.  0.  0.]
<class 'MEAutility.core.Electrode'> [-1.  0.  0.]
```

Now le's make some rotations!!

```
plt.plot(neuroseeker.positions[:, 1], neuroseeker.positions[:, 2], 'b*')
neuroseeker.rotate([1, 0, 0], 45)
plt.plot(neuroseeker.positions[:, 1], neuroseeker.positions[:, 2], 'r*')
_ = plt.axis('equal')
```




```
plt.plot(neuroseeker.positions[:, 1], neuroseeker.positions[:, 2], 'b*')
neuroseeker.rotate([0, 1, 0], 45)
plt.plot(neuroseeker.positions[:, 1], neuroseeker.positions[:, 2], 'r*')
neuroseeker.rotate([0, 1, 0], 90)
plt.plot(neuroseeker.positions[:, 1], neuroseeker.positions[:, 2], 'g*')
_ = plt.axis('equal')
```

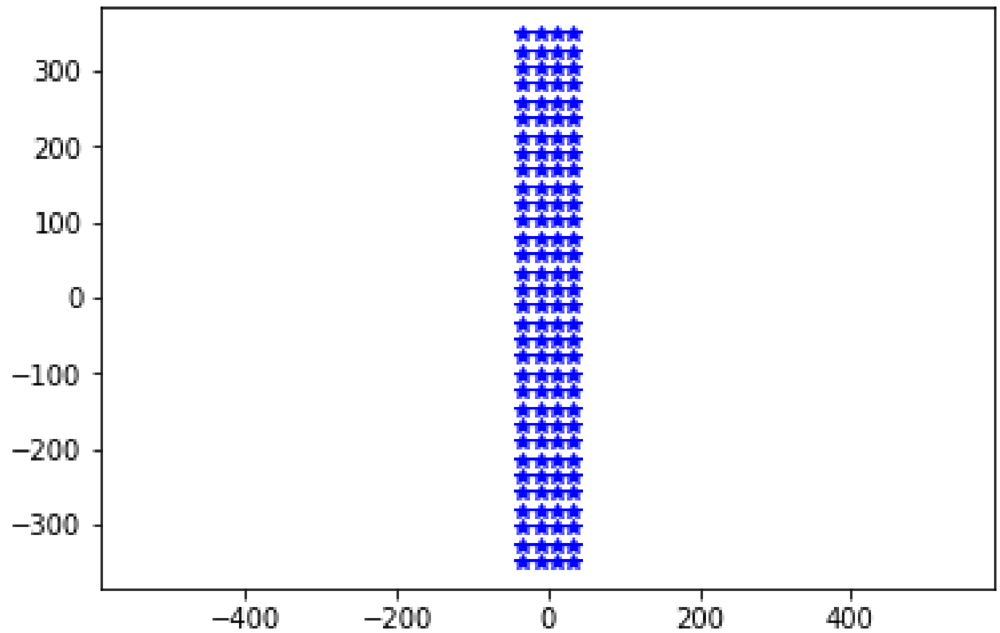



```
# back to normal
neuroseeker.rotate([0, 1, 0], -90)
```

(continues on next page)

(continued from previous page)

```
neuroseeker.rotate([0, 1, 0], -45)
neuroseeker.rotate([1, 0, 0], -45)
plt.plot(neuroseeker.positions[:, 1], neuroseeker.positions[:, 2], 'b*')
_ = plt.axis('equal')
```



3.3 MEA stimulation

This notebook shows how to simulate the electric potential generated by electrode currents using a MEA object. Stimulation is performed by means of currents. Voltage stimulation is not implemented as it strongly depends on the electrode itself (e.g. faradaic/capacitive).

```
import MEAutility as MEA
import matplotlib.pylab as plt
import numpy as np
```

First, let's instantiate a MEA object among the available MEA models:

```
MEA.return_mea()
```

```
Available MEA:
 ['SqMEA-15-10um', 'SqMEA-6-25um', 'Neuronexus-32-cut-30', 'SqMEA-5-30um', 'Neuropixels-
↪384', 'SqMEA-10-15um', 'Neuropixels-128', 'SqMEA-7-20um', 'Neuronexus-32-Kampff',
↪'Neuroseeker-128', 'tetrode', 'Neuropixels-24', 'Neuronexus-32', 'Neuroseeker-128-
↪Kampff', 'tetrode_mea']
```

```
sqmea = MEA.return_mea('SqMEA-10-15um')
```

By default, the stimulation `model` is set to `semi`. This is the default for MEA objects of type `mea` and it models that currents radiate only on one side of the probe (the MEA is considered as an infinite insulating plane). The underlying assumption is that ground is infinitely far away. In this case the electric potential at point $\vec{r}$ generated by the electrode currents I_i is (electrode positions are $\vec{r_i}$):

$$V(\vec{r}) = \sum_i \frac{I_i}{2\sigma\pi|\vec{r} - \vec{r_i}|}$$

where σ is the tissue conductivity.

Instead, for mea type `wire`, the tissue is assumed to be infinite and homogeneous, that is the probe has no effect on the electric potential and currents radiate in all directions:

$$V(\vec{r}) = \sum_i \frac{I_i}{4\sigma\pi|\vec{r} - \vec{r_i}|}$$

3.3.1 Conventions

- currents are in nA
- distances and positions are in μm
- electric potentials are in mV

3.3.2 Handling currents

MEA currents can be easily accessed and changed in various ways:

```
# check currents
print(sqmea.currents)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0.]
```

```
# set currents with an array
curr = np.arange(sqmea.number_electrodes)
sqmea.currents = curr
print(sqmea.currents)

#set currents with a list
curr = list(curr)
sqmea.currents = curr
print(sqmea.currents)
```

```
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
 18. 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35.
 36. 37. 38. 39. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49. 50. 51. 52. 53.
 54. 55. 56. 57. 58. 59. 60. 61. 62. 63. 64. 65. 66. 67. 68. 69. 70. 71.
 72. 73. 74. 75. 76. 77. 78. 79. 80. 81. 82. 83. 84. 85. 86. 87. 88. 89.
 90. 91. 92. 93. 94. 95. 96. 97. 98. 99.]
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
 18. 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35.
```

(continues on next page)

(continued from previous page)

```
 36. 37. 38. 39. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49. 50. 51. 52. 53.
 54. 55. 56. 57. 58. 59. 60. 61. 62. 63. 64. 65. 66. 67. 68. 69. 70. 71.
 72. 73. 74. 75. 76. 77. 78. 79. 80. 81. 82. 83. 84. 85. 86. 87. 88. 89.
 90. 91. 92. 93. 94. 95. 96. 97. 98. 99.]
```

```
# reset currents to 0
sqmea.reset_currents()
print(sqmea.currents)

# reset currents to 100
sqmea.reset_currents(100)
print(sqmea.currents)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0.]
[100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100.
 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100.
 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100.
 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100.
 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100.
 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100.
 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100.
 100. 100.]
```

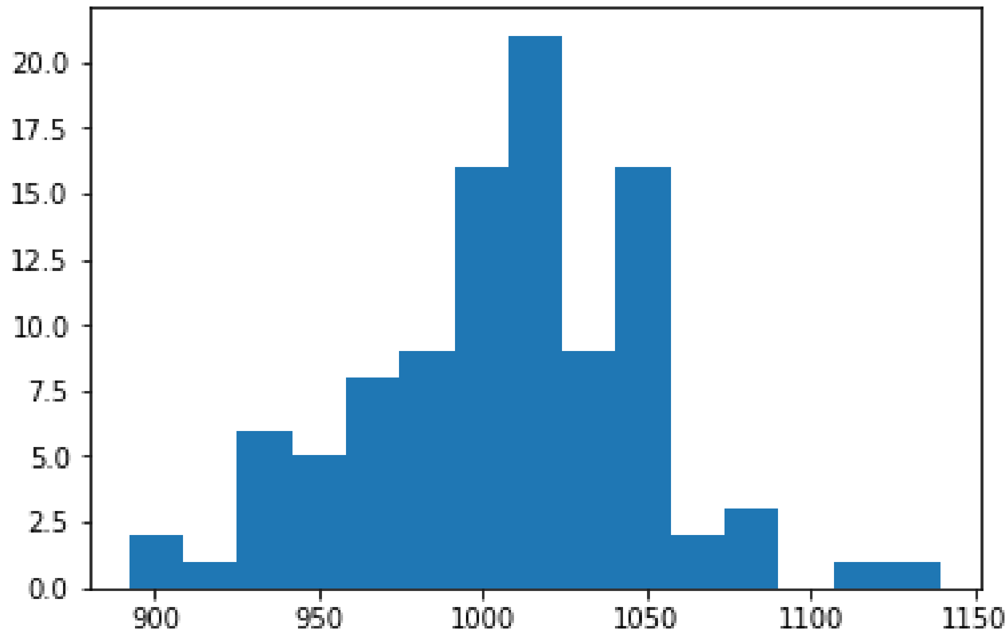
```
# random values with a certain amplitude and standard deviation
sqmea.set_random_currents(mean=1000, sd=50)
print(sqmea.currents)
_ = plt.hist(sqmea.currents, bins=15)
```

```
[ 973.91615691 1016.83720943 1089.49043841 1139.8579249   927.79316233
 1000.89661725 1047.3334144  1051.8497402   927.37268018  996.62983039
 1016.49251336 1043.75742297 1004.9168758   940.30748105 1054.53993841
  973.75422086  983.60405175 1042.34697708 1040.74580548 1014.98436691
 1001.8608754   995.65886874 1012.95710254  970.06809296  927.99036328
  999.92788465 1049.19541344  997.14646988 1039.79123706  984.20047048
  930.55017661 1009.74184644 1023.24453635 1018.02056444 1049.41097968
 1017.43562542 1062.60398159  973.51622737 1053.37464287  892.22969949
  999.73394752 1012.93137879  980.73150404  953.77253661  951.55426365
  905.11921863 1107.92750924  913.69396055 1077.18729127  962.6261477
 1043.49287399  952.72622053  993.51633173 1029.79201114 1014.65998008
  986.78997864 1007.9228314   973.1521672  1039.92862132  993.2816604
 1058.30275146  951.99364936 1047.30143561 1004.77930621 1010.1738069
  960.06196844  991.50504623  999.62108637 1037.74033168 1022.7296349
 1016.31311019 1020.75966681 1039.98604723  937.02190389 1050.16695834
 1041.47298494 1057.30344821 1022.87078261 1026.73934869 1049.05606228
 1010.57269555 1019.66052338  977.72552581 1043.29217666  988.32520744
 1003.95374263 1088.5345568   981.05722135  976.19800375 1037.08286147
 1026.14202785 1016.49830716 1012.46829058 1041.29563699 1010.75733243
```

(continues on next page)

(continued from previous page)

```
 1005.74013272  958.06708739 1007.22074273  985.12744284  969.1025596 ]
```



For Rectangular MEAs, currents can be handled with matrices:

```
print(sqmea.get_current_matrix())
print('Shape: ', sqmea.get_current_matrix().shape)
```

```
[[ 973.91615691 1016.49251336 1001.8608754   930.55017661  999.73394752
  1043.49287399 1058.30275146 1016.31311019 1010.57269555 1026.14202785]
 [1016.83720943 1043.75742297  995.65886874 1009.74184644 1012.93137879
   952.72622053  951.99364936 1020.75966681 1019.66052338 1016.49830716]
 [1089.49043841 1004.9168758  1012.95710254 1023.24453635  980.73150404
   993.51633173 1047.30143561 1039.98604723  977.72552581 1012.46829058]
 [1139.8579249   940.30748105  970.06809296 1018.02056444  953.77253661
  1029.79201114 1004.77930621  937.02190389 1043.29217666 1041.29563699]
 [ 927.79316233 1054.53993841  927.99036328 1049.41097968  951.55426365
  1014.65998008 1010.1738069  1050.16695834  988.32520744 1010.75733243]
 [1000.89661725  973.75422086  999.92788465 1017.43562542  905.11921863
   986.78997864  960.06196844 1041.47298494 1003.95374263 1005.74013272]
 [1047.3334144   983.60405175 1049.19541344 1062.60398159 1107.92750924
  1007.9228314   991.50504623 1057.30344821 1088.5345568   958.06708739]
 [1051.8497402  1042.34697708  997.14646988  973.51622737  913.69396055
   973.1521672   999.62108637 1022.87078261  981.05722135 1007.22074273]
 [ 927.37268018 1040.74580548 1039.79123706 1053.37464287 1077.18729127
  1039.92862132 1037.74033168 1026.73934869  976.19800375  985.12744284]
 [ 996.62983039 1014.98436691  984.20047048  892.22969949  962.6261477
   993.2816604  1022.7296349  1049.05606228 1037.08286147  969.1025596 ]]
Shape:  (10, 10)
```

```
current_of_zeros = np.zeros((10,10))
```

(continues on next page)

(continued from previous page)

```
print(current_of_zeros)
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

```
sqmea.set_current_matrix(current_of_zeros)
sqmea.get_current_matrix()
```

```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

Single currents can be set separately either by:

```
# set elecectrode 50 current to 10000
sqmea.set_current(24, 10000)
sqmea.currents
```

```
array([    0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
           0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
           0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
       10000.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
           0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
           0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
           0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
           0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
           0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
           0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
           0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
           0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
           0.,     0.,     0.,     0.])
```

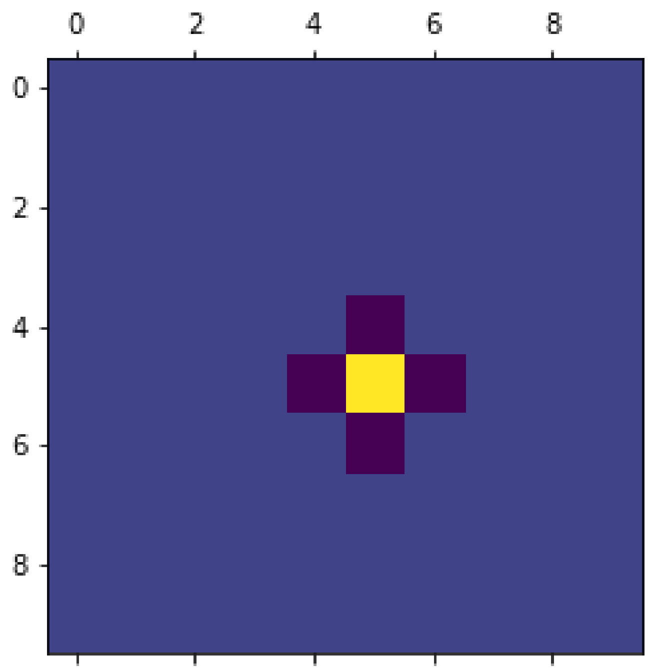
Or by using matrix notation for rectangular MEAs. This makes it easy, for example, to create multipolar current sets.

```
# reset elecectrode 50 current to 0
sqmea.set_current(24, 0)
```

(continues on next page)

(continued from previous page)

```
center_electrode = sqmea.dim[0]//2

# build a multipolar current set
sqmea[center_electrode][center_electrode].current = 8000
sqmea[center_electrode+1][center_electrode].current = -2000
sqmea[center_electrode-1][center_electrode].current = -2000
sqmea[center_electrode][center_electrode+1].current = -2000
sqmea[center_electrode][center_electrode-1].current = -2000

_ = plt.matshow(sqmea.get_current_matrix())
```



3.3.3 Stimulation

Once currents are set, electric potentials can be computed with the `compute field` function. Let's first create a bunch of 3d points, for example, on a straight line from close to the active electrode.

```
center_pos = sqmea[center_electrode][center_electrode].position
print(center_pos)
```

```
[0.  7.5 7.5]
```

```
npoints = 1000
x_vec = np.linspace(5, 100, npoints)
y_vec = [center_pos[1]] * npoints
z_vec = [center_pos[2]] * npoints

points = np.array([x_vec, y_vec, z_vec]).T
# points should be a np.array (or list) o npoints x 3
```

(continues on next page)

(continued from previous page)

```
print(points.shape)
print(points)
```

```
(1000, 3)
[[  5.           7.5          7.5        ]
 [  5.0950951    7.5          7.5        ]
 [  5.19019019   7.5          7.5        ]
 ...
 [ 99.80980981   7.5          7.5        ]
 [ 99.9049049    7.5          7.5        ]
 [100.           7.5          7.5        ]]
```

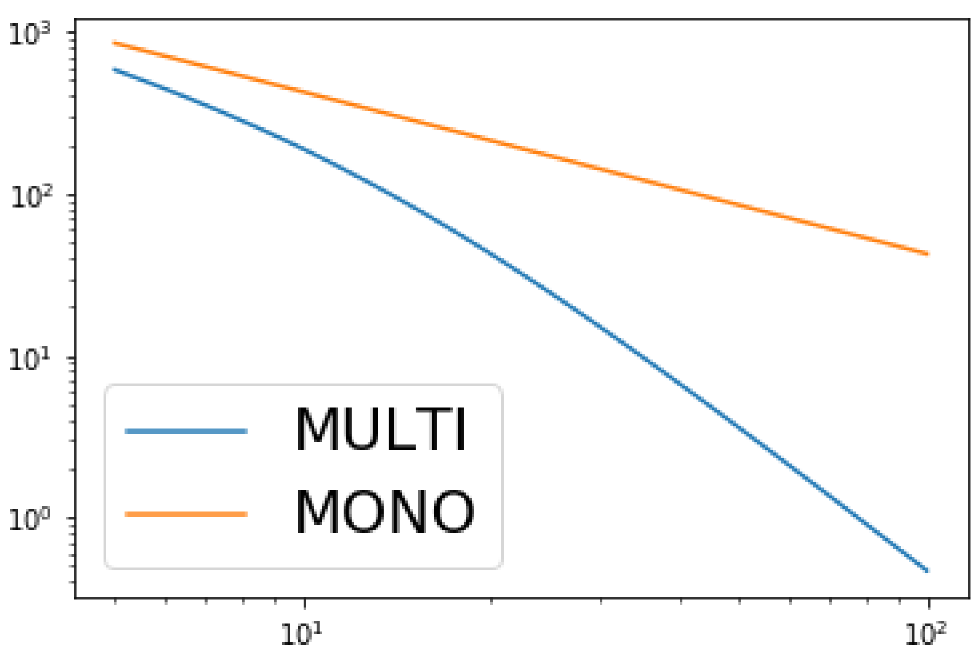
Now, we can compute the electric potential:

```
# multipolar currents
Vp_multi = sqmea.compute_field(points)
```

and compare the field generated by a single electrode (monopolar current source).

```
# monopolar currents
sqmea.reset_currents()
sqmea[5][5].current = 8000
Vp_mono = sqmea.compute_field(points)
```

```
_ = plt.loglog(x_vec, Vp_multi, label='MULTI')
_ = plt.loglog(x_vec, Vp_mono, label='MONO')
_ = plt.legend(fontsize=22)
```



The potential fall for the multipolar is faster than the monopolar configuration (which is linear in log scale)!

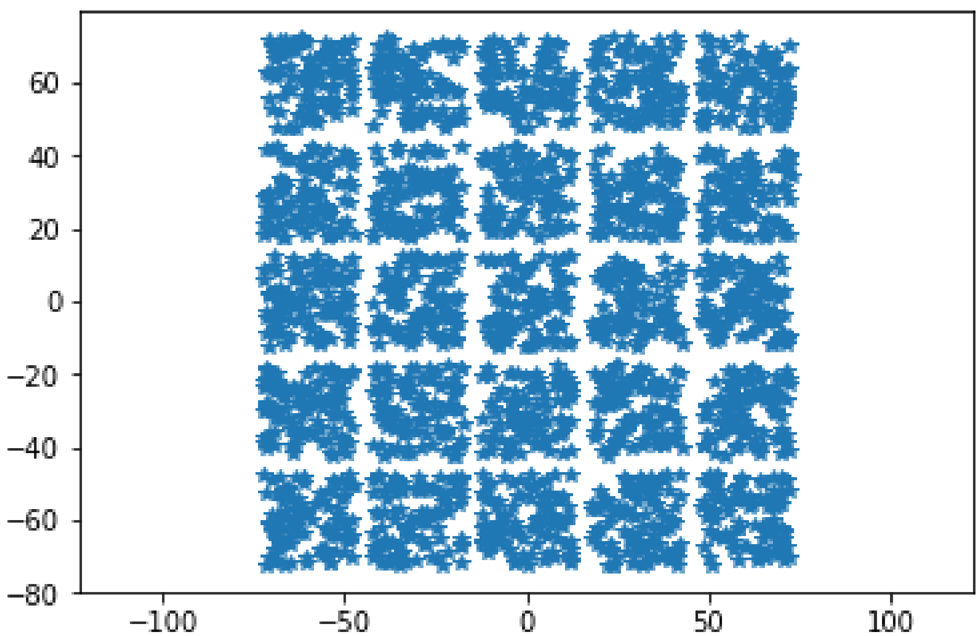
3.3.4 Finite electrode effect

So far, we assumed that the electrodes were point sources, but this is of course not the case as they have a finite size. In some cases the finite size of the electrode may be taken into consideration. In order to do so, one can set the variable `points_per_electrode` of the MEA object to the number of points within the electrode in which the entire electrode current is split.

Let's take a look at an example:

```
sqmea_r = MEA.return_mea('SqMEA-5-30um')
center_electrode = sqmea_r.dim[0] // 2

# Activate all electrodes
sqmea_r.set_random_currents(mean=0, sd=10000)
reduced_points = points[:10]
```

```
sqmea_r.points_per_electrode = 100

# compute electric potential and return stimulation points
vp, stim_points = sqmea_r.compute_field(reduced_points, return_stim_points=True)
_  = plt.plot(stim_points[:, 1], stim_points[:, 2], '*')
_ = plt.axis('equal')
```



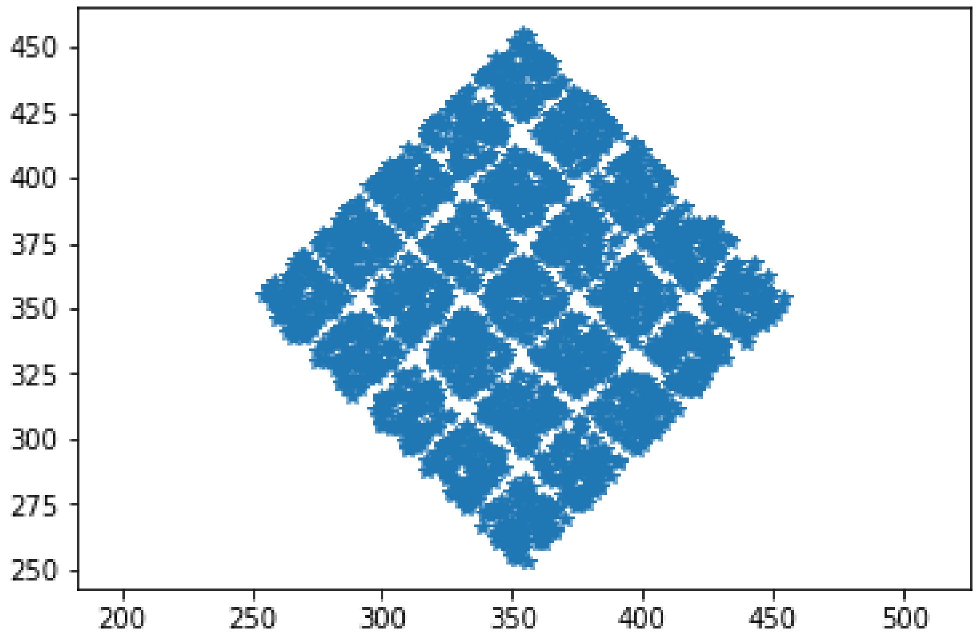
The stimulation points are within the electrode `square`. Stimulation positions are consistent with after probe shifts and rotations:

```
sqmea_r.move([0,500,0])
sqmea_r.rotate([1, 0, 0], 45)

# compute electric potential and return stimulation points
vp, stim_points = sqmea_r.compute_field(reduced_points, return_stim_points=True)
_  = plt.plot(stim_points[:, 1], stim_points[:, 2], '*')
```

(continues on next page)

(continued from previous page)

```
_ = plt.axis('equal')
```



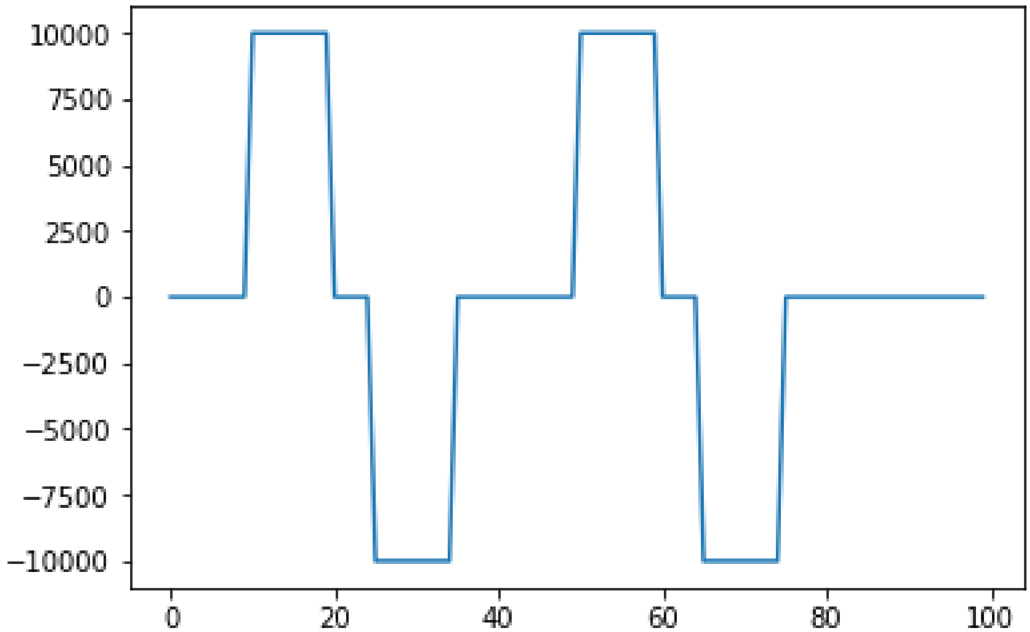
The effect of the electrode finite size on the electric potential in proximity of the stimulation site is shown in the `MEA_plotting` section.

3.3.5 Temporal dynamics

So far, we used *static* currents, but the effect of current dynamics can be very important for exciting neurons. Temporal vatying currents can be easily implemented with the MEAutility package.

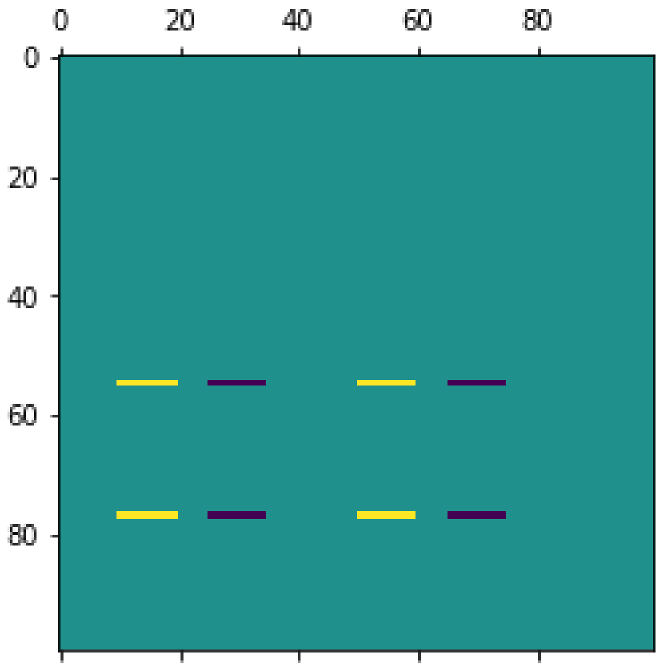
Let's instantiate a new MEA object and set a monopolar biphasic source with 2 pulses:

```
sqmea = MEA.return_mea('SqMEA-10-15um')
center_electrode = sqmea.dim[0] // 2

ntimes = 100
bipolar_source = np.zeros(ntimes)
bipolar_source[10:20] = 10000
bipolar_source[25:35] = -10000
bipolar_source[50:60] = 10000
bipolar_source[65:75] = -10000

_ = plt.plot(bipolar_source)
```




```
# the current can be set directly accessing the electrode current
sqmea[center_electrode][center_electrode].current = bipolar_source

# OR

# using set_current() (get_linear_id returns the index of the matrix in the linear array)
sqmea.set_current(sqmea.get_linear_id(center_electrode+2, center_electrode+2), bipolar_
↪source)
```

```
_ = plt.matshow(sqmea.currents)
```

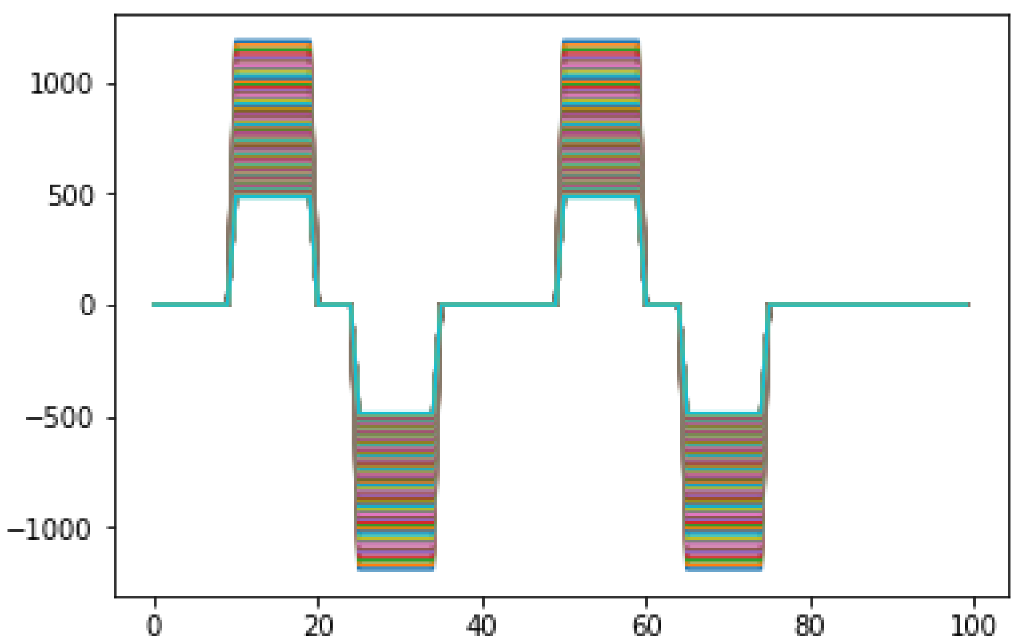


Computing the electrical potential returns un array when currents have temporal dynamics:

```
vp = sqmea.compute_field(points[:100])
```

```
print(vp.shape)
_ = plt.plot(vp.T)
```

```
(100, 100)
```



As expected the potential becomes lower moving further away from the probe!

3.4 MEA plotting

This notebook shows some plotting routines implemented in the MEAutility package.

```
import MEAutility as MEA
import matplotlib.pylab as plt
import numpy as np
%matplotlib notebook
```

3.4.1 2D plotting

As usual, let's first define some MEA objects:

```
sqmea = MEA.return_mea('SqMEA-10-15um')
neuronexus = MEA.return_mea('Neuronexus-32')
neuropixels = MEA.return_mea('Neuropixels-128')
```

The `plot_probe()` function plots the probe in 2D. The axis is returned and an existing axis can be passed with the `ax` argument. Here are some examples:

```
MEA.plot_probe(neuropixels)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4d5a79f630>
```

```
fig, ax1 = plt.subplots()
ax1 = MEA.plot_probe(neuronexus, ax=ax1, type='shank')
_ = ax1.axis('off')
```

`plot_probe()` always plots the probe along its main axes:

```
neuronexus.rotate([1,0,0], 45)
ax1 = MEA.plot_probe(neuronexus, type='shank')
_ = ax1.axis('off')
```

```
_ = MEA.plot_probe(sqmea, type='planar', xlim=[-400,400], ylim=[-200,200])
```

To visualize the stimulating currents, one can use the `color_currents` parameter:

```
sqmea.set_random_currents()
ax = MEA.plot_probe(sqmea, color_currents=True)
```

```
# colormap can be changed
ax = MEA.plot_probe(sqmea, color_currents=True, cmap='hot')
```

3.4.2 3D plotting

The function `plot_probe_3d` allows to plot MEA objects in 3d axes. The plots reflect the current position and rotation of the MEA.

```
neuronexus = MEA.return_mea('Neuronexus-32')
_ = MEA.plot_probe_3d(neuronexus)
```

```
neuronexus.rotate([1,0,0], 45)
_ = MEA.plot_probe_3d(neuronexus)
```

```
neuronexus.set_random_currents()
_ = MEA.plot_probe_3d(neuronexus, color_currents=True, cmap='jet')
```

```
ax = MEA.plot_probe_3d(neuronexus, color_currents=True, cmap='jet',
                       xlim=[-100,100], ylim=[-100,100], zlim=[-100,100])
_ = ax.axis('off')
```

3.4.3 Electric potential images

The functions `plot_v_image()` and `plot_v_surf()` allows the user to plot potential images on a plane. The plane can be defined with the `plane` argument and boundaries can be given with `x_bound`, `y_bound`, and `z_bound` arguments (e.g. if `plane` is `xz`, `x_bound` and `z_bound` are required). The offset on the other direciotn (i.e. `y` when plane is `xz`) is controlled by the `offet` parameter.

```
sqmea = MEA.return_mea('SqMEA-10-15um')
sqmea.points_per_electrode = 1
sqmea.reset_currents()
sqmea[0][0].current = 10000
sqmea[5][0].current = 10000
sqmea[0][7].current = 10000
```

```
_ = MEA.plot_v_image(sqmea, y_bound=[-100, 100], z_bound=[-100, 100], plane='yz',␣
↪offset=10)
```

With `plot_v_image` we can show the effect of electrodes of finite sizes:

```
print(sqmea[0][0].position)
```

```
[  0.  -67.5 -67.5]
```

```
fig, axes = plt.subplots(1, 2)
# points per electrode = 1
sqmea.points_per_electrode = 1
_, v1 = MEA.plot_v_image(sqmea, y_bound=[-55, -80], z_bound=[-55, -80], offset=2,
                         npoints=30, plane='yz', ax=axes[0])

# points per electrode = 100
sqmea.points_per_electrode = 100
_, v100 = MEA.plot_v_image(sqmea, y_bound=[-55, -80], z_bound=[-55, -80], offset=2,
                           npoints=30, plane='yz', ax=axes[1])
```

The finite size results in a *squarer* electric potential in proximity of the electrode!

```
fig = plt.figure()
ax1 = fig.add_subplot(1, 2, 1, projection='3d')
ax2 = fig.add_subplot(1, 2, 2, projection='3d')

sqmea.points_per_electrode = 1
_ = MEA.plot_v_surf(sqmea, v_plane=v1, y_bound=[-55, -80], z_bound=[-55, -80], offset=10,
                           npoints=30, plane='yz', ax=ax1)

_ = MEA.plot_v_surf(sqmea, v_plane=v100, y_bound=[-55, -80], z_bound=[-55, -80],␣
↪offset=10,
                           npoints=30, plane='yz', ax=ax2)
```

```
sqmea.points_per_electrode = 1
sqmea[0][0].current = 10000
ax, v = MEA.plot_v_surf(sqmea, y_bound=[-100, 100], z_bound=[-100, 100],
                        plane='yz', plot_plane='yz', offset=30, distance=200)
MEA.plot_probe_3d(sqmea, ax=ax, xlim=[-500, 500], color_currents=True)
```

```
<matplotlib.axes._subplots.Axes3DSubplot at 0x7f4d5829be48>
```

```
sqmea.rotate([0,1,0], 90)
print(sqmea.main_axes)
```

```
[[0. 1. 0.]
 [1. 0. 0.]]
```

```
ax, v = MEA.plot_v_surf(sqmea, x_bound=[-100, 100], y_bound=[-100, 100],
                        plane='xy', plot_plane='xy', offset=30, distance=30)
MEA.plot_probe_3d(sqmea, ax=ax, xlim=[-100, 100], zlim=[-100, 300], color_currents=True,␣
↪type='planar')
```

```
<matplotlib.axes._subplots.Axes3DSubplot at 0x7f4d4aed14e0>
```

```
sqmea.rotate([0,1,0], -90)
sqmea.rotate([0,0,1], -90)
print(sqmea.main_axes)
```

```
[[1. 0. 0.]
 [0. 0. 1.]]
```

```
ax, v = MEA.plot_v_surf(sqmea, x_bound=[-100, 100], z_bound=[-100, 100],
                        plane='xz', plot_plane='xz', offset=30, distance=100)
MEA.plot_probe_3d(sqmea, ax=ax, color_currents=True,)
```

```
<matplotlib.axes._subplots.Axes3DSubplot at 0x7f4d582e0710>
```

3.4.4 Plot signal traces

```
# fake noise signal
signals = np.random.randn(sqmea.number_electrodes, 10000)
_ = MEA.plot_mea_recording(signals, sqmea, lw=0.1)
```

3.4.5 Animations

```
# %matplotlib notebook
# from IPython.display import HTML

# anim = MEA.play_mea_recording(signals, sqmea, 1000, interval =100, lw=0.1)
# HTML(anim.to_jshtml())
```

3.5 Module `MEAutility.core`

3.5.1 class `Electrode`

class core.**Electrode**(*position=None*, *normal=None*, *current=0*, *sigma=0.3*, *npoints=1*, *shape='square'*, *size=5*, *main_axes=None*, *model='inf'*, *dist_limit=2*)

Bases: `object`

are_points_inside(*points*)

compute_mapping(*points*, *npoints=1*, *model='inf'*, *seed=None*)

Computes forward mapping between electrode stimulation position and an array of points

Parameters

- **points** (*np.array*) – 3D points to compute mapping
- **npoints** (*int*) – Number of random points within electrode
- **model** (*str*) – 'inf' or 'semi'
- **seed** (*int*) – Random seed

field_contribution(*points*, *npoints=1*, *model='inf'*, *seed=None*)

Computes extracellular potential arising from stimulating current

Parameters

- **points** (*np.array or list*) – It can be a single 3d point or an array/list of positions (n_points, 3)
- **npoints** (*int*) – Number of points to split the current within the electrode. It is used to simulate the spatial extent of the electrode
- **model** (*str*) – 'inf' or 'semi'. If 'semi' the method of images is used and the electrode is modeled as lying on semi-infinite plane
- **main_axes** (*np.array*) – The main axes indicating the plane on which the electrode lies (2 x 3).
- **seed** (*int*) – Seed for drawing random points

Returns
potential – Potential computed at the indicated point (float) or points (np.array)

Return type
float or np.array

get_n_points(*npoints*)

set_mapping(*mapping*, *points*)
Sets mapping for the electrode with the provided points

Parameters

- **mapping** (*np.array*) – Mapping
- **points** (*np.array*) – 3D points to compute mapping

3.5.2 class MEA

class core.MEA(*positions*, *info*, *normal=None*, *points_per_electrode=1*, *sigma=0.3*)
Bases: object

center()
Centers the MEA to (0,0,0)

compute_field(*points*, *return_stim_points=False*, *points_per_electrode=1*, *seed=None*)
Computes extracellular potential from stimulating currents.

Parameters

- **points** (*list or np.array*) – Single point (3) or array of points (n_points, 3) to compute potential
- **return_stim_points** (*bool*) – If True, stimulating points are returned
- **points_per_electrode** (*int*) – Number of stimulation points per electrode
- **seed** (*int*) – Seed for random drawing of stimulation points

Returns

- **vp** (*float or np.array*) – Computed potential at the specified point (float) or points (np.array)
- **stim_points** (*np.array*) – Stimulating point(s)

property currents

get_closest_electrode_idx(*pos*)
Returns index of electrode closest to the specified 3D point.

Parameters
pos (*list or np.array*) – Array of 3d position

Returns
index – The index of the closest electrode

Return type
int

get_random_points_inside(*npoints*, *seed=None*)

Returns 'npoints' random positions inside each electrode

Parameters

- **npoints** (*int*) – Number of points for each electrode
- **seed** (*int*) – Random seed

Returns
random_points – Array with (n_electrodes, n_points, 3) dimensions with random positions

Return type
np.array

load_currents(*filename*)

Load currents from file

Parameters
filename (*str*) – Path to file

move(*vector*)

Moves the MEA in 3d

Parameters
vector (*list or np.array*) – 3d array with x, y, z shifts

property positions

reset_currents(*amp=0*)

Resets all currents

Parameters
amp (*float*) – Amplitude to reset currents (default=0)

rotate(*axis*, *theta*)

Rotates the MEA in 3d along a specified axis

Parameters

- **axis** (*np.array*) – Rotation axis
- **theta** (*float*) – Angle in degrees counterclock-wise

save_currents(*filename*)

Saves MEA currents to file in .npy format

Parameters
filename (*str*) – Path to file

set_current(*el_id*, *current_value*)

Sets MEA current

Parameters

- **el_id** (*int*) – Electrode index
- **current_value** (*float or array*) – Current value for the specified electrode

set_current_pulses(*el_id*, *amp1*, *width1*, *interpulse*, *t_stop*, *dt*, *biphasic=False*, *interphase=None*, *n_pulses=None*, *n_bursts=None*, *interburst=None*, *amp2=None*, *width2=None*, *t_start=None*)

Computes and sets pulsed currents on specified electrode.

Parameters

- **el_id** (*int*) – Electrode index
- **amp1** (*float*) – Amplitude of stimulation. If 'biphasic', amplitude of the first phase.
- **width1** (*float*) – Duration of the pulse in ms. If 'biphasic', duration of the first phase.
- **interpulse** (*float*) – Interpulse interval in ms
- **t_stop** (*float*) – Stop time in ms
- **dt** (*float*) – Sampling period in ms
- **biphasic** (*bool*) – If True, biphasic pulses are used
- **amp2** (*float*) – Amplitude of second phase stimulation (when 'biphasic'). If None, this amplitude is the opposite of 'amp1'
- **width2** (*float*) – Duration of the pulse of the second phase in ms (when 'biphasic'). If None, it is the same as 'width1'
- **interphase** (*float*) – For biphasic stimulation, duration between two phases in ms
- **n_pulses** (*int*) – Number of pulses. If 'interburst' is given, this indicates the number of pulses in a burst
- **interburst** (*float*) – Inter burst interval in ms
- **n_bursts** (*int*) – Total number of burst (if None, no limit is used).
- **t_start** (*float*) – Start of the stimulation in ms

Returns

- **current** (*np.array*) – Array of current values
- **t_ext** (*np.array*) – timestamps of computed currents

set_currents(*current_values*)

Sets MEA currents

Parameters
current_values (*np.array*) – Current values. Either (n_elec) or (n_elec, n_timepoints)

set_electrodes(*electrodes*)

Sets MEA electrodes

Parameters
electrodes (*list*) – List of Electrode objects

set_mapping(*mapping*, *points*)

Sets forward mapping for all electrodes

Parameters

- **mapping** (*np.array*) – Mapping values (n_electrodes, n_points, n_points_per_electrode)
- **points** (*np.array*) – Array of 3D points for which the forward mapping is computed

set_random_currents(*mean=0*, *sd=1000*)

Parameters

- **mean**
- **sd**

3.5.3 class RectMEA

`class core.RectMEA(positions, info)`

Bases: *MEA*

get_current_matrix()

Returns MEA currents as a matrix

Returns
current_matrix – 2D array with MEA currents

Return type
np.array

get_electrode_matrix()

Returns MEA electrode matrix

Returns
electrode_matrix – 2D array with MEA electrode objects

Return type
np.array

get_linear_id(*i*, *j*)

Returns linear index for position i, j

Parameters

- **i** (*int*) – Row index
- **j** (*int*) – Column index

Returns
index – Linear index

Return type
int

set_current_matrix(*currents*)

Sets MEA currents with a matrix

Parameters
currents (*np.array*) – 2D array with MEA currents

3.5.4 function return_mea

`core.return_mea(electrode_name=None, info=None)`

Returns MEA object.

Parameters

- **electrode_name** (*str*) – Probe name
- **info** (*dict*) – Probe info (alternative to electrode_name)

Returns
mea – The MEA object

Return type
MEA

3.5.5 function `return_mea_info`

`core.`**`return_mea_info`**(*electrode_name=None*)

Returns probe information.

Parameters
`electrode_name` (*`str`*) – Probe name

Returns
info – Dictionary with electrode info

Return type
dict

3.5.6 function `return_mea_list`

`core.`**`return_mea_list`**()

Returns available probe models.

Returns
probes – List of available probe_names

Return type
list

3.5.7 function `add_mea`

`core.`**`add_mea`**(*mea*, *electrode_name=None*)

Adds the mea probe defined by the yaml file in the installation folder.

Parameters

- **`mea`** (*`str or MEA object`*) – Path to yaml file or MEA instance
- **`electrode_name`** (*`str`*) – Probe name (if not in mea.info)

3.5.8 function `remove_mea`

`core.`**`remove_mea`**(*mea_name*)

Removes the mea from the installation folder.

Parameters
`mea_name` (*`str`*) – The probe name to be removed

3.6 Module `MEAutility.plotting`

3.6.1 function `plot_probe`

`plotting.`**`plot_probe`**(*mea*, *ax=None*, *xlim=None*, *ylim=None*, *color_currents=False*, *top=None*, *bottom=None*, *cmap='viridis'*, *type='shank'*, *alpha_elec=0.7*, *alpha_prb=0.3*)

Plots probe in 2d.

Parameters

- **mea** (*MEA object*) – The MEA to be plotted
- **ax** (*matplotlib axis*) – The axis to plot on
- **xlim** (*list*) – Limits for x axis
- **ylim** (*list*) – Limits for y axis
- **color_currents** (*bool*) – If True currents are color-coded
- **top** (*int*) – The length of the probe in the top direction
- **bottom** (*int*) – The length of the probe in the bottom direction
- **cmap** (*matplotlib colormap*) – The colormap to use for currents
- **type** (*str*) – 'shank' or 'plane' type
- **alpha_elec** (*float*) – Alpha value for electrodes
- **alpha_prb** (*float*) – Alpha value for probe

Returns
ax – The output axis

Return type
matplotlib axis

3.6.2 function plot_probe_3d

plotting.**plot_probe_3d**(*mea*, *ax=None*, *xlim=None*, *ylim=None*, *zlim=None*, *top=None*, *bottom=None*, *type='shank'*, *color_currents=False*, *cmap='viridis'*, *alpha_elec=0.7*, *alpha_prb=0.3*)

Plots probe in 3d.

mea: MEA object
The MEA to be plotted

ax: matplotlib axis
The axis to plot on

xlim: list
Limits for x axis

ylim: list
Limits for y axis

ylim: list
Limits for z axis

color_currents: bool
If True currents are color-coded

top: int
The length of the probe in the top direction

bottom: int
The length of the probe in the bottom direction

cmap: matplotlib colormap
The colormap to use for currents

type: str
'shank' or 'plane' type

alpha_elec: float
Alpha value for electrodes

alpha_prb: float
Alpha value for probe

Returns
ax – The output axis

Return type
matplotlib axis

3.6.3 function `plot_cylinder_3d`

`plotting.plot_cylinder_3d`(*bottom*, *direction*, *length*, *radius*, *color='k'*, *alpha=0.5*, *ax=None*, *xlim=None*, *ylim=None*, *zlim=None*)

Plots cylinder in 3d.

Parameters

- **bottom** (*list or np.array*) – 3D point of the bottom of the cylinder
- **direction** (*list or np.array*) – 3D direction of the cylinder
- **length** (*float*) – Length of the cylinder in um
- **radius** (*float*) – Radius of the cylinder in um
- **color** (*matplotlib color*) – Color of the cylinder
- **alpha** (*float*) – Alpha vlue of the cylinder
- **ax** (*matplotlib axis*) – The axis to plot on
- **xlim** (*list*) – Limits for x axis
- **ylim** (*list*) – Limits for y axis
- **ylim** – Limits for z axis

Returns
ax – The output axis

Return type
matplotlib axis

3.6.4 function `plot_v_image`

`plotting.plot_v_image`(*mea*, *v_plane=None*, *x_bound=None*, *y_bound=None*, *z_bound=None*, *offset=0*, *plane='yz'*, *npoints=30*, *ax=None*, *cmap='viridis'*, ***kwargs*)

Plots voltage image generated by mea currents.

Parameters

- **mea** (*MEA object*) – The MEA to be plotted
- **v_plane** (*np.array*) – The voltage values to be plotted (default=None)

- **x_bound** (*list*) – X boundaries to compute grid (if plane has 'x' dimension)
- **y_bound** (*list*) – Y boundaries to compute grid (if plane has 'y' dimension)
- **z_bound** (*list*) – Z boundaries to compute grid (if plane has 'z' dimension)
- **offset** (*float*) – Offset in um from probe plane to compute electrical potential
- **plane** (*str*) – 'xy', 'yz', 'xz'
- **npoints** (*int*) – Number of points in grid
- **ax** (*matplotlib axis*) – The axis to plot on
- **cmap** (*matplotlib colormap*) – The colormap to use for currents
- **kwargs** (*keyword args*) – Other arguments for matshow function

Returns

- **ax** (*matplotlib axis*) – The output axis
- **v_grid** (*np.array*) – The voltage image

3.6.5 function `plot_v_surf`

`plotting.plot_v_surf`(*mea*, *v_plane=None*, *x_bound=None*, *y_bound=None*, *z_bound=None*, *offset=0*, *plane='yz'*, *plot_plane=None*, *npoints=30*, *ax=None*, *cmap='viridis'*, *alpha=0.8*, *distance=30*)

Plots voltage image generated by mea currents.

Parameters

- **mea** (*MEA object*) – The MEA to be plotted
- **v_plane** (*np.array*) – The voltage values to be plotted (default=None)
- **x_bound** (*list*) – X boundaries to compute grid (if plane has 'x' dimension)
- **y_bound** (*list*) – Y boundaries to compute grid (if plane has 'y' dimension)
- **z_bound** (*list*) – Z boundaries to compute grid (if plane has 'z' dimension)
- **offset** (*float*) – Offset in um from probe plane to compute electrical potential
- **plane** (*str*) – 'xy', 'yz', 'xz'
- **plot_plane** (*str*) – Plane to plot surf
- **npoints** (*int*) – Number of points in grid
- **ax** (*matplotlib axis*) – The axis to plot on
- **cmap** (*matplotlib colormap*) – The colormap to use for currents
- **alpha** (*float*) – Aplha value for surf plot
- **distance** (*float*) – Distance of surf plot from mea

Returns

- **ax** (*matplotlib axis*) – The output axis
- **v_grid** (*np.array*) – The voltage image

3.6.6 function `plot_mea_recording`

`plotting.`**`plot_mea_recording`**(*signals*, *mea*, *colors=None*, *ax=None*, *spacing=None*, *scalebar=False*, *time=None*, *vscale=None*, *hide_axis=True*, *axis_equal=False*, *lw=1*, *alpha=1*, *channels=None*, ***ax_kwargs*)

Plots mea signals at electrode locations.

Parameters

- **signals** (*np.array*) – The signals to plot. Can be 2D (single signals) or 3D (multiple signals)
- **mea** (*MEA object*) – The MEA to be plotted
- **colors** (*matplotlib colors*) – The color or colors to use
- **lw** (*float*) – Line width of the lines
- **ax** (*matplotlib axis*) – The axis to plot on
- **spacing** (*float*) – The spacing in the x-direction
- **scalebar** (*bool*) – If True, a scale bar is plotted
- **time** (*float*) – If scalebar is True, the time of the scalebar
- **vscale** (*float*) – The scale to use for the signal (default .5 times the maximum signal)
- **hide_axis** (*bool*) – If True (default), axis are hidden
- **axis_equal** (*bool*) – If True, axis aspect is set to 'equal'

Returns

ax – The output axis

Return type

matplotlib axis

3.6.7 function `play_mea_recording`

`plotting.`**`play_mea_recording`**(*signals*, *mea*, *fs*, *window=1*, *step=0.1*, *colors=None*, *lw=1*, *ax=None*, *spacing=None*, *vscale=None*, *repeat=False*, *interval=10*, *hide_axis=True*)

Plays animation of the signals at electrode locations.

Parameters

- **signals** (*np.array*) – The signals to plot. Can be 2D (single signals) or 3D (multiple signals)
- **mea** (*MEA object*) – The MEA to be plotted
- **fs** (*float*) – Sampling frequency in Hz
- **window** (*float*) – The sliding window in seconds
- **step** (*float*) – The step of the sliding window in seconds
- **colors** (*matplotlib colors*) – The color or colors to use
- **lw** (*float*) – Line width of the lines
- **ax** (*matplotlib axis*) – The axis to plot on
- **spacing** (*float*) – The spacing in the x-direction

- **vscale** (*float*) – The scale to use for the signal (default .5 times the maximum signal)
- **repeat** (*bool*) – If True (default=False), animation is repeated
- **interval** (*int*) – Interval between frames in ms
- **hide_axis** (*bool*) – If True (default), axis are hidden

Returns
anim – The output animation

Return type
matplotlib animation

CHAPTER

FOUR

CONTACT

If you have questions or comments, contact Alessio Buccino: alessiob@ifi.uio.no

PYTHON MODULE INDEX

INDEX