magpylib Documentation

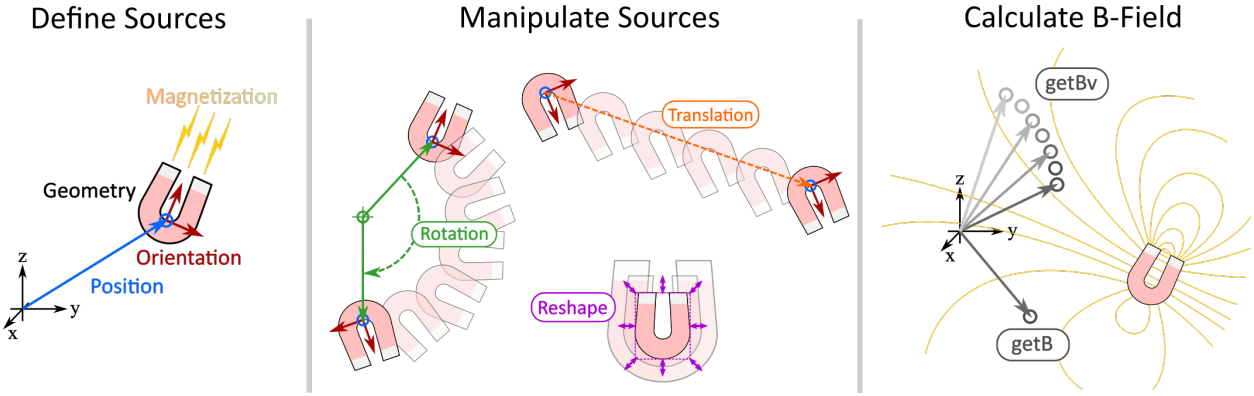
Release 2.3.0-beta

Michael Ortner

Apr 19, 2021

Content:

- Free Python package for calculating magnetic fields of magnets, currents and moments (sources).
- Provides convenient methods to create, geometrically manipulate, group and visualize assemblies of sources.
- The magnetic fields are determined from underlying analytical solutions which results in fast computation times and requires little computation power.
- For high performance computation (e.g. for multivariate parameter space analysis) all functions are also available in vectorized form.



CHAPTER 1

Quickstart

Install magpylib with pip: `>> pip install magpylib`.

Example:

Run this simple code to calculate the magnetic field of a cylindrical magnet.

```
from magpylib.source.magnet import Cylinder
s = Cylinder( mag = [0,0,350], dim = [4,5])
print(s.getB([4,4,4]))

# Output: [ 5.08641867  5.08641867 -0.60532983]
```

In this example the cylinder axis is parallel to the z-axis. The diameter and height of the magnet are 4 millimeter and 5 millimeter respectively and the magnet position (=geometric center) is in the origin. The magnetization / remanence field is homogeneous and points in z-direction with an amplitude of 350 millitesla. Finally, the magnetic field **B** is calculated in units of millitesla at the posititon *[4,4,4]* millimeter.

Example:

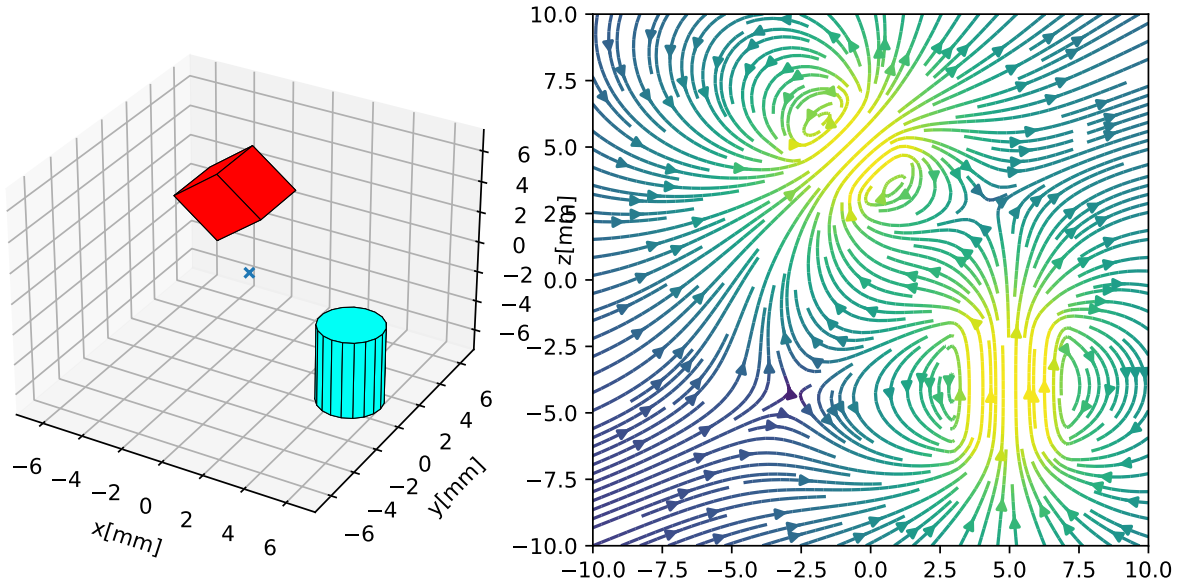
The following code calculates the combined field of two magnets. They are geometrically manipulated, the system geometry is displayed together with the field in the xz-plane.

```
# imports
import numpy as np
import matplotlib.pyplot as plt
import magpylib as magpy

# create figure
fig = plt.figure(figsize=(8,4))
ax1 = fig.add_subplot(121, projection='3d')  # this is a 3D-plotting-axis
ax2 = fig.add_subplot(122)                   # this is a 2D-plotting-axis

# create magnets
s1 = magpy.source.magnet.Box(mag=[0,0,600],dim=[3,3,3],pos=[-4,0,3])
s2 = magpy.source.magnet.Cylinder(mag=[0,0,500], dim=[3,5])
```

(continues on next page)

(continued from previous page)

```
# manipulate magnets
s1.rotate(45,[0,1,0],anchor=[0,0,0])
s2.move([5,0,-4])

# create collection
c = magpy.Collection(s1,s2)

# display system geometry on ax1
magpy.displaySystem(c,subplotAx=ax1,suppress=True)

# calculate B-field on a grid
xs = np.linspace(-10,10,29)
zs = np.linspace(-10,10,29)
Bs = np.array([[c.getB([x,0,z]) for x in xs] for z in zs])

# display field in xz-plane using matplotlib
X,Z = np.meshgrid(xs,zs)
U,V = Bs[:,:,0], Bs[:,:,2]
ax2.streamplot(X, Z, U, V, color=np.log(U**2+V**2),density=2)

plt.show()
```



More examples can be found in the Examples Section.

Technical details can be found in the *Documentation v2.0.1b* .

1.1 Documentation v2.0.1b

The idea behind magpylib is to provide a simple and easy-to-use interface for computing the magnetic field of magnets, currents and moments. The computations are based on (semi-)analytical solutions found in the literature, discussed in the *[WIP] Physics & Computation* section.

1.1.1 Contents

1.1.2 Package Structure

The top level of magpylib contains the sub-packages and `source`, `vector` and `math`, the classes `magpylib.Collection` and `magpylib.Sensor` as well as the function `magpylib.displaySystem()`.

1. The **source module** includes a set of classes that represent physical sources of the magnetic field (e.g. permanent magnets).
2. The **vector module** includes functions for performance computation.
3. The **math module** contains practical functions for working with angle-axis rotations and transformation to and from Euler angle representation.
4. The **Collection class** is used to group sources and for common manipulation.
5. The **Sensor class** represents a 3D magnetic sensor.
6. The **displaySystem function** is used to create a graphical output of the system geometry.

Check out the genindex and modindex for more details.

1.1.3 Units and IO types

In magpylib all inputs and outputs are made in the physical units of

- **Millimeter** for lengths
- **Degree** for angles
- **Millitesla** for magnetization/remanence, magnetic moment and magnetic field,
- **Ampere** for currents.

Unless specifically state otherwise in the docstring (see vector package), **scalar input** can be of `int` or `float` type and **vector/matrix input** can be given either in the form of a `list`, as a `tuple` or as a `numpy.array`.

The library output and all object attributes are either of `numpy.float64` or `numpy.array64` type.

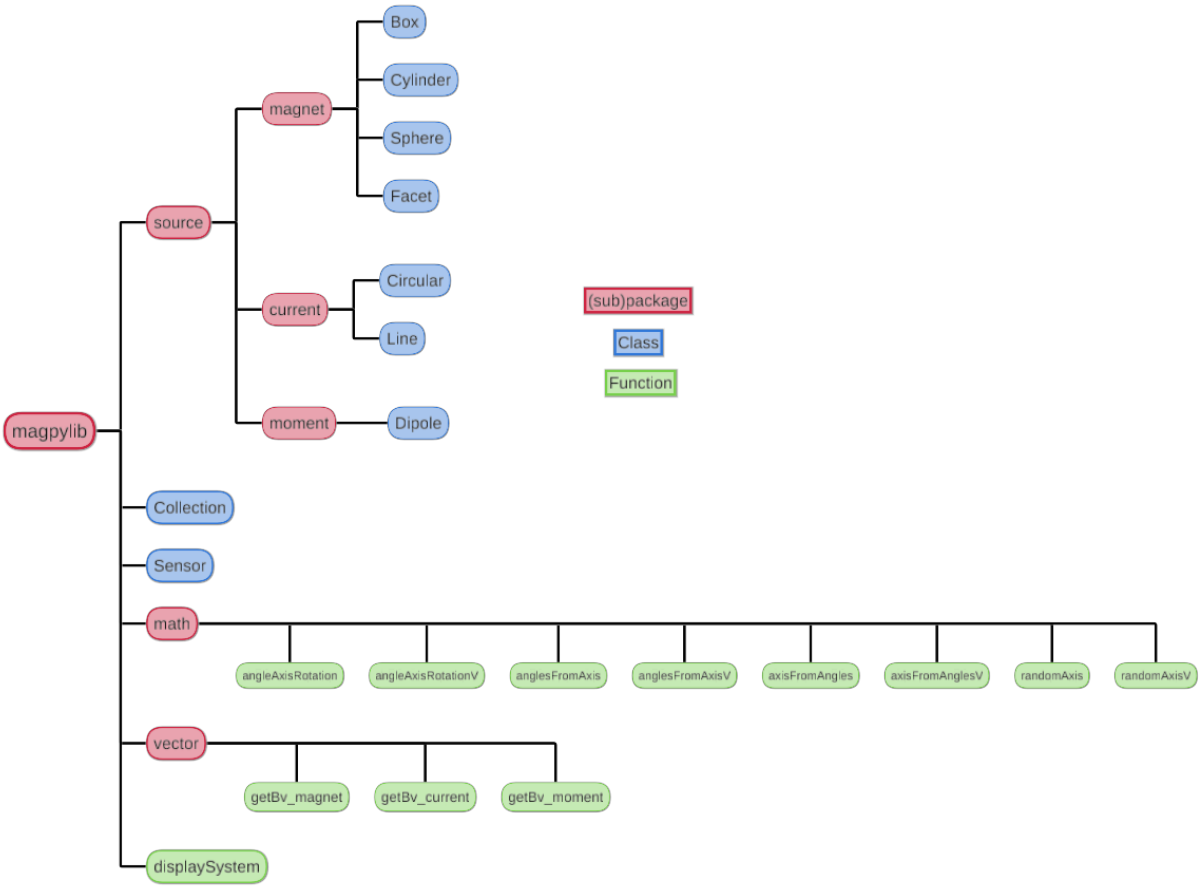


Fig. 1: **Figure:** Outline of library structure.

1.1.4 The Source Class

This is the core class of the library. The idea is that source objects represent physical magnetic field sources in Cartesian three-dimensional space. The following source types are currently implemented,

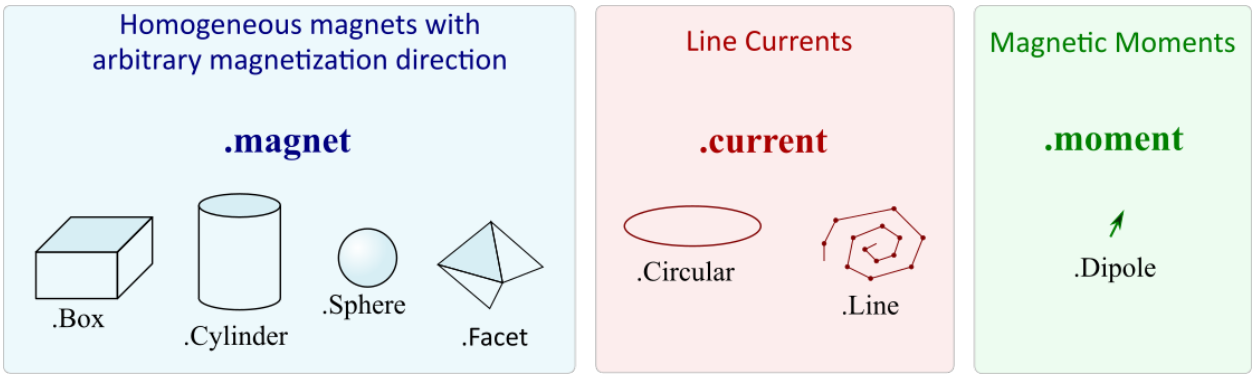


Fig. 2: **Figure:** Source types currently available in magpylib.

All source objects share various attributes and methods. The attributes characterize the source (e.g. position, orientation, dimension) while the methods can be used for geometric manipulation and for calculating the magnetic field. The figure below gives a graphical overview.

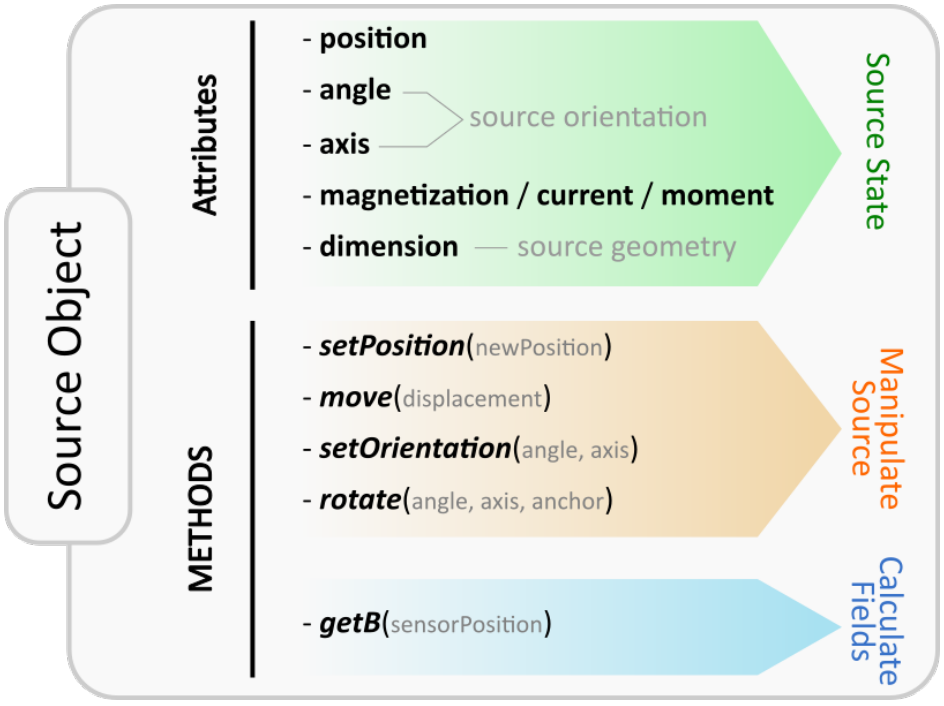


Fig. 3: **Figure:** Illustration of attributes and methods of the source class objects.

Position and Orientation

The most fundamental properties of a source object `s` are position and orientation which are represented through the attributes `s.position` (*arr3*), `s.angle` (*float*) and `s.axis` (*arr3*). At source initialization, if no values are

specified, the source object is initialized by default with `position=(0,0,0)`, and **init orientation** defined to be `angle=0` and `axis=(0,0,1)`.

Due to their different nature each source type is characterized by different attributes. However, in general the `position` attribute refers to the position of the geometric center of the source. The **init orientation** generally defines sources standing upright oriented along the Cartesian coordinates axes, see e.g. the following image below.

An orientation of a source `s` given by (`angle`, `axis`) refers to a rotation of `s` RELATIVE TO the **init orientation** about an axis specified by the `s.axis` vector which is anchored at `s.position`. The angle of this rotation is given by the `s.angle` attribute. Mathematically, every possible orientation can be expressed by such a single angle-axis rotation. For easier use of the angle-axis rotation and transformation to Euler angles the *Math Package* provides some useful methods.

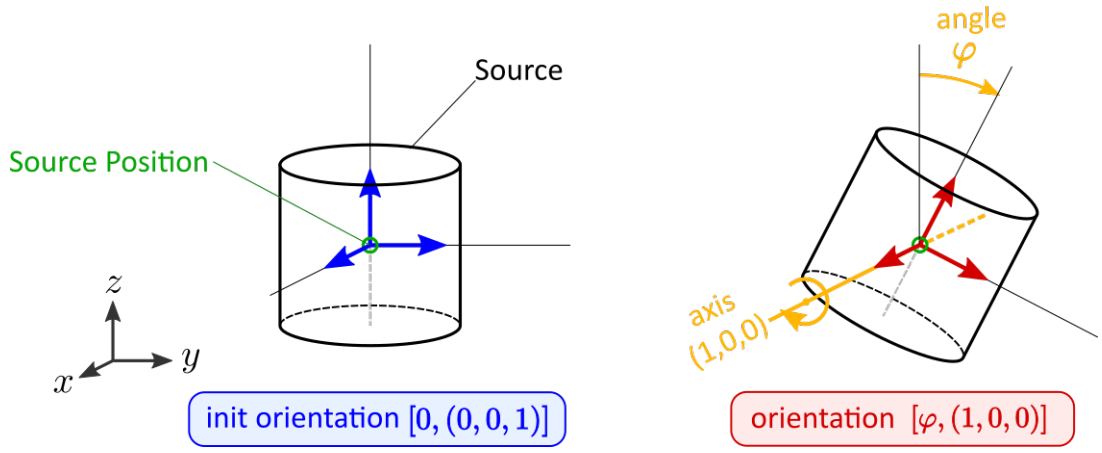


Fig. 4: **Figure:** Illustration of the angle-axis system used to describe source orientations.

Dimension & Excitation

While position and orientation have default values, a source is defined through its geometry (e.g. cylinder) and excitation (e.g. magnetization vector) which must be initialized to provide meaning.

The `dimension` input specifies the size of the source. However, as each source type requires different input parameters the format is always different:

- `Box.dimension` is a 3D array of the cuboid sides, *[a,b,c]*
- `Cylinder.dimension` is a 2D array of diameter and height, *[d,h]*
- `Sphere.dimension` is a float describing the diameter *d*
- `Facet.dimension` is a 3x3 array of the three corner vertices, *[A,B,C]*
- `Line.dimension` is a Nx3 array of N subsequent vertices, *[V1,V2,V3,...]*
- `Circular.dimension` is a float describing the diameter *d*

The excitation of a source is either the `magnetization`, the `current` or the magnetic `moment`:

- Magnet sources represent homogeneously magnetized permanent magnets (other types with radial or multipole magnetization are not implemented at this point). Such excitations are given by the `magnetization` (*vec3*) input which is always given with respect to the **init orientation** of the magnet.
- Current sources represent electrical line currents. Their excitation is simply the electrical current in units of Ampere defined by the `current` (*float*) input.
- The moment class represents a magnetic dipole moment described by the `moment` (*vec3*) input.

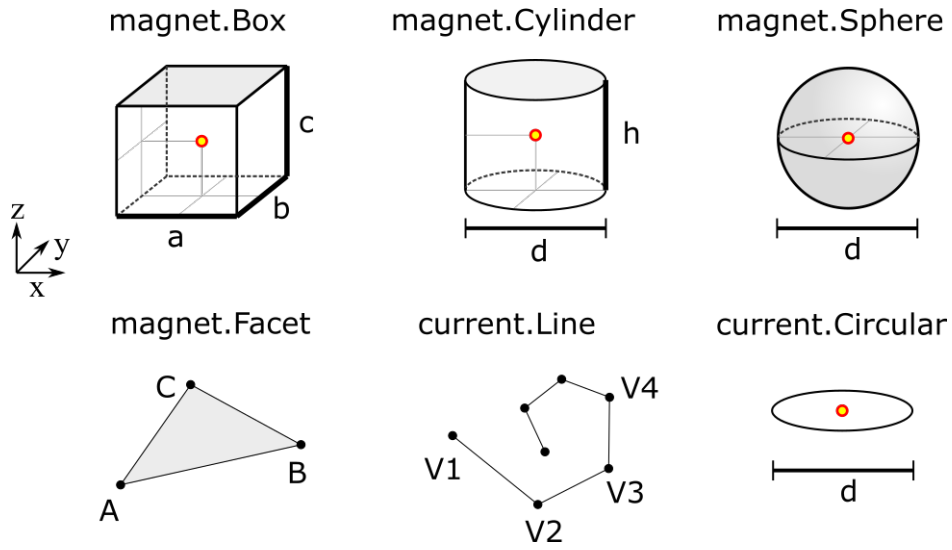


Fig. 5: **Figure:** Illustration of information given by the dimension-attribute. The source positions (typically the geometric center) are indicated by the red dot.

Detailed information about the attributes of each specific source type and how to initialize them can also be found in the respective class docstrings: *Box*, *Cylinder*, *Sphere*, *Facet*, *Line*, *Circular*, *Dipole*

Note: For convenience the attributes `magnetization`, `current`, `dimension` and `position` are initialized through the keywords `mag`, `curr`, `dim` and `pos`.

The following code shows how to initialize a source object, a D4H5 permanent magnet cylinder with diagonal magnetization, positioned with the center in the origin, standing upright with axis in z-direction.

```
from magpylib.source.magnet import Cylinder

s = Cylinder( mag = [500,0,500], # The magnetization vector in mT.
              dim = [4,5])       # dimension (diameter,height) in mm.

# no pos, angle, axis specified so default values are used

print(s.magnetization)  # Output: [500. 0. 500.]
print(s.dimension)      # Output: [4. 5.]
print(s.position)       # Output: [0. 0. 0.]
print(s.angle)          # Output: 0.0
print(s.axis)           # Output: [0. 0. 1.]
```

Methods for Geometric Manipulation

In most cases we want to move the source to a designated position, orient it in a desired way or change its dimension dynamically. There are several ways to achieve this:

At initialization:

When initializing the source we can set all attributes as desired. So instead of *moving one source around* one could create a new source for each parameter set of interest.

Manipulation after initialization:

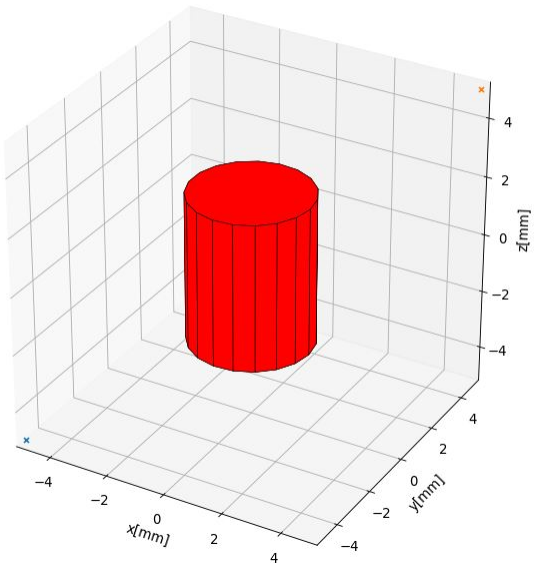


Fig. 6: **Figure:** Magnet geometry created by above code: A cylinder which stands upright with geometric center at the origin.

We initialize the source and manipulate it afterwards as desired by

1. directly setting the source attributes (e.g. `s.position=newPosition`),
2. or by using provided methods of manipulation.

The latter is often the most practical and intuitive way. To this end the source class provides a set of methods for convenient geometric manipulation. The methods include `setPosition` and `move` for translation of the objects as well as `setOrientation` and `rotate` for rotation operations. Upon application they will simply modify the source object attributes accordingly.

- `s.setPosition(newPos)`: Moves the source to the position given by the argument vector (`s.position` -> `newPos`)
- `s.move(displacement)`: Moves the source by the argument vector. (`s.position` -> `s.position + displacement`)
- `s.setOrientation(angle,axis)`: Sets a new source orientation given by the arguments. (`s.angle`-> `angle`, `s.axis` -> `axis`)
- `s.rotate(ang,ax,anchor=anch)`: Rotates the source object by the angle `ang` about the axis `ax` which passes through a position given by `anch`. As a result, source position and orientation attributes are modified. If no value for anchor is specified, the anchor is set to the object position, which means that the object rotates about itself.

The following videos show the application of the four methods for geometric manipulation.

The following example code shows how geometric operations are applied to source objects.

```
from magpylib.source.magnet import Cylinder

s = Cylinder( mag = [500,0,500], dim = [4,5])
```

(continues on next page)

(continued from previous page)

```
print(s.position)       # Output: [0. 0. 0.]

s.move([1,2,3])
print(s.position)       # Output: [1. 2. 3.]

s.move([1,2,3])
print(s.position)       # Output: [2. 4. 6.]
```

1.1.5 Calculating the Magnetic Field

To calculate the field, magpylib uses mostly analytical expressions that can be found in the literature. References, validity and discussion of these solutions can be found in the *[WIP] Physics & Computation* section. In a nutshell, the fields of the dipole and the currents are exact. The analytical magnet solutions deal with homogeneous, fixed magnetizations. For hard ferromagnets with large coercive fields like Ferrite, Neodyme and SmCo the error is typically below 2%.

There are two possibilities to calculate the magnetic field of a source object `s`:

1. Using the `s.getB(pos)` method
2. Using the `magpylib.vector` subpackage

The first method: Each source object (or collection) `s` has a method `s.getB(pos)` which returns the magnetic field generated by `s` at the position `pos`.

```
from magpylib.source.magnet import Cylinder
s = Cylinder( mag = [500,0,500], dim = [4,5])
print(s.getB([4,4,4]))

# Output: [ 7.69869084 15.407166    6.40155549]
```

Using magpylib.vector

The second method: In most cases one will be interested to determine the field for a set of positions, or for different magnet positions and orientations. While this can manually be achieved by looping `s.getB` this is computationally inefficient. For performance computation the `magpylib.vector` subpackage contains the `getBv` functions that offer quick access to VECTORIZED CODE. A discussion of vectorization, SIMD and performance is presented in the *[WIP] Physics & Computation* section.

The `magpylib.vector.getBv` functions evaluate the field for *N* different sets of input parameters. These *N* parameter sets are provided to the `getBv` functions as arrays of size *N* for each input (e.g. an *Nx3* array for the *N* different positions):

`getBv_magnet(type, MAG, DIM, POSo, POSm, [angs1,angs2,...], [AXIS1,AXIS2,...], [ANCH1,ANCH2,...])`

- `type` is a string that specifies the magnet geometry (e.g. *'box'* or *'sphere'*).
- `MAG` is an *Nx3* array of magnetization vectors.
- `DIM` is an *Nx3* array of magnet dimensions.
- `POSo` is an *Nx3* array of observer positions.
- `POSm` is an *Nx3* array of initial (before rotation) magnet positions.

- The inputs `[angs1, angs2, ...]`, `[AXIS1, AXIS2, ...]`, `[ANCH1, ANCH2, ...]` are a lists of *N*/*Nx3* arrays that correspond to angles, axes and anchors of rotation operations. By providing multiple list entries one can apply subsequent rotation operations. By ommitting these inputs it is assumed that no rotations are applied.

As a rule of thumb, `s.getB()` will be faster than `getBv` for ~5 or less field evaluations while the vectorized code will be up to ~100 times faster for 10 or more field evaluations. To achieve this performance it is critical that one follows the vectorized code paradigm (use only numpy native) when creating the `getBv` inputs. This is demonstrated in the following example where the magnetic field at a fixed observer position is calculated for a magnet that moves linearly in x-direction above the observer.

```
import magpylib as magpy
import numpy as np

# vector size: we calculate the field N times with different inputs
N = 1000

# Constant vectors, specify dtype
mag  = np.array([0,0,1000.])     # magnet magnetization
dim  = np.array([2,2,2.])        # magnet dimension
poso = np.array([0,0,-4.])       # position of observer

# magnet x-positions
xMag = np.linspace(-10,10,N)

# magpylib classic ----------------------------

Bc = np.zeros((N,3))
for i,x in enumerate(xMag):
    s = magpy.source.magnet.Box(mag,dim,[x,0,0])
    Bc[i] = s.getB(poso)

# magpylib vector -----------------------------

# Vectorizing input using numpy native instead of python loops
MAG = np.tile(mag,(N,1))
DIM = np.tile(dim,(N,1))
POSo = np.tile(poso,(N,1))
POSm = np.c_[xMag,np.zeros((N,2))]

# Evaluation of the N fields using vectorized code
Bv = magpy.vector.getBv_magnet('box',MAG,DIM,POSo,POSm)

# result ----------------------------------
# Bc == Bv    ... up to some 1e-16
# Copare classic and vector computation times using e.g. time.perf_counter()
```

More examples of vectorized code can be found in the *Vectorized Code Example* section.

Warning: The functions included in the `magpylib.vector` package do not check the input format. All input must be in the form of numpy arrays.

1.1.6 Collections

The idea behind the top level *magpylib.Collection* class is to group multiple source objects for common manipulation and evaluation of the fields.

In principle a collection `c` is simply a list of source objects that are collected in the attribute `c.sources` (*list*). Operations applied individually to the collection will be applied to all sources that are part of the collection.

Collections can be constructed at initialization by simply giving the sources objects as arguments. It is possible to add single sources, lists of multiple sources and even other collection objects. All sources are simply added to the `sources` attribute of the target collection.

With the collection kwarg `dupWarning=True`, adding multiples of the same source will be prevented, and a warning will be displayed informing the user that a source object is already in the collection's `source` attribute. Adding the same object multiple times can be done by setting `dupWarning=False`.

In addition, the collection class features methods to add and remove sources for command line like manipulation. The method `c.addSources(*sources)` will add all sources given to it to the collection `c`. The method `c.removeSource(ref)` will remove the referenced source from the collection. Here the `ref` argument can be either a source or an integer indicating the reference position in the collection, and it defaults to the latest added source in the collection.

```
import magpylib as magpy

#define some magnet objects
mag1 = magpy.source.magnet.Box(mag=[1,2,3],dim=[1,2,3])
mag2 = magpy.source.magnet.Box(mag=[1,2,3],dim=[1,2,3],pos=[5,5,5])
mag3 = magpy.source.magnet.Box(mag=[1,2,3],dim=[1,2,3],pos=[-5,-5,-5])

#create/manipulate collection and print source positions
c = magpy.Collection(mag1,mag2,mag3)
print([s.position for s in c.sources])
#OUTPUT: [array([0., 0., 0.]), array([5., 5., 5.]), array([-5., -5., -5.])]

c.removeSource(1)
print([s.position for s in c.sources])
#OUTPUT: [array([0., 0., 0.]), array([-5., -5., -5.])]

c.addSources(mag2)
print([s.position for s in c.sources])
#OUTPUT: [array([0., 0., 0.]), array([-5., -5., -5.]), array([5., 5., 5.])]
```

All methods of geometric operation, `setPosition`, `move`, `setOrientation` and `rotate` are also methods of the collection class. A geometric operation applied to a collection is directly applied to each object within that collection individually. In practice this means that a whole group of magnets can be rotated about a common pivot point with a single command.

For calculating the magnetic field that is generated by a whole collection the method `getB` is also available. The total magnetic field is simply given by the superposition of the fields of all sources.

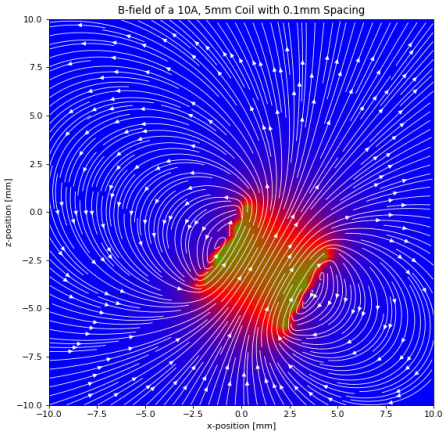


Figure: *Collection Example. Circular current sources are grouped into a collection to form a coil. The whole coil is then geometrically manipulated and the total magnetic field is calculated and shown in the xz-plane.*

1.1.7 The Sensor Class

The `getB` method will always calculate the field in the underlying canonical basis. But often one is dealing with moving and tilting sensors. For this magpylib also offers a *`magpylib.Sensor`* class.

Geometrically, a sensor object `sens` behaves just like a source object, having position and orientation attributes that can be set using the convenient methods `sens.setPosition`, `sens.move`, `sens.setOrientation` and `sens.rotate`.

To return the field of the source `s` as seen by the sensor one can use the method `sens.getB(s)`. Here `s` can be a source object or a collection of sources.

```
import magpylib as magpy

# define sensor
sens = magpy.Sensor(pos=[5,0,0])

# define source
s = magpy.source.magnet.Sphere(mag=[123,0,0],dim=5)

# determine sensor-field
B1 = sens.getB(s)

# rotate sensor about itself (no anchor specified)
sens.rotate(90,[0,0,1])

# determine sensor-field
B2 = sens.getB(s)

# print fields
print(B1)    # output: [10.25  0.  0.]
print(B2)    # output: [0. -10.25  0.]
```

1.1.8 Display System Graphically

Then top level function `displaySystem(c)` can be used to quickly check the geometry of a source-sensor-marker assembly graphically in a 3D plot. Here `c` can be a source, a list of sources or a collection. `displaySystem` uses the matplotlib package and its limited capabilities of 3D plotting which often results in bad object overlapping.

`displaySystem(c)` comes with several keyword arguments:

- `markers=listOfPos` for displaying designated reference positions. By default a marker is set at the origin. By providing `[a,b,c,'text']` instead of just a simple position vector `'text'` is displayed with the marker.
- `suppress=True` for suppressing the immediate figure output when the function is called. To do so it is necessary to deactivate the interactive mode by calling `pyplot.ioff()`. With Spyder's IPython *Inline* plotting, graphs made with *`displaySystem()`* can be blank if the `suppress=True` option is not used. Set IPython Graphics backend to *Automatic* or *Qt5* instead of *Inline* in settings/IPython console/Graphics method to address this.
- `direc=True` for displaying current and magnetization directions in the figure.
- `subplotAx=None` for displaying the plot on a designated figure subplot instance.
- `figsize=(8,8)` for setting the size of the output graphic.

The following example code shows how to use `displaySystem()`:

```
import magpylib as magpy

# create sources
s1 = magpy.source.magnet.Cylinder( mag = [1,1,0],dim = [4,5], pos = [0,0,5])
s2 = magpy.source.magnet.Box( mag = [0,0,-1],dim = [1,2,3],pos=[0,0,-5])
s3 = magpy.source.current.Circular( curr = 1, dim =10)

#create collection
c = magpy.Collection(s1,s2,s3)

# create sensors
se1 = magpy.Sensor(pos=[10,0,0])
se2 = magpy.Sensor(pos=[10,0,0])
se3 = magpy.Sensor(pos=[10,0,0])
se2.rotate(70,[0,0,1],anchor=[0,0,0])
se3.rotate(140,[0,0,1],anchor=[0,0,0])

#display system
markerPos = [(0,0,0,'origin'),(10,10,10),(-10,-10,-10)]
magpy.displaySystem(c,sensors=[se1,se2,se3],markers=markerPos)
```

1.1.9 Complex Magnet Geometries

As a result of the superposition principle complex magnet shapes and inhomogeneous magnetizations can be generated by combining multiple sources. Specifically, when two magnets overlap in this region a *vector union* applies. This means that in the overlap the magnetization vector is given by the sum of the two vectors of each object.

Geometric addition is simply achieved by placing magnets with similar magnetization next to each other. Subtraction is realized by placing a small magnet with opposite magnetization inside a large magnet. The magnetization vectors cancel in the overlap, meaning that a small volume is cut out from a larger one. An example of a hollow cylinder is given in the examples section: *Complex Magnet Shapes: Hollow Cylinder*.

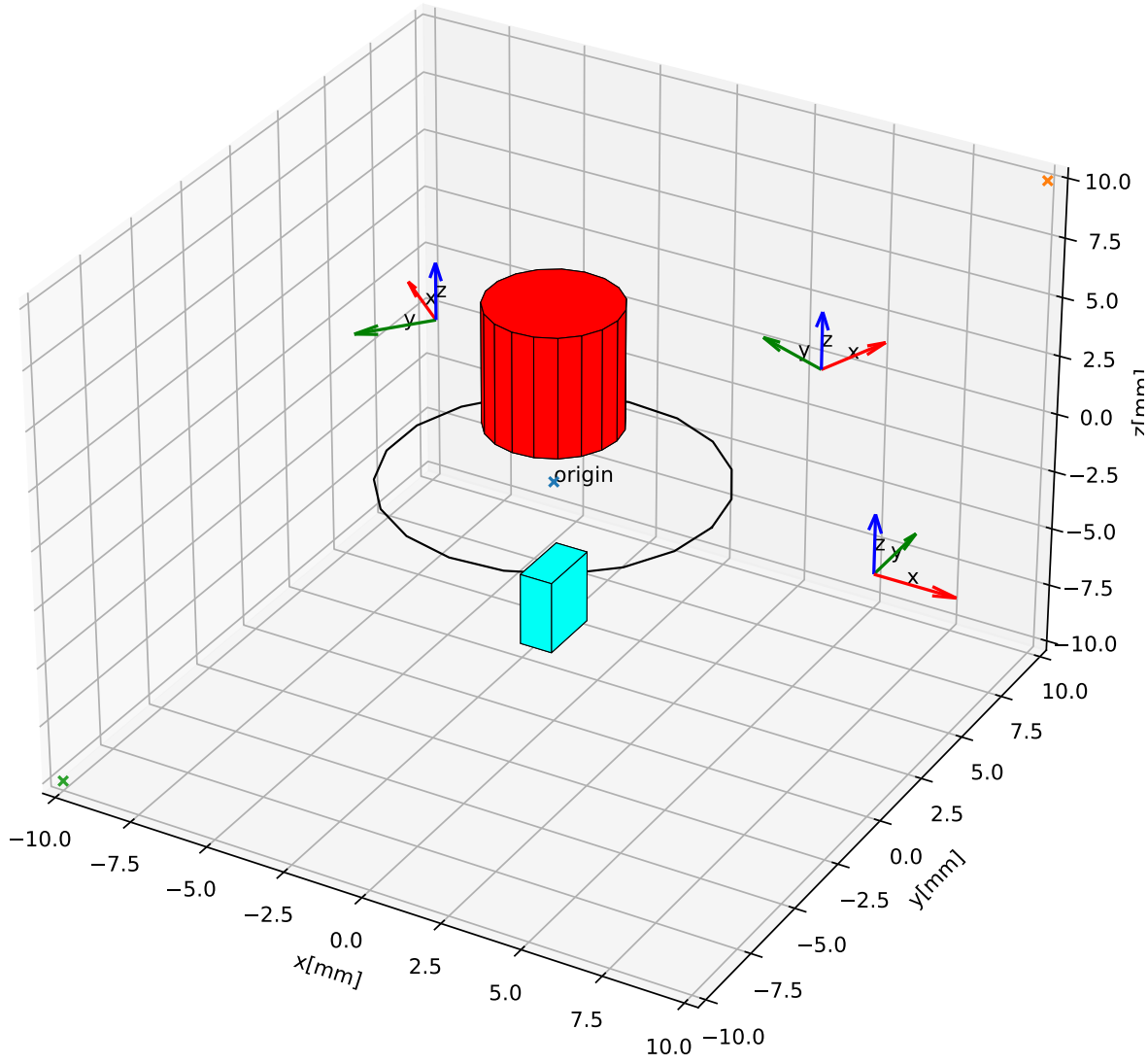


Fig. 7: **Figure:** Several magnet and sensor objects are created and manipulated. Using `displaySystem()` they are shown in a 3D plot together with some markers which allows one to quickly check if the system geometry is ok.

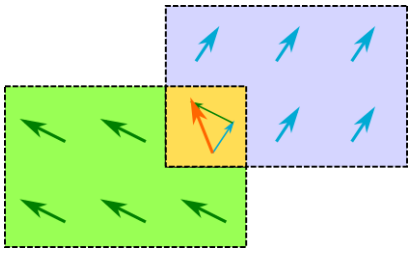


Fig. 8: **Figure:** Schematic of the *vector union* principle for magnetizations.

1.1.10 Math Package

The math package provides several practical functions that relate angle-axis (quaternion) rotations with the Euler angle rotations. All functions are also available in their vectorized versions for performance computation.

- `anglesFromAxis(axis)`: This function takes an arbitrary `axis` argument (*vec3*) and returns its orientation given by the angles `(phi, theta)` that are defined as in spherical coordinates. `phi` is the azimuth angle and `theta` is the polar angle.

```
import magpylib as magpy

angles = magpy.math.anglesFromAxis([1,1,0])
print(angles)

# Output = [45. 90.]
```

- `anglesFromAxisV(AXIS)`: This is the vectorized version of `anglesFromAxis`. It takes an *Nx3* array of axis-vectors and returns an *Nx2* array of angle pairs. Each angle pair is `(phi,theta)` which are azimuth and polar angle in a spherical coordinate system respectively.

```
import numpy as np
import magpylib as magpy

AX = np.array([[0,0,1],[0,0,1],[1,0,0]])
ANGS = magpy.math.anglesFromAxisV(AX)
print(ANGS)

# Output: [[0. 0.]  [90. 90.]  [0. 90.]])
```

- `axisFromAngles(angles)`: This function generates an axis (*vec3*) from the angle pair `angles=(phi, theta)`. Here `phi` is the azimuth angle and `theta` is the polar angle of a spherical coordinate system.

```
import magpylib as magpy

ax = magpy.math.axisFromAngles([90,90])
print(ax)

# Output: [0.0  1.0  0.0]
```

- `axisFromAnglesV(ANGLES)`: This is the vectorized version of `axisFromAngles`. It generates an *Nx3* array of axis vectors from the *Nx2* array of input angle pairs `angles`. Each angle pair is `(phi,theta)` which are azimuth and polar angle in a spherical coordinate system respectively.

```
import magpylib as magpy
import numpy as np

ANGS = np.array([[0,90],[90,180],[90,0]])
AX = magpy.math.axisFromAnglesV(ANGS)
print(np.around(AX,4))

# Output: [[1.  0. 0.]  [0. 0. -1.]  [0. 0. 1.]]
```

- `randomAxis()`: Designed for Monte Carlo simulations, this function returns a random axis (*arr3*) of length 1 with equal angular distribution.

```
import magpylib as magpy

```

(continues on next page)

(continued from previous page)

```
ax = magpy.math.randomAxis()
print(ax)

# Output: [-0.24834468  0.96858637  0.01285925]
```

- `randomAxisV(N)`: This is the vectorized version of `randomAxis`. It simply returns an *Nx3* array of random vectors.

```
import magpylib as magpy

AXS = magpy.math.randomAxisV(3)
print(AXS)

#Output: [[ 0.39480364 -0.53600779 -0.74620757]
#         [ 0.02974442  0.10916333  0.9935787 ]
#         [-0.54639126  0.76659756 -0.33731997]]
```

- `angleAxisRotation(pos, angle, axis, anchor=[0,0,0])`: This function applies an angle-axis rotation. The position vector `pos` (*vec3*) is rotated by the angle `angle` (*float*) about an axis defined by the `axis` vector (*vec3*) which passes through the `anchor` position (*vec3*). The anchor argument is optional and is set to `anchor=[0,0,0]` if ommitted.

```
import magpylib as magpy

pos = [1,1,0]
angle = -90
axis = [0,0,1]
anchor = [1,0,0]

posNew = magpy.math.angleAxisRotation(pos,angle,axis,anchor)
print(posNew)

# Output = [2. 0. 0.]
```

- `angleAxisRotationV(POS, ANG, AXS, ANCH)`: This is the vectorized version of `angleAxisRotation`. Each entry of `POS` (*arrNx3*) is rotated according to the angles `ANG` (*arrN*), about the axis vectors `AXS` (*arrNx3*) which pass throught the anchors `ANCH` (*arrNx3*) where *N* refers to the length of the input vectors.

```
import magpylib as magpy
import numpy as np

POS = np.array([[1,0,0]]*5) # avoid this slow Python loop for performance␣
↪conputation
ANG = np.linspace(0,180,5)
AXS = np.array([[0,0,1]]*5) # avoid this slow Python loop for performance␣
↪conputation
ANCH = np.zeros((5,3))

POSnew = magpy.math.angleAxisRotationV(POS,ANG,AXS,ANCH)
print(np.around(POSnew,4))

# Output: [[ 1.      0.      0.    ]
#          [ 0.7071  0.7071  0.    ]
#          [ 0.      1.      0.    ]
#          [-0.7071  0.7071  0.    ]
```

(continues on next page)

(continued from previous page)

```
#           [-1.      0.      0.    ]]
```

1.2 Installation

Warning: magpylib works only with Python 3.6 or later !

Dependencies:

- numpy
- matplotlib

The latest versions will be installed automatically with magpylib.

1.2.1 Content

1.2.2 Install with pip

The quickest installation on any platform is through pip.

```
pip install magpylib
```

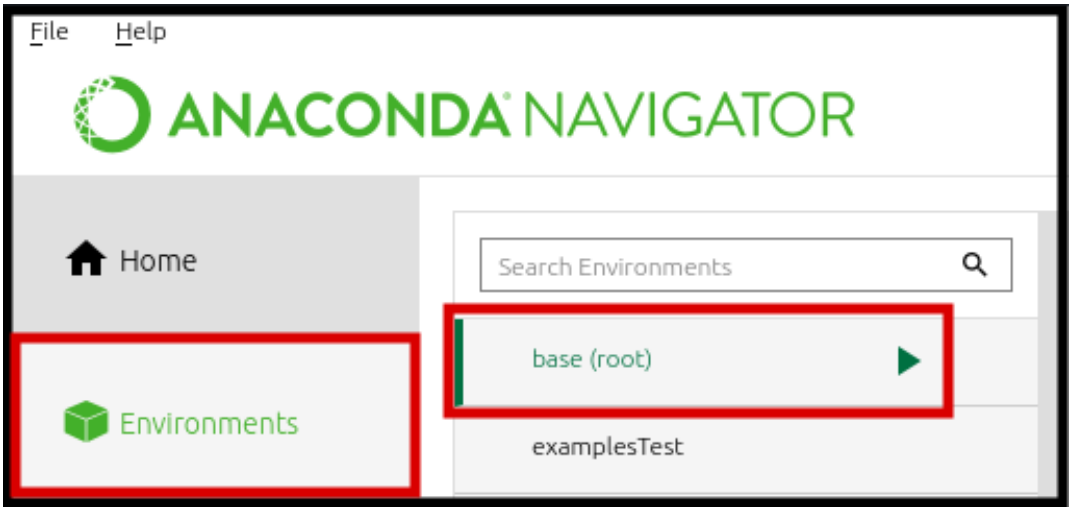
If you are unfamiliar with pip, please follow the detailed guides below:

1.2.3 Windows

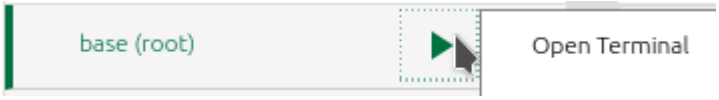
Anaconda 3 Install

If you have little experience with Python we recommand using Anaconda.

1. Download & install Anaconda3
2. Start Anaconda Navigator
3. On the interface, go to *Environments* and choose the environment you wish to install magpylib in. For this example, we will use the base environment:



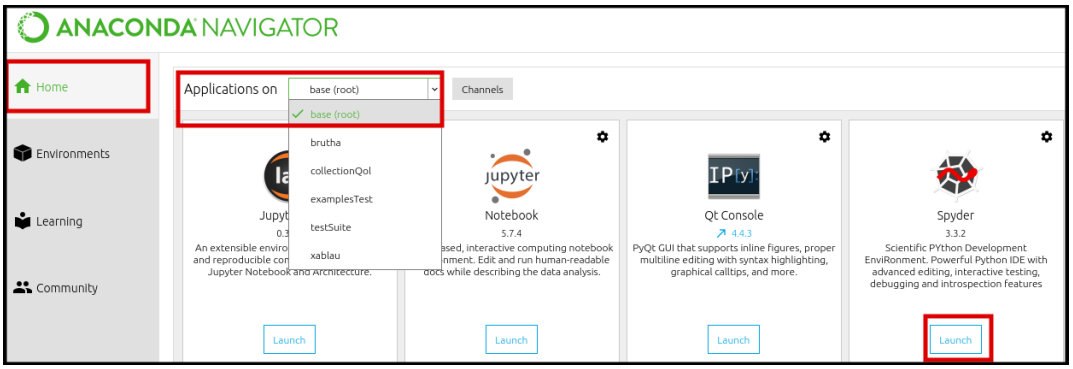
4. Click the arrow, and open the conda terminal



5. Input the following to install from conda-forge:

```
conda install -c conda-forge magpylib
```

6. Dont forget to select the proper environment in your IDE.



Clean Python 3 Install

If you want to have a custom environment without using conda, you may simply install the library with pip. A simple guide for installation and functionality of pip is found here

1.2.4 Linux

Recommended: use Anaconda environment. Simply download Anaconda3 and follow installation steps as under Windows.

Terminal Python 3 Install

1. Install Python3.

2. Open your Terminal and install with

```
pip install magpylib
```

1.2.5 Download Sites

Currently magpylib is hosted at:

- Conda Cloud
- Python Package Index
- GitHub repository

1.3 Example Codes

This section includes a few code examples that show how the library can be used and what i can be used for. A detailed technical library documentation can be found in the *Documentation v2.0.1b*.

1.3.1 Contents

1.3.2 Simplest Example

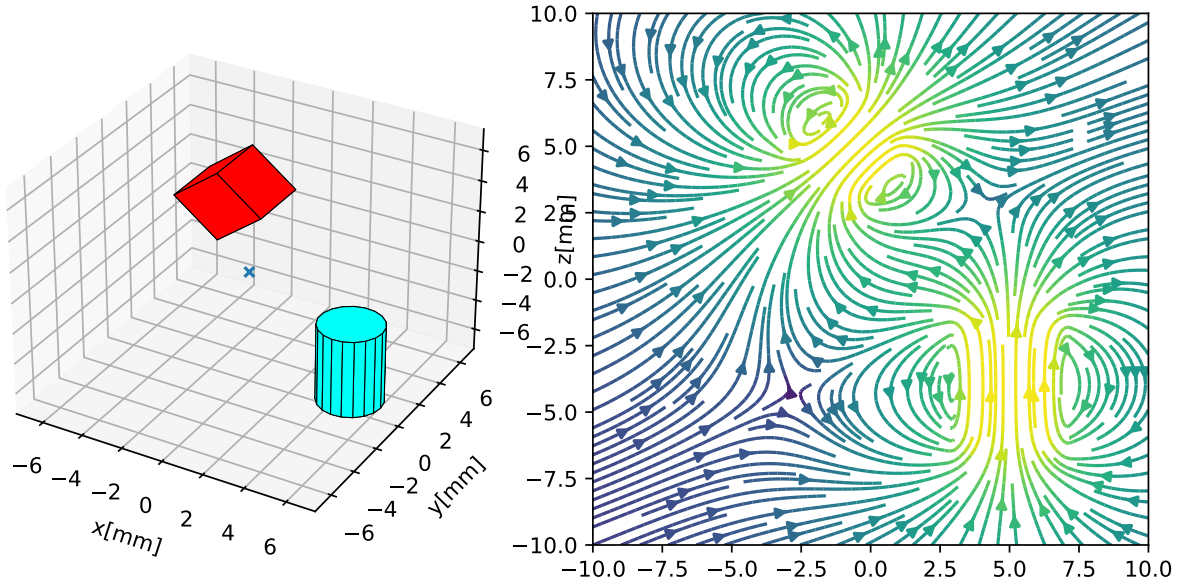
The simplest possible example - calculate the B-field of a cylinder with 3 lines of code.

```
from magpylib.source.magnet import Cylinder
s = Cylinder( mag = [0,0,350], dim = [4,5])
print(s.getB([4,4,4]))

# Output: [ 5.08641867  5.08641867 -0.60532983]
```

1.3.3 Basic Functionality: The Field of a Collection

In this example the basic functionality is outlined. Two magnet objects are created and geometrically manipulated. The system geometry is then displayed using the `displaySystem` function. Finally, the field is calculated on a grid and displayed in the xz-plane.

```
# imports
import numpy as np
import matplotlib.pyplot as plt
import magpylib as magpy

# create figure
fig = plt.figure(figsize=(8,4))
ax1 = fig.add_subplot(121, projection='3d')  # this is a 3D-plotting-axis
ax2 = fig.add_subplot(122)                   # this is a 2D-plotting-axis

# create magnets
s1 = magpy.source.magnet.Box(mag=[0,0,600],dim=[3,3,3],pos=[-4,0,3])
s2 = magpy.source.magnet.Cylinder(mag=[0,0,500], dim=[3,5])

# manipulate magnets
s1.rotate(45,[0,1,0],anchor=[0,0,0])
s2.move([5,0,-4])

# create collection
c = magpy.Collection(s1,s2)

# display system geometry on ax1
magpy.displaySystem(c,subplotAx=ax1,suppress=True)

# calculate B-field on a grid
xs = np.linspace(-10,10,29)
zs = np.linspace(-10,10,29)
Bs = np.array([[c.getB([x,0,z]) for x in xs] for z in zs])

# display field in xz-plane using matplotlib
X,Z = np.meshgrid(xs,zs)
U,V = Bs[:,:,0], Bs[:,:,2]
ax2.streamplot(X, Z, U, V, color=np.log(U**2+V**2),density=2)

plt.show()
```



01_SimpleCollection.py

1.3.4 The Source Objects and their Fields

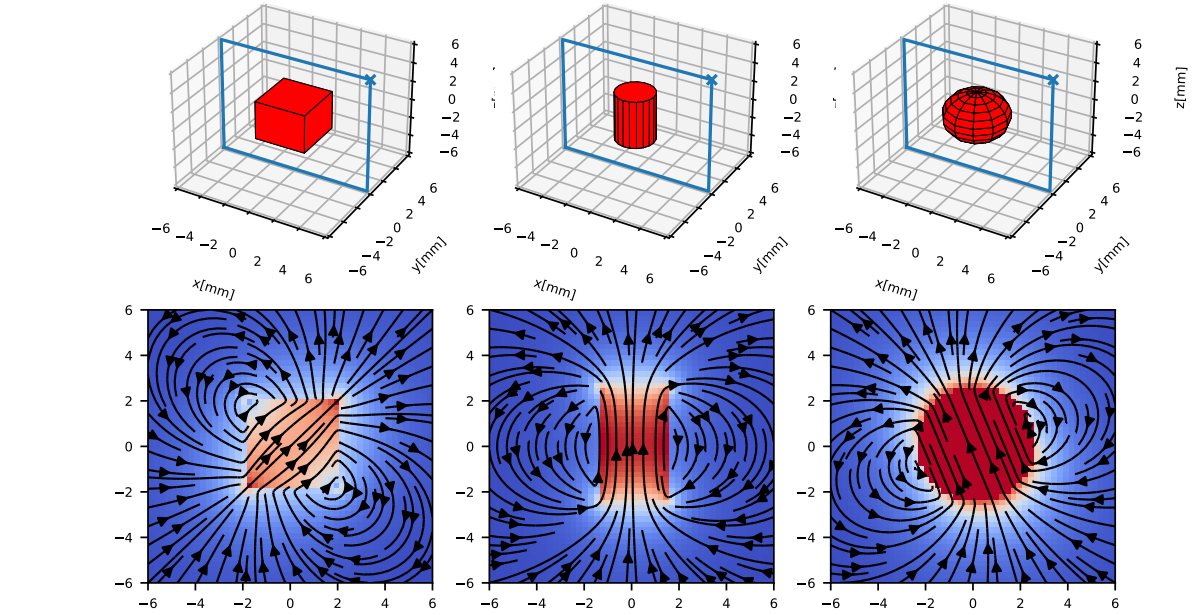
In this example we define all currently implemented source objects and display their fields. Notice that the respective magnetization vectors are chosen arbitrarily.

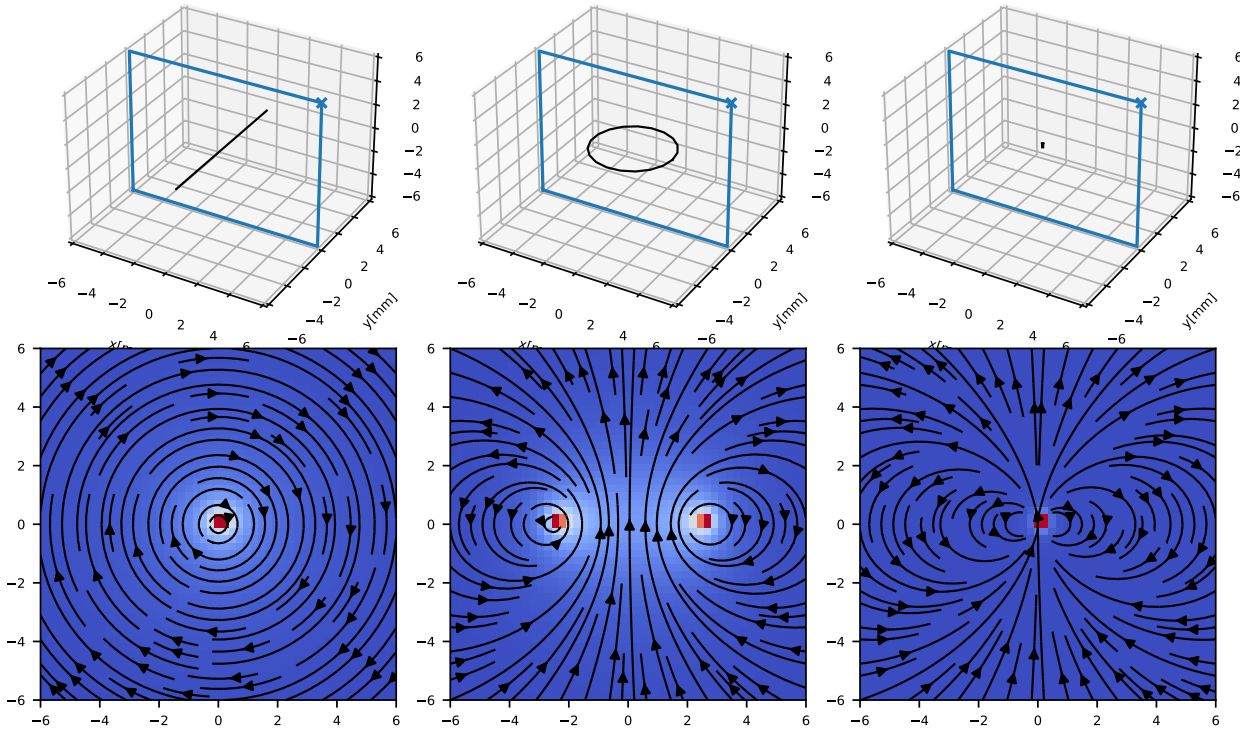
```
import magpylib as magpy
import numpy as np
from matplotlib import pyplot as plt

# set font size and define figures
plt.rcParams.update({'font.size': 6})

fig1 = plt.figure(figsize=(8, 5))
axsA = [fig1.add_subplot(2,3,i, projection='3d') for i in range(1,4)]
axsB = [fig1.add_subplot(2,3,i) for i in range(4,7)]

fig2 = plt.figure(figsize=(8, 5))
axsA += [fig2.add_subplot(2,3,i, projection='3d') for i in range(1,4)]
axsB += [fig2.add_subplot(2,3,i) for i in range(4,7)]

# position grid
ts = np.linspace(-6,6,50)
posis = np.array([(x,0,z) for z in ts for x in ts])
X,Y = np.meshgrid(ts,ts)

# create the source objects
s1 = magpy.source.magnet.Box(mag=[500,0,500], dim=[4,4,4])                #Box
s2 = magpy.source.magnet.Cylinder(mag=[0,0,500], dim=[3,5])               #Cylinder
s3 = magpy.source.magnet.Sphere(mag=[-200,0,500], dim=5)                  #Sphere
s4 = magpy.source.current.Line(curr=10, vertices=[(0,-5,0),(0,5,0)])      #Line
s5 = magpy.source.current.Circular(curr=10, dim=5)                        #Circular
s6 = magpy.source.moment.Dipole(moment=[0,0,100])                         #Dipole

for i,s in enumerate([s1,s2,s3,s4,s5,s6]):

    # display system on respective axes, use marker to zoom out
    magpy.displaySystem(s,subplotAx=axsA[i],markers=[(6,0,6)],suppress=True)
    axsA[i].plot([-6,6,6,-6,-6],[0,0,0,0,0],[-6,-6,6,6,-6])

    # plot field on respective axes
    B = np.array([s.getB(p) for p in posis]).reshape(50,50,3)
    axsB[i].pcolor(X,Y,np.linalg.norm(B,axis=2),cmap=plt.cm.get_cmap('coolwarm'))    #␣
↪amplitude
    axsB[i].streamplot(X, Y, B[:,:,0], B[:,:,2], color='k',linewidth=1)              #␣
↪field lines

plt.show()
```

01b_AllSources.py

1.3.5 Translation, Orientation and Rotation Basics

Translation of magnets can be realized in three ways, using the methods `move` and `setPosition`, or by directly setting the object `position` attribute.


x[mm]
y[mm]
z[mm]


y[mm]

```
import numpy as np
import magpylib as magpy
from magpylib.source.magnet import Box

# fixed magnet parameters
M = [0,0,1] #magnetization
D = [2,2,2] #dimension

# Translation of magnets can be realized in several ways
s1 = Box(mag=M, dim=D, pos = [-4,0, 4])

s2 = Box(mag=M, dim=D, pos = [-2,0, 4])
s2.move([0,0,-2])

s3 = Box(mag=M, dim=D, pos = [ 0,0, 4])
s3.move([0,0,-2])
s3.move([0,0,-2])

s4 = Box(mag=M, dim=D, pos = [ 2,0, 4])
s4.setPosition([2,0,-2])

s5 = Box(mag=M, dim=D, pos = [ 4,0, 4])
s5.position = np.array([4,0,0])

#collection
c = magpy.Collection(s1,s2,s3,s4,s5)

#display collection
magpy.displaySystem(c,figsize=(6,6))
```

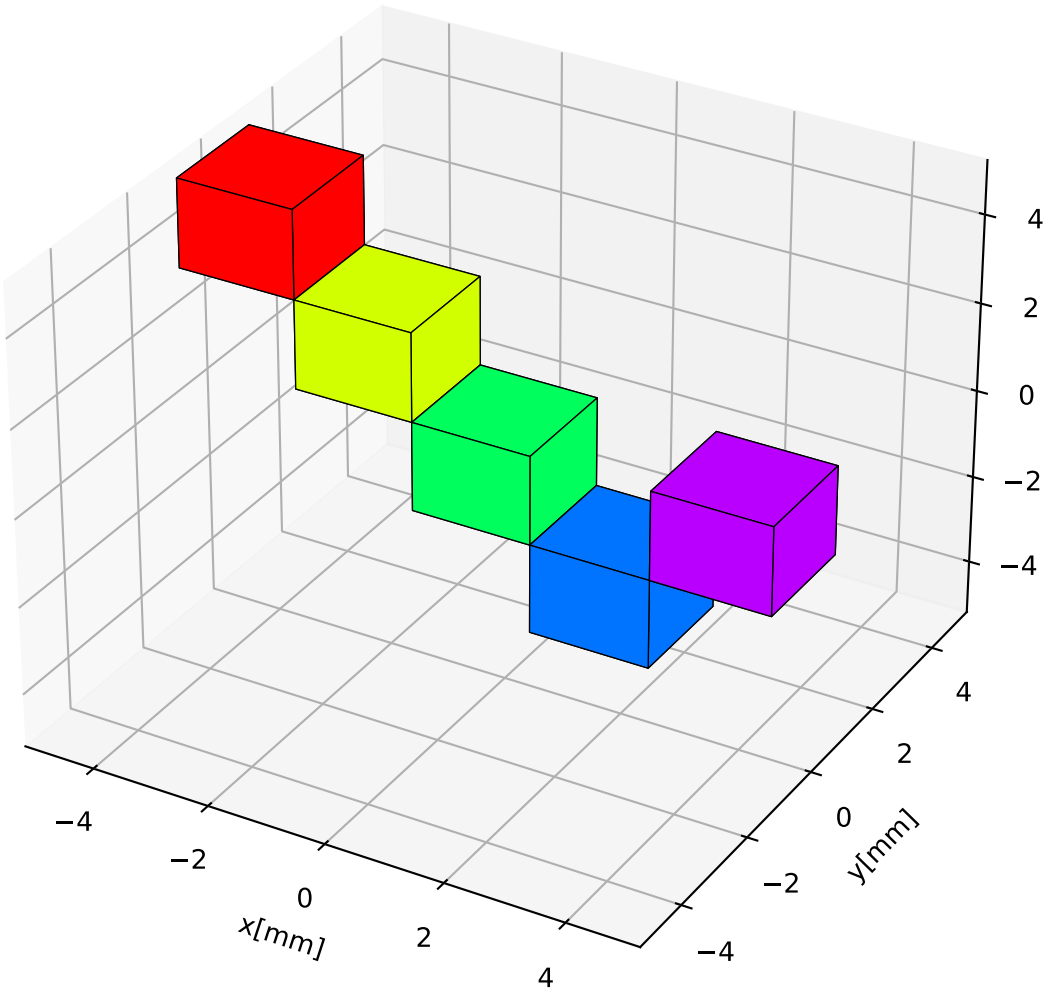
00a_Trans.py

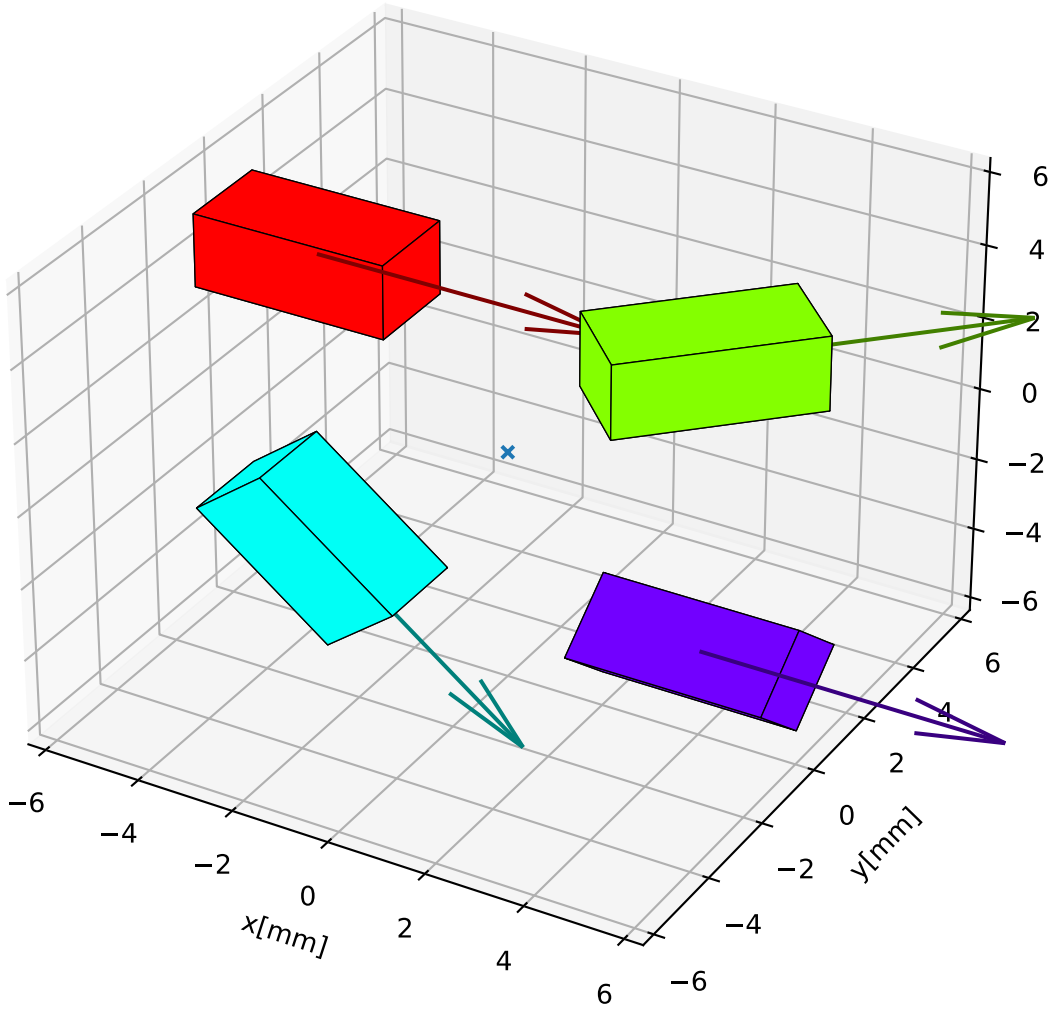
Initialize magnets with different orientations defined by classical Euler angle rotations about the three Cartesian axes. Notice that the magnetization direction is fixed with respect to the **init orientation** of the magnet and will rotate together with the magnet.

```
from magpylib.source.magnet import Box
import magpylib as magpy

# fixed magnet parameters
M = [1,0,0] #magnetization
D = [4,2,2] #dimension

# magnets with Euler angle orientations
s1 = Box(mag=M, dim=D, pos = [-4,0, 4])
s2 = Box(mag=M, dim=D, pos = [ 4,0, 4], angle=45, axis=[0,0,1])
s3 = Box(mag=M, dim=D, pos = [-4,0,-4], angle=45, axis=[0,1,0])
s4 = Box(mag=M, dim=D, pos = [ 4,0,-4], angle=45, axis=[1,0,0])

# collection
c = magpy.Collection(s1,s2,s3,s4)

# display collection
magpy.displaySystem(c,direc=True,figsize=(6,6))
```

00b_OrientRot1.py

The following example shows functionality beyond Euler angle rotation. This means rotation about an arbitrary axis


4
2
0
−2
−4
−4
−2
0
2
4
x[mm]
−4
−2
0
2
4
y[mm]


6
4
2
0
−2
−4
−6
6
4
2
0
−2
−4
−6
y[mm]
−6
−4
−2
0
2
4
6
x[mm]

of choice, here `(1,-1,1)`. The upper three boxes are initialized with different orientations. The lower three boxes are all initialized with **init orientation** and are then rotated (about themselves) to achieve the same result as above.

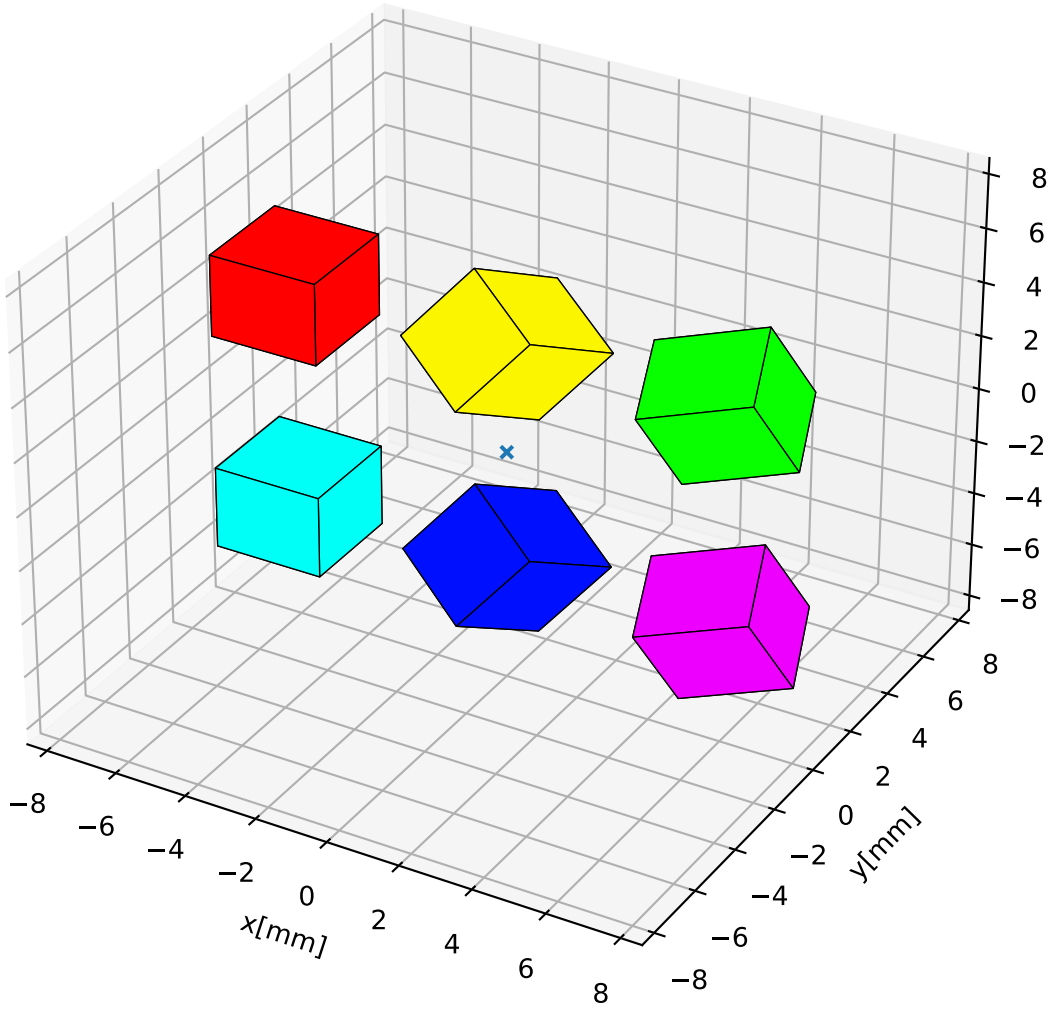
```
import magpylib as magpy
from magpylib.source.magnet import Box

# fixed magnet parameters
M = [0,0,1] #magnetization
D = [3,3,3] #dimension

# rotation axis
rax = [-1,1,-1]

# magnets with different orientations
s1 = Box(mag=M, dim=D, pos=[-6,0,4], angle=0, axis=rax)
s2 = Box(mag=M, dim=D, pos=[ 0,0,4], angle=45, axis=rax)
s3 = Box(mag=M, dim=D, pos=[ 6,0,4], angle=90, axis=rax)

# magnets that are rotated differently
s4 =  Box(mag=M, dim=D, pos=[-6,0,-4])
s5 =  Box(mag=M, dim=D, pos=[ 0,0,-4])
s5.rotate(45,rax)
s6 = Box(mag=M, dim=D, pos=[ 6,0,-4])
s6.rotate(90,rax)

# collect all sources
c = magpy.Collection(s1,s2,s3,s4,s5,s6)

# display collection
magpy.displaySystem(c,figsize=(6,6))
```

`00c_OrientRot2.py`

The following example shows rotations with designated anchor-axis combinations. Here we distinguish between pivot points (the closest point on the rotation axis to the magnet) and anchor points which are simply required to define an axis in 3D space (together with a direction).

```
import magpylib as magpy
from magpylib.source.magnet import Box
import matplotlib.pyplot as plt

# define figure
fig = plt.figure(figsize=(6,6))
ax = fig.add_subplot(1,1,1, projection='3d')

# fixed magnet parameters
M = [0,0,1] #magnetization
D = [2,4,1] #dimension

# define magnets rotated with different pivot and anchor points
piv1 = [-7,0,5]
s1 = Box(mag=M, dim=D, pos = [-7,-3,5])

piv2 = [0,0,5]
s2 = Box(mag=M, dim=D, pos = [0,-3,5])
s2.rotate(-30,[0,0,1],anchor=piv2)

piv3 = [7,0,5]
```

(continues on next page)


x[mm]
y[mm]
−8
−6
−4
−2
0
2
4
6
8

(continued from previous page)

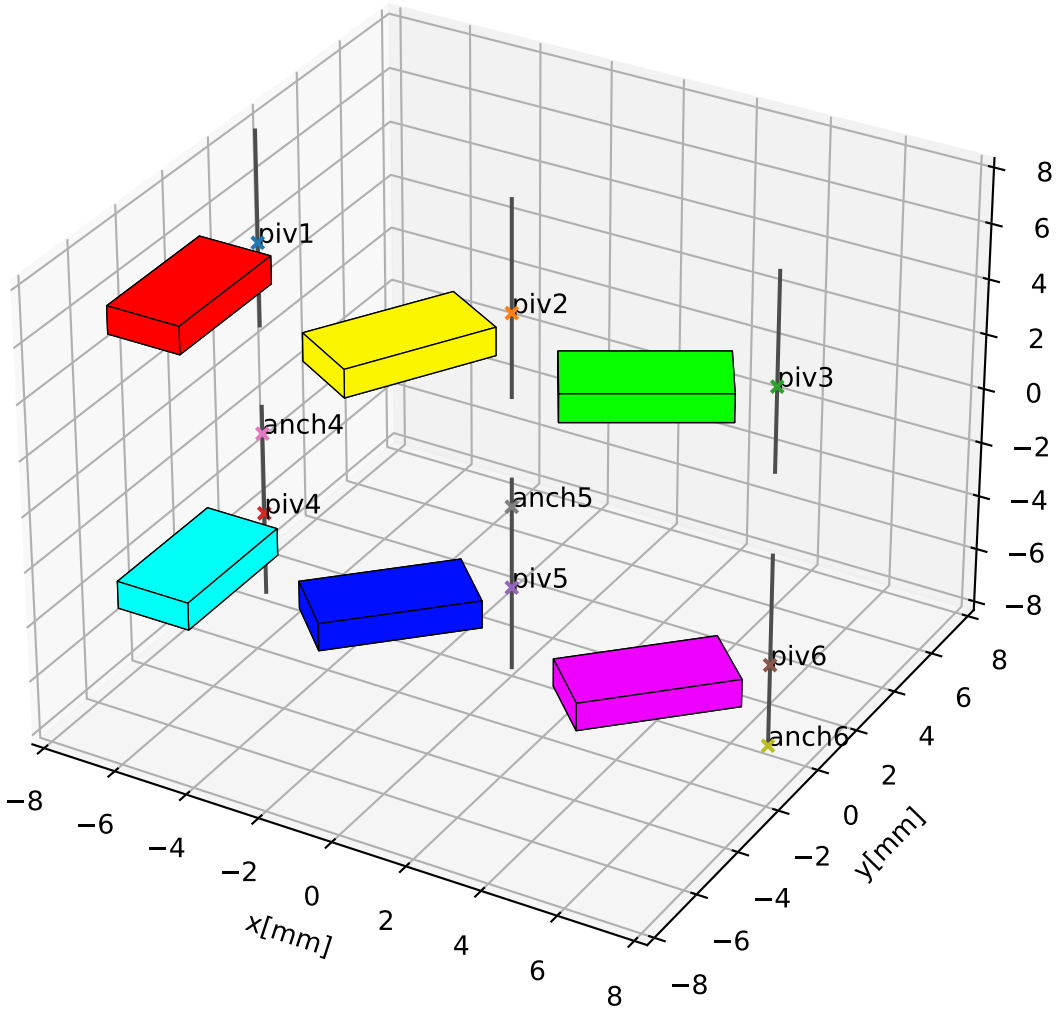
```
s3 = Box(mag=M, dim=D, pos = [7,-3,5])
s3.rotate(-60,[0,0,1],anchor=piv3)

piv4 = [-7,0,-5]
anch4 = [-7,0,-2]
s4 = Box(mag=M, dim=D, pos = [-7,-3,-5])

piv5 = [0,0,-5]
anch5 = [0,0,-2]
s5 = Box(mag=M, dim=D, pos = [0,-3,-5])
s5.rotate(-45,[0,0,1],anchor=anch5)

piv6 = [7,0,-5]
anch6 = [7,0,-8]
s6 = Box(mag=M, dim=D, pos = [7,-3,-5])
s6.rotate(-45,[0,0,1],anchor=anch6)

# collect all sources
c = magpy.Collection(s1,s2,s3,s4,s5,s6)

# draw rotation axes
for x in [-7,0,7]:
    for z in [-5,5]:
        ax.plot([x,x],[0,0],[z-3,z+4],color='.3')

# define markers
Ms = [piv1+['piv1'], piv2+['piv2'], piv3+['piv3'], piv4+['piv4'],
      piv5+['piv5'], piv6+['piv6'], anch4+['anch4'],anch5+['anch5'],anch6+['anch6']]

# display system
magpy.displaySystem(c,subplotAx=ax,markers=Ms,suppress=True)

plt.show()
```

00d_OrientRot3.py

Collections can be manipulated using the previous logic as well. Notice how objects can be grouped into collections and sub-collections for common manipulation. For rotations keep in mind that if an anchor is not provided, all objects will rotate relative to their own center.

```
import magpylib as magpy
from magpylib.source.current import Circular
from numpy import linspace

# windings of three parts of a coil
coil1a = [Circular(curr=1,dim=3,pos=[0,0,z]) for z in linspace( -3,-1,10)]
coil1b = [Circular(curr=1,dim=3,pos=[0,0,z]) for z in linspace( -1, 1,10)]
coil1c = [Circular(curr=1,dim=3,pos=[0,0,z]) for z in linspace(  1, 3,10)]

# create collection and manipulate step by step
c1 = magpy.Collection(coil1a)
c1.move([-1,-1,0])
c1.addSources(coil1b)
c1.move([-1,-1,0])
c1.addSources(coil1c)
c1.move([-1,-1,0])
```

(continues on next page)


piv1
piv2
piv3
anch4
piv4
anch5
piv5
piv6
anch6
x[mm]
y[mm]
−8
−6
−4
−2
0
2
4
6
8

(continued from previous page)

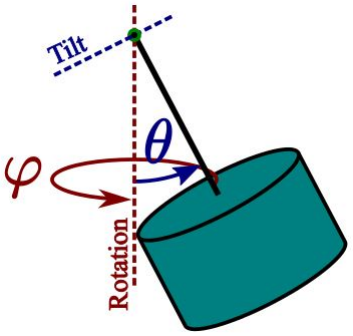
```
# windings of three parts of another coil
coil2a = [Circular(curr=1,dim=3,pos=[3,3,z]) for z in linspace(-3,-1,15)]
coil2b = [Circular(curr=1,dim=3,pos=[3,3,z]) for z in linspace( -1,1,15)]
coil2c = [Circular(curr=1,dim=3,pos=[3,3,z]) for z in linspace( 1,3,15)]

# create individual sub-collections
c2a = magpy.Collection(coil2a)
c2b = magpy.Collection(coil2b)
c2c = magpy.Collection(coil2c)

# combine sub-collections to one big collection
c2 = magpy.Collection(c2a,c2b,c2c)

# still manipulate each individual sub-collection
c2a.rotate(-15,[1,-1,0],anchor=[0,0,0])
c2c.rotate(15,[1,-1,0],anchor=[0,0,0])

# combine all collections and display system
c3 = magpy.Collection(c1,c2)
magpy.displaySystem(c3,figsize=(6,6))
```

`00e_ColTransRot.py`

1.3.6 Magnet Motion: Simulating a Magnetic Joystick

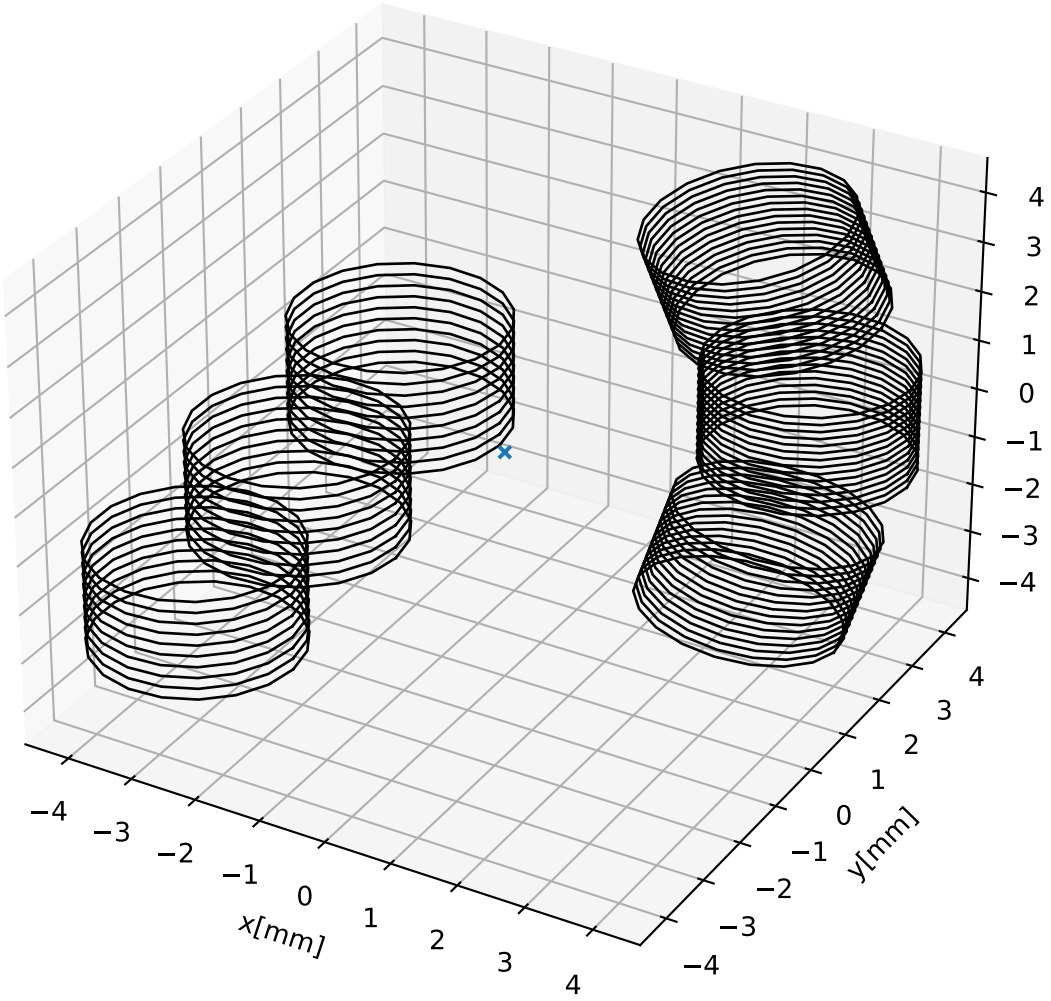
In this example a joystick is simulated. A magnetic joystick is realized by a rod that can tilt freely (two degrees of freedom) about a center of tilt. The upper part of the rod is the joystick handle. At the bottom of the rod a cylindrical magnet (`dim=(D,H)`) with axial magnetization `` `mag=[0,0,M0] `` is fixed. The magnet lies at a distance `d` below the center of tilt. The system is constructed such that, when the joystick is in the center position a sensor lies at distance `gap` below the magnet and in the origin of a Cartesian coordinate system. The magnet thus moves with the joystick above the fixed sensor.

In the following program the magnetic field is calculated for all degrees of freedom. Different tilt angles are set by rotation about the center of tilt by the angle `th` (different colors). Then the tilt direction is varied from 0 to 360 degrees by simulating the magnet motion as rotation about the z-axis, see also the following sketch.




```
from magpylib.source.magnet import Cylinder
import matplotlib.pyplot as plt
import numpy as np

# system parameters
D,H = 5,4        #magnet dimension
```

(continues on next page)


x[mm]
y[mm]
−4
−3
−2
−1
0
1
2
3
4

(continued from previous page)

```
M0 = 1200        #magnet magnetization amplitude
gap = 3          #airgap
d = 5            #distance magnet to center of tilt
thMAX = 15       #maximal joystick tilt angle

# define figure
fig = plt.figure(figsize=(7,6))
ax = plt.axes(projection='3d')
cm = plt.get_cmap("jet") #colormap

# set tilt angle
for th in np.linspace(1,thMAX,30):

    # store fields here
    Bs = np.zeros([181,3])

    # create magnet for joystick in center position
    s = Cylinder(dim=[D,H],mag=[0,0,M0],pos=[0,0,H/2+gap])

    # set joystick tilt th
    s.rotate(th,[0,1,0],anchor=[0,0,gap+H+d])

    # rotate joystick for fixed tilt
    for i in range(181):

        # calculate field (sensor at [0,0,0]) and store in Bs
        Bs[i] = s.getB([0,0,0])

        # rotate magnet to next position
        s.rotate(2,[0,0,1],anchor=[0,0,0])

    # plot fields
    ax.plot(Bs[:,0],Bs[:,1],Bs[:,2],color=cm(th/15))

# annotate
ax.set(
       xlabel = 'Bx [mT]',
       ylabel = 'By [mT]',
       zlabel = 'Bz [mT]')

# display
plt.show()
```

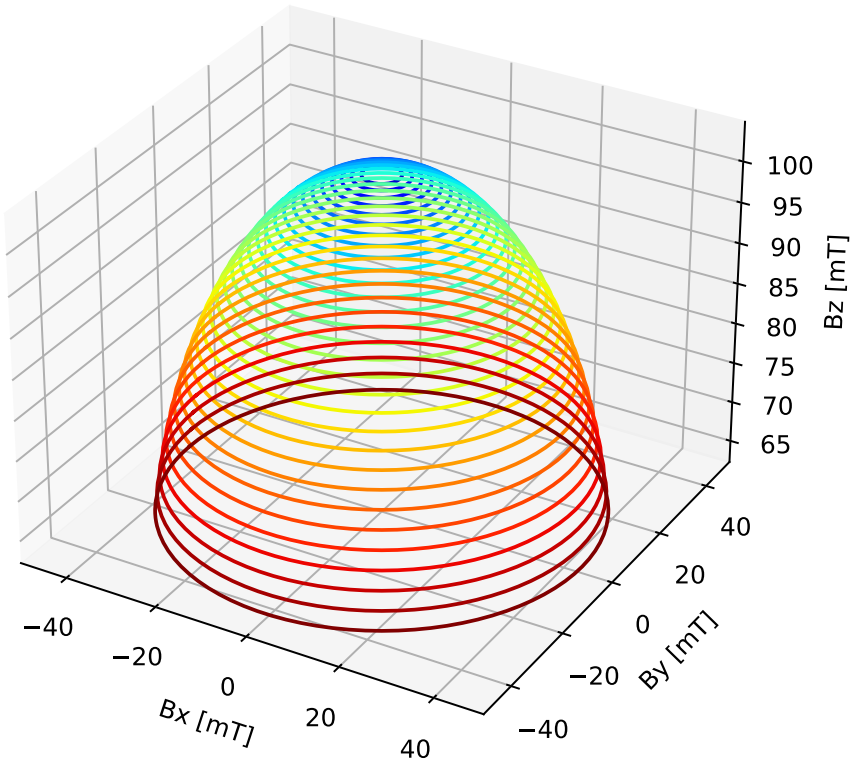
02_MagnetMotion.py

1.3.7 Complex Magnet Shapes: Hollow Cylinder

The superposition principle allows us to calculate complex magnet shapes by *addition* and *subtraction* operations. An example application for this is the field of an axially magnetized hollow cylinder. The hollow part is cut out from the outer cylinder by placing a second, smaller cylinder inside with opposite magnetization. Unfortunately the `displaySystem` method cannot properly display such objects intersecting with each other.

```
import numpy as np
import matplotlib.pyplot as plt
from magpylib.source.magnet import Cylinder
```

(continues on next page)


100
95
90
85
80
75
70
65
Bz [mT]
−40
−20
0
20
40
Bx [mT]
40
20
0
−20
−40
By [mT]

(continued from previous page)

```
import magpylib as magpy

# create collection of two magnets
s1 = Cylinder(mag=[0,0,1000], dim=[5,5])
s2 = Cylinder(mag=[0,0,-1000], dim=[2,6])
c = magpy.Collection(s1,s2)

# create positions
xs = np.linspace(-8,8,100)
zs = np.linspace(-6,6,100)
posis = [[x,0,z] for z in zs for x in xs]

# calculate field and amplitude
B = [c.getB(pos) for pos in posis]
Bs = np.array(B).reshape([100,100,3]) #reshape
Bamp = np.linalg.norm(Bs,axis=2)

# define figure with a 2d and a 3d axis
fig = plt.figure(figsize=(8,4))
ax1 = fig.add_subplot(121,projection='3d')
ax2 = fig.add_subplot(122)

# add displaySystem on ax1
magpy.displaySystem(c,subplotAx=ax1,suppress=True)
ax1.view_init(elev=75)

# amplitude plot on ax2
X,Z = np.meshgrid(xs,zs)
ax2.pcolor(xs,zs,Bamp,cmap='jet',vmin=-200)

# plot field lines on ax2
U,V = Bs[:,:,0], Bs[:,:,2]
ax2.streamplot(X,Z,U,V,color='k',density=2)

#display
plt.show()
```

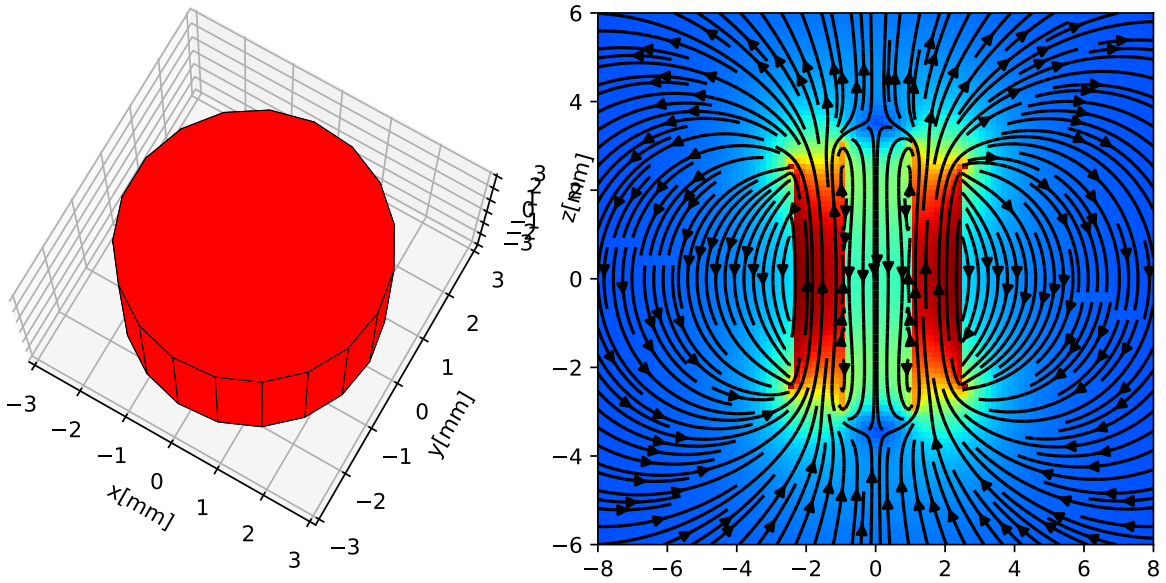
`04_ComplexShape.py`

1.3.8 Vectorized Code Example

In this example a magnet is tilted above a sensor just like in a 1D-joystick system. The magnetic field is computed using vectorized code, taking care to create the `getBv` input using numpy native methods only.

```
import magpylib as magpy
import numpy as np
import matplotlib.pyplot as plt

import time

# vector size: we calculate the field N times with different inputs
N = 100000

# Constant vectors
mag  = np.array([0,0,1000])     # magnet magnetization
```

(continues on next page)

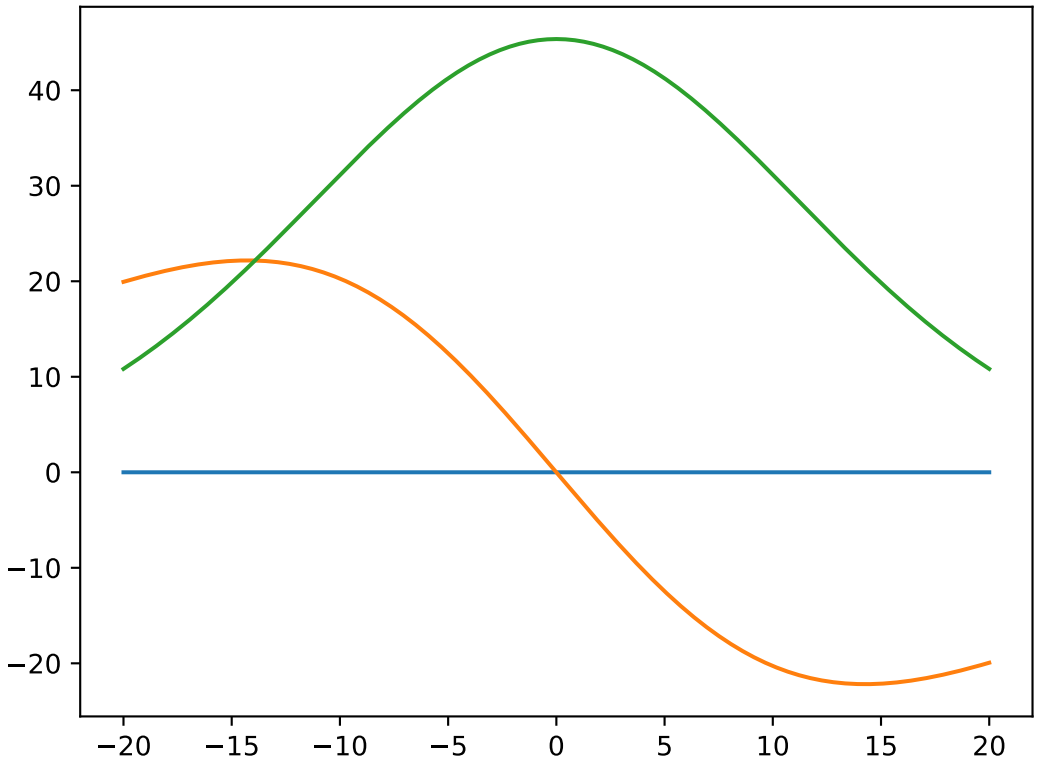


(continued from previous page)

```
dim  = np.array([2,2,2])        # magnet dimension
poso = np.array([0,0,0])        # position of observer
posm = np.array([0,0,3])        # initial magnet position
anch = np.array([0,0,8])        # rotation anchor
axis = np.array([1,0,0])        # rotation axis

# different angles for each evaluation
angs = np.linspace(-20,20,N)

# Vectorizing input using numpy native instead of python loops
MAG = np.tile(mag,(N,1))
DIM = np.tile(dim,(N,1))
POSo = np.tile(poso,(N,1))
POSm = np.tile(posm,(N,1))   # inital magnet positions before rotations are applied
ANCH = np.tile(anch,(N,1))   # always same axis
AXIS = np.tile(axis,(N,1))   # always same anchor

# N-times evalulation of the field with different inputs
Bv = magpy.vector.getBv_magnet('box',MAG,DIM,POSo,POSm,[angs],[AXIS],[ANCH])

# plot field
plt.plot(angs,Bv[:,0])
plt.plot(angs,Bv[:,1])
plt.plot(angs,Bv[:,2])

plt.show()
```

05_VectorJoystick1d.py

1.4 MATLAB Integration


40
30
20
10
0
−10
−20
−20
−15
−10
−5
0
5
10
15
20

Note: MATLAB does not support Tkinter, which disables matplotlib. This means that `displaySystem()` will not generate a display and might interrupt the program.

1.4.1 Setting Python Interpreter

As of version R2015b, MATLAB allows you to call libraries from other programming languages, including Python, which enables users to run magpylib from the MATLAB interface. The following guide intends to provide a digest of the Official MATLAB documentation with a focus on utilizing this interface with magpylib.

Running `>>> pyversion` following line in the MATLAB console tells you which Python environment (user space interpreter) is connected to your MATLAB interface.

If magpylib is already installed in this environment you can directly call it, as shown in the *Example* below. If not please follow the *Installation* instructions and install magpylib.

If you choose to install magpylib in a different environment than the one that is currently connected to your MATLAB interpreter, use the following command in the MATLAB console to connect the new environment instead (choose correct path pointing at your Python interpreter).

```
>>> pyversion C:\Users\...\AppData\Local\Continuum\anaconda3\envs\magpy\python.exe
```

1.4.2 Example

The following MATLAB 2019 script showcases most functionalities.

```
%%%%%%%%%%%%%%%%%%%% magpytest.m %%%%%%%%%%%%%%%
%% Showcase Python + MATLAB Interoperability.
%% Define and calculate the field of a
%% Cuboid magnet inside a Collection.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Import the library
py.importlib.import_module("magpylib")

%% Define Python types for input
vec3 = py.list({1,2,3})
scalar = py.int(90)

%% Define input
mag = vec3
dim = vec3
angle = scalar
sensorPos = vec3

%% Execute Python
% 2 positional and 1 keyword argument in Box
box = py.magpylib.source.magnet.Box(mag,dim,pyargs('angle',angle))
col = py.magpylib.Collection(box)
pythonResult = col.getB(sensorPos)

%% Convert Python Result to MATLAB data format
matlabResult = double(pythonResult)
```

Note: With old versions of Matlab the *double(pythonResult)* type conversion might give an error message.

1.5 Credits & Contribution

1.5.1 Maintainers & Contact

Michael Ortner - Concept, Physics and Overseeing

- michael.ortner@silicon-austria.com
- Silicon Austria Labs, Sensors division, 9500 Villach, Austria

Lucas Gabriel Coliado Bandeira - software engineering

- lucascoliado@hotmail.com

1.5.2 Contributions

We want to thank a lot of ppl who have helped to realize and advance this project.

We welcome any feedback (Bug reports, feature requests, comments, really anything) via email magpylib@gmail.com or through gitHub channels.

1.6 [WIP] Physics & Computation

1.6.1 The analytical solutions

Details about how they are set up Formulas and expressions used in magpylib, references to literature

Hard and soft magnetic materials

We cannot do soft, theres a lot of reasons we we dont need to

Demagnetization

Solution accuracy, analytical modeling of demagnetization and interaction

multiple sources, no interaction

1.6.2 Limits and Convergence

Convergence of diametral Cylinder No. iterations

1.6.3 Computation

SIMD

vectorized code

performance tests

parallelization

1.7 magpylib package

This is the top level of the package. From here you can call subpackages *source* and *math*, the classes *Collection* and *Sensor* as well as the functions *getBv* and *displaySystem*.

class magpylib.**Collection**(**sources*, *dupWarning=True*)

Bases: magpylib._lib.classes.base.FieldSampler

Create a collection of *magpylib.source* objects for common manipulation.

Parameters **sources** (*source objects*) – python magic variable passes source objects to the collection at initialization.

sources

List of all sources that have been added to the collection.

Type list of source objects

Example

```
>>> from magpylib import source, Collection
>>> pm1 = source.magnet.Box(mag=[0,0,1000],dim=[1,1,1])
>>> pm2 = source.magnet.Cylinder(mag=[0,0,1000],dim=[1,1])
>>> pm3 = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> col = Collection(pm1,pm2,pm3)
>>> B = col.getB([1,0,1])
>>> print(B)
[9.93360625e+01 1.76697482e-14 3.12727683e+01]
```

addSources(**sources*, *dupWarning=True*)

This method adds the argument source objects to the collection. May also include other collections.

Parameters

- **source** (*source object*) – adds the source object *source* to the collection.
- **dupWarning** (*bool*) – Warn and prevent if there is an attempt to add a duplicate source into the collection. Set to false to disable check and increase performance.

Returns

Return type None

Example

```
>>> from magpylib import source, Collection
>>> pm1 = source.magnet.Box(mag=[0,0,1000],dim=[1,1,1])
>>> pm2 = source.magnet.Cylinder(mag=[0,0,1000],dim=[1,1])
>>> pm3 = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> col = Collection(pm1)
>>> print(col.getB([1,0,1]))
  [4.29223532e+01 1.76697482e-14 1.37461635e+01]
>>> col.addSource(pm2)
>>> print(col.getB([1,0,1]))
  [7.72389756e+01 1.76697482e-14 2.39070726e+01]
>>> col.addSource(pm3)
>>> print(
  [9.93360625e+01 1.76697482e-14 3.12727683e+01]
```

getB(*pos*)

This method returns the magnetic field vector generated by the whole collection at the argument position *pos* in units of [mT]

Parameters **pos** (*vec3 [mm]*) – Position where magnetic field should be determined.

Returns **magnetic field vector** – Magnetic field at the argument position *pos* generated by the collection in units of [mT].

Return type arr3 [mT]

move(*displacement*)

This method moves each source in the collection by the argument vector *displacement*. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters **displacement** (*vec3 - [mm]*) – Displacement vector

Returns

Return type None

Example

```
>>> from magpylib import source, Collection
>>> pm1 = source.magnet.Box(mag=[0,0,1000],dim=[1,1,1])
>>> pm2 = source.magnet.Cylinder(mag=[0,0,1000],dim=[1,1])
>>> print(pm1.position,pm2.position)
  [0. 0. 0.] [0. 0. 0.]
>>> col = Collection(pm1,pm2)
>>> col.move([1,1,1])
>>> print(pm1.position,pm2.position)
  [1. 1. 1.] [1. 1. 1.]
```

removeSource(*source_ref=-1*)

Remove a source from the sources list.

Parameters **source_ref** (*source object or int*) – [Optional] Remove the inputted source from the list [Optional] If given an int, remove a source at the given index position. Default: Last position.

Returns

Return type Popped source object.

Raises

- `ValueError` – Will be thrown if you attempt to remove a source that is not in the Collection.
- `AssertionError` – Will be thrown if inputted index kwarg type is not type int

Example

```
>>> from magpylib import Collection, source
>>> s = source.magnet.Sphere(mag=[1,2,3],dim=1,pos=[3,3,3])
>>> s2 = source.magnet.Sphere(mag=[1,2,3],dim=2,pos=[-3,-3,-3])
>>> m = source.moment.Dipole(moment=[1,2,3],pos=(0,0,0))
>>> c = Collection(s,s2,m)
>>> print(c.sources)
[<magpylib._lib.classes.magnets.Sphere object at 0xa31eafcc>,
<magpylib._lib.classes.magnets.Sphere object at 0xa31ea1cc>,
<magpylib._lib.classes.moments.Dipole object at 0xa31ea06c>]
>>> c.removeSource(s)
>>> print(c.sources)
[<magpylib._lib.classes.magnets.Sphere object at 0xa31ea1cc>,
<magpylib._lib.classes.moments.Dipole object at 0xa31ea06c>]
>>> c.removeSource(s2)
>>> print(c.sources)
[<magpylib._lib.classes.moments.Dipole object at 0xa31ea06c>]
>>> c.removeSource()
>>> print(c.sources)
[]
```

rotate(*angle*, *axis*, *anchor='self.position'*)

This method rotates each source in the collection about *axis* by *angle*. The axis passes through the center of rotation anchor. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **angle** (*scalar [deg]*) – Angle of rotation in units of [deg]
- **axis** (*vec3*) – Axis of rotation
- **anchor** (*vec3*) – The Center of rotation which defines the position of the axis of rotation. If not specified all sources will rotate about their respective center.

Returns

Return type None

Example

```
>>> from magpylib import source, Collection
>>> pm1 = source.magnet.Box(mag=[0,0,1000],dim=[1,1,1])
>>> pm2 = source.magnet.Cylinder(mag=[0,0,1000],dim=[1,1])
>>> print(pm1.position, pm1.angle, pm1.axis)
  [0. 0. 0.] 0.0 [0. 0. 1.]
>>> print(pm2.position, pm2.angle, pm2.axis)
  [0. 0. 0.] 0.0 [0. 0. 1.]
>>> col = Collection(pm1,pm2)
>>> col.rotate(90, [0,1,0], anchor=[1,0,0])
>>> print(pm1.position, pm1.angle, pm1.axis)
```

(continues on next page)

(continued from previous page)

```
  [1. 0. 1.] 90.0 [0. 1. 0.]
>>> print(pm2.position, pm2.angle, pm2.axis)
  [1. 0. 1.] 90.0 [0. 1. 0.]
```

class magpylib.**Sensor**(*pos=[0, 0, 0], angle=0, axis=[0, 0, 1]*)

Bases: magpylib._lib.classes.base.RCS

Create a rotation-enabled sensor to extract B-fields from individual Sources and Source Collections. It may be displayed with *Collection*'s displaySystem() using the sensors kwarg.

Parameters

- **position** (*vec3*) – Cartesian position of where the sensor is.
- **angle** (*scalar*) – Angle of rotation
- **axis** (*vec3*) – Rotation axis information (x,y,z)

Example

```
>>> from magpylib import source, Sensor
>>> sensor = Sensor([0,0,0],90,(0,0,1)) # This sensor is rotated in respect to␣
↪space
>>> cyl = source.magnet.Cylinder([1,2,300],[1,2])
>>> absoluteReading = cyl.getB([0,0,0])
>>> print(absoluteReading)
    [  0.552   1.105  268.328 ]
>>> relativeReading = sensor.getB(cyl)
>>> print(relativeReading)
    [  1.105  -0.552  268.328 ]
```

getB(**sources*, *dupWarning=True*)

Extract the magnetic field based on the Sensor orientation

Parameters **dupWarning** (*Check if there are any duplicate sources, optional.*) – This will prevent duplicates and throw a warning, by default True.

Returns B-Field as perceived by the sensor

Return type [vec3]

Example

```
>>> from magpylib import source, Sensor
>>> sensor = Sensor([0,0,0],90,(0,0,1)) # This sensor is rotated in respect␣
↪to space
>>> cyl = source.magnet.Cylinder([1,2,300],[1,2])
>>> absoluteReading = cyl.getB([0,0,0])
>>> print(absoluteReading)
    [  0.552   1.105  268.328 ]
>>> relativeReading = sensor.getB(cyl)
>>> print(relativeReading)
    [  1.105  -0.552  268.328 ]
```

move(*displacement*)

This method moves the source by the argument vector *displacement*. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters **displacement** (*vec3 [mm]*) – Set displacement vector

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1,pos=[1,2,3])
>>> print(pm.position)
    [1. 2. 3.]
>>> pm.move([3,2,1])
>>> print(pm.position)
    [4. 4. 4.]
```

rotate (*angle*, *axis*, *anchor='self.position'*)

This method rotates the source about *axis* by *angle*. The axis passes through the center of rotation anchor. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **angle** (*scalar [deg]*) – Set angle of rotation in units of [deg]
- **axis** (*vec3 []*) – Set axis of rotation
- **anchor** (*vec3 [mm]*) – Specify the Center of rotation which defines the position of the axis of rotation. If not specified the source will rotate about its own center.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000], dim=1)
>>> print(pm.position, pm.angle, pm.axis)
  [0. 0. 0.] 0.0 [0. 0. 1.]
>>> pm.rotate(90, [0,1,0], anchor=[1,0,0])
>>> print(pm.position, pm.angle, pm.axis)
  [1., 0., 1.] 90.0 [0., 1., 0.]
```

setOrientation (*angle*, *axis*)

This method sets a new source orientation given by *angle* and *axis*. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **angle** (*scalar [deg]*) – Set new angle of source orientation.
- **axis** (*vec3 []*) – Set new axis of source orientation.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> print([pm.angle,pm.axis])
    [0.0, array([0., 0., 1.])]
>>> pm.setOrientation(45,[0,1,0])
>>> print([pm.angle,pm.axis])
    [45.0, array([0., 1., 0.])]
```

setPosition(*newPos*)

This method moves the source to the position given by the argument vector *newPos*. Vector input format can be either list, tuple or array of any data type (float, int)

Parameters **newPos** (*vec3 [mm]*) – Set new position of the source.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> print(pm.position)
    [0. 0. 0.]
>>> pm.setPosition([5,5,5])
>>> print(pm.position)
    [5. 5. 5.]
```

magpylib.**displaySystem**(*sources, markers=[[0.0, 0.0, 0.0]], subplotAx=None, sensors=['sensor type', 'sensor type'], suppress=False, direc=False, figsize=(8, 8)*)

Shows the collection system in an interactive pyplot and returns a matplotlib figure identifier.

Warning: As a result of an inherent problem in matplotlib the Poly3DCollections z-ordering fails when bounding boxes intersect.

Parameters **markers** (*list[scalar,scalar,scalar,[label]]*) – List of position vectors to add visual markers to the display, optional label. Default: [[0,0,0]]

Example

```
>>> from magpylib import Collection, source
>>> c=source.current.Circular(3,7)
>>> x = Collection(c)
>>> marker0 = [0,0,0,"Neutral Position"]
>>> marker1 = [10,10,10]
>>> x.displaySystem(markers=[ marker0,
...                           marker1])
```

Parameters **sensors** (*list[sensor]*) – List of *Sensor* objects to add the display. Default: None

Example

```
>>> from magpylib import Collection, source
>>> c=source.current.Circular(3,7)
>>> x = Collection(c)
>>> sensor0 = Sensor()
>>> sensor1 = Sensor(pos=[1,2,3], angle=180)
>>> x.displaySystem(sensors=[ sensor0,
...                           sensor1])
```

Parameters **suppress** (*bool*) – If True, only return Figure information, do not show. Interactive mode must be off. Default: False.

Example

```
>>> ## Suppress matplotlib.pyplot.show()
>>> ## and returning figure from showing up
>>> from matplotlib import pyplot
>>> pyplot.ioff()
>>> figureData = Collection.displayFigure(suppress=True)
```

Parameters **direc** (*bool*) – Set to True to show current directions and magnetization vectors. Default: False

Returns graphics object is displayed through plt.show()

Return type matplotlib Figure object

Example

```
>>> from magpylib import source, Collection
>>> pm1 = source.magnet.Box(mag=[0,0,1000],dim=[1,1,1],pos=[-1,-1,-1],angle=45,
↪axis=[0,0,1])
>>> pm2 = source.magnet.Cylinder(mag=[0,0,1000],dim=[2,2],pos=[0,-1,1],angle=45,
↪axis=[1,0,0])
>>> pm3 = source.magnet.Sphere(mag=[0,0,1000],dim=3,pos=[-2,1,2],angle=45,axis=[1,
↪0,0])
>>> C1 = source.current.Circular(curr=100,dim=6)
>>> col = Collection(pm1,pm2,pm3,C1)
>>> col.displaySystem()
```

Parameters **subplotAx** (*matplotlib subplot axe instance*) – Use an existing matplotlib subplot instance to draw the 3D system plot into. Default: None

Example

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> from magpylib.source.magnet import Box
>>> from magpylib import Collection
>>> #create collection of one magnet
>>> s1 = Box(mag=[ 500,0, 500], dim=[3,3,3], pos=[ 0,0, 3], angle=45, axis=[0,1,
↪0])
```

(continues on next page)

(continued from previous page)

```
>>> c = Collection(s1)
>>> #create positions
>>> xs = np.linspace(-8,8,100)
>>> zs = np.linspace(-6,6,100)
>>> posis = [[x,0,z] for z in zs for x in xs]
>>> #calculate fields
>>> Bs = c.getBsweep(posis)
>>> #reshape array and calculate amplitude
>>> Bs = np.array(Bs).reshape([100,100,3])
>>> Bamp = np.linalg.norm(Bs,axis=2)
>>> X,Z = np.meshgrid(xs,zs)
>>> # Define figure
>>> fig = plt.figure()
>>> ## Define ax for 2D
>>> ax1 = fig.add_subplot(1, 2, 1, axisbelow=True)
>>> ## Define ax for 3D displaySystem
>>> ax2 = fig.add_subplot(1, 2, 2, axisbelow=True,projection='3d')
>>> ## field plot 2D
>>> ax1.contourf(X,Z,Bamp,100,cmap='rainbow')
>>> U,V = Bs[:,:,0], Bs[:,:,2]
>>> ax1.streamplot(X, Z, U, V, color='k', density=2)
>>> ## plot Collection system in 3D ax subplot
>>> c.displaySystem(subplotAx=ax2)
```

Raises `AssertionError` – If Marker position list is poorly defined. i.e. listOfPos=(x,y,z) instead of lisOfPos=[(x,y,z)]

1.7.1 Subpackages

magpylib.math package

This module includes several practical functions for working with axis-angle relations and genmeralized rotations.

`magpylib.math.`**`randomAxis`**`()`

This function generates a random *axis* (3-vector of length 1) from an equal angular distribution using a Monte-Carlo scheme.

Returns **axis** – A random axis from an equal angular distribution of length 1

Return type arr3

Example

```
>>> magpylib as magPy
>>> ax = magPy.math.randomAxis()
>>> print(ax)
  [-0.24834468  0.96858637  0.01285925]
```

`magpylib.math.`**`randomAxisV`**`(`*`N`*`)`

This is the vectorized version of randomAxis(). It generates an N-sized vector of random *axes* (3-vector of length 1) from equal angular distributions using a MonteCarlo scheme.

Parameters **N** (*int*) – Size of random axis vector.

Returns **axes** – A vector of random axes from an equal angular distribution of length 1.

Return type Nx3 arr

Example

```
>>> import magpylib as magpy
>>> import magpylib as magpy
>>> AXS = magpy.math.randomAxisV(3)
>>> print(AXS)
>>> # Output: [[ 0.39480364 -0.53600779 -0.74620757]
... [ 0.02974442  0.10916333  0.9935787 ]
... [-0.54639126  0.76659756 -0.33731997]]
```

magpylib.math.**axisFromAngles**(*angles*)

This function generates an *axis* (3-vector of length 1) from two *angles* = [phi,th] that are defined as in spherical coordinates. phi = azimuth angle, th = polar angle. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters **angles** (*vec2 [deg]*) – The two angels [phi,th], azimuth and polar, in units of deg.

Returns **axis** – An axis of length that is oriented as given by the input angles.

Return type arr3

Example

```
>>> magpylib as magPy
>>> angles = [90,90]
>>> ax = magPy.math.axisFromAngles(angles)
>>> print(ax)
  [0.0  1.0  0.0]
```

magpylib.math.**axisFromAnglesV**(*ANG*)

This is the vectorized version of axisFromAngles(). It generates an Nx3 array of axis vectors from the Nx2 array of input angle pairs angles. Each angle pair is (phi,theta) which are azimuth and polar angle of a spherical coordinate system respectively.

Parameters **ANG** (*arr Nx2 [deg]*) – An N-sized array of angle pairs [phi th], azimuth and polar, in units of deg.

Returns **AXIS** – An N-sized array of unit axis vectors oriented as given by the input ANG.

Return type arr Nx3

Example

```
>>> import magpylib as magpy
>>> import numpy as np
>>> ANGS = np.array([[0,90],[90,180],[90,0]])
>>> AX = magpy.math.axisFromAnglesV(ANGS)
>>> print(np.around(AX,4))
>>> # Output: [[1.  0. 0.]  [0. 0. -1.]  [0. 0. 1.]]
```

magpylib.math.**anglesFromAxis**(*axis*)

This function takes an arbitrary *axis* (3-vector) and returns the orientation given by the *angles* = [phi,th] that are defined as in spherical coordinates. phi = azimuth angle, th = polar angle. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters **axis** (*vec3*) – Arbitrary input axis that defines an orientation.

Returns **angles** – The angles [phi,th], azimuth and polar, that anchorrespond to the orientation given by the input axis.

Return type arr2 [deg]

Example

```
>>> magpylib as magPy
>>> axis = [1,1,0]
>>> angles = magPy.math.anglesFromAxis(axis)
>>> print(angles)
  [45. 90.]
```

magpylib.math.**anglesFromAxisV**(*AXIS*)

This is the vectorized version of anglesFromAxis(). It takes an Nx3 array of axis-vectors and returns an Nx2 array of angle pairs. Each angle pair is (phi,theta) which are azimuth and polar angle in a spherical coordinate system respectively.

Parameters **AXIS** (*arr Nx3*) – N-sized array of axis-vectors (do not have to be not be normalized).

Returns **ANGLES** – N-sized array of angle pairs [phi,th], azimuth and polar, that chorrespond to the orientations given by the input axis vectors in a spherical coordinate system.

Return type arr Nx2 [deg]

Example

```
>>> import numpy as np
>>> import magpylib as magpy
>>> AX = np.array([[0,0,1],[0,0,1],[1,0,0]])
>>> ANGS = magpy.math.anglesFromAxisV(AX)
>>> print(ANGS)
>>> # Output: [[0. 0.]  [90. 90.]  [0. 90.]])
```

magpylib.math.**angleAxisRotation**(*position, angle, axis, anchor=[0, 0, 0]*)

This function uses angle-axis rotation to rotate the *position* vector by the *angle* argument about an axis defined by the *axis* vector which passes through the center of rotation *anchor* vector. Scalar input is either integer or float.Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **position** (*vec3*) – Input position to be rotated.
- **angle** (*scalar [deg]*) – Angle of rotation in untis of [deg]
- **axis** (*vec3*) – Axis of rotation
- **anchor** (*vec3*) – The Center of rotation which defines the position of the axis of rotation

Returns **newPosition** – Rotated position

Return type arr3

Example

```
>>> magpylib as magPy
>>> from numpy import pi
>>> position0 = [1,1,0]
>>> angle = -90
>>> axis = [0,0,1]
>>> centerOfRotation = [1,0,0]
>>> positionNew = magPy.math.angleAxisRotation(position0,angle,axis,
↪anchor=centerOfRotation)
>>> print(positionNew)
  [2. 0. 0.]
```

magpylib.math.**angleAxisRotationV**(*POS*, *ANG*, *AXIS*, *ANCHOR*)

This is the vectorized version of angleAxisRotation(). Each entry of POS (arrNx3) is rotated according to the angles ANG (arrN), about the axis vectors AXS (arrNx3) which pass throught the anchors ANCH (arrNx3) where N refers to the length of the input vectors.

Parameters

- **POS** (*arrNx3*) – The input vectors to be rotated.
- **ANG** (*arrN [deg]*) – Rotation angles in units of [deg].
- **AXIS** (*arrNx3*) – Vector of rotation axes.
- **anchor** (*arrNx3*) – Vector of rotation anchors.

Returns

- **newPOS** (*arrNx3*) – Vector of rotated positions.
- *>>> import magpylib as magpy*
- *>>> import numpy as np*
- *>>> POS = np.array([[1,0,0]]*5) # avoid this slow Python loop*
- *>>> ANG = np.linspace(0,180,5)*
- *>>> AXS = np.array([[0,0,1]]*5) # avoid this slow Python loop*
- *>>> ANCH = np.zeros((5,3))*
- *>>> POSnew = magpy.math.angleAxisRotationV(POS,ANG,AXS,ANCH)*
- *>>> print(np.around(POSnew,4))*
- **>>> # Output** (*[[1. 0. 0.]*)
- *... [0.7071 0.7071 0.]*
- *... [0. 1. 0.]*
- *... [-0.7071 0.7071 0.]*
- *... [-1. 0. 0.]]*

magpylib.source package

All available sources are collected here, accessible through the following subpackages: *magnet*, *current* and *moment*.

Subpackages

magpylib.source.current package

This subpackage provides the current classes that are used for field computation. They include the classes *Circular* (a current loop) and *Line* (a line current running along given vertices).

class magpylib.source.current.**Circular**(*curr=0.0*, *dim=0.0*, *pos=(0.0, 0.0, 0.0)*, *angle=0.0*, *axis=(0.0, 0.0, 1.0)*)

Bases: magpylib._lib.classes.base.LineCurrent

A circular line current loop with diameter *dim* and a current *curr* flowing in positive orientation. In the canonical basis (position=[0,0,0], angle=0.0, axis=[0,0,1]) the loop lies in the x-y plane with the origin at its center. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **curr** (*scalar [A]*) – Set current in loop in units of [A]
- **dim** (*float [mm]*) – Set diameter of current loop in units of [mm]
- **pos=[0,0,0]** (*vec3 [mm]*) – Set position of the center of the current loop in units of [mm].
- **angle=0.0** (*scalar [deg]*) – Set angle of orientation of current loop in units of [deg].
- **axis=[0,0,1]** (*vec3 []*) – Set axis of orientation of the current loop.

current
Current in loop in units of [A]

Type float [A]

dimension
Loop diameter in units of [mm]

Type float [mm]

position
Position of center of loop in units of [mm]

Type arr3 [mm]

angle
Angle of orientation of the current loop.

Type float [deg]

axis
Axis of orientation of the current loop.

Type arr3 []

Example

```
>>> from magpylib import source
>>> cd = source.current.Circular(curr=10,dim=2)
>>> B = cd.getB([0,0,2])
>>> print(B)
  [0.         0.         0.56198518]
```

Note: The following Methods are available to all sources objects.

getB(*pos*)
This method returns the magnetic field vector generated by the source at the argument position *pos* in units of [mT]

Parameters **pos** (*vec3 [mm] Position or list of Positions where magnetic field*) – should be determined.

Returns **magnetic field vector** – position *pos* generated by the source in units of [mT].

Return type arr3 [mT] Magnetic field at the argument

move(*displacement*)
This method moves the source by the argument vector *displacement*. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters **displacement** (*vec3 [mm]*) – Set displacement vector

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1,pos=[1,2,3])
>>> print(pm.position)
    [1. 2. 3.]
>>> pm.move([3,2,1])
>>> print(pm.position)
    [4. 4. 4.]
```

rotate(*angle*, *axis*, *anchor='self.position'*)
This method rotates the source about *axis* by *angle*. The axis passes through the center of rotation anchor. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **angle** (*scalar [deg]*) – Set angle of rotation in units of [deg]
- **axis** (*vec3 []*) – Set axis of rotation
- **anchor** (*vec3 [mm]*) – Specify the Center of rotation which defines the position of the axis of rotation. If not specified the source will rotate about its own center.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000], dim=1)
>>> print(pm.position, pm.angle, pm.axis)
  [0. 0. 0.] 0.0 [0. 0. 1.]
```

(continues on next page)

(continued from previous page)

```
>>> pm.rotate(90, [0,1,0], anchor=[1,0,0])
>>> print(pm.position, pm.angle, pm.axis)
  [1., 0., 1.] 90.0 [0., 1., 0.]
```

setOrientation(*angle*, *axis*)

This method sets a new source orientation given by *angle* and *axis*. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **angle** (*scalar [deg]*) – Set new angle of source orientation.
- **axis** (*vec3 []*) – Set new axis of source orientation.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> print([pm.angle,pm.axis])
    [0.0, array([0., 0., 1.])]
>>> pm.setOrientation(45,[0,1,0])
>>> print([pm.angle,pm.axis])
    [45.0, array([0., 1., 0.])]
```

setPosition(*newPos*)

This method moves the source to the position given by the argument vector *newPos*. Vector input format can be either list, tuple or array of any data type (float, int)

Parameters **newPos** (*vec3 [mm]*) – Set new position of the source.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> print(pm.position)
    [0. 0. 0.]
>>> pm.setPosition([5,5,5])
>>> print(pm.position)
    [5. 5. 5.]
```

class magpylib.source.current.**Line**(*curr=0.0, vertices=typing.List[typing.Tuple[~x_i, ~y_i, ~z_i]], pos=(0.0, 0.0, 0.0), angle=0.0, axis=(0.0, 0.0, 1.0)*)

Bases: magpylib._lib.classes.base.LineCurrent

A line current flowing along linear segments from vertex to vertex given by a list of positions *vertices* in the canonical basis (position=[0,0,0], angle=0.0, axis=[0,0,1]). Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **curr** (*scalar [A]*) – Set current in loop in units of [A]
- **vertices** (*vecNx3 [mm]*) – N positions given in units of [mm] that make up N-1 linear segments along which the current *curr* flows, starting from the first position and ending with the last one. [[x,y,z], [x,y,z], ...] "[pos1,pos2,...]"
- **pos=[0,0,0]** (*vec3 [mm]*) – Set reference position of the current distribution in units of [mm].
- **angle=0.0** (*scalar [deg]*) – Set angle of orientation of current distribution in units of [deg].
- **axis=[0,0,1]** (*vec3 []*) – Set axis of orientation of the current distribution.

current
Current flowing along line in units of [A].

Type float [A]

vertices
Positions of line current vertices in units of [mm].

Type arrNx3 [mm]

position
Reference position of line current in units of [mm].

Type arr3 [mm]

angle
Angle of orientation of line current in units of [deg].

Type float [deg]

axis
Axis of orientation of the line current.

Type arr3 []

Examples

```
>>> from magpylib import source
>>> from numpy import sin,cos,pi,linspace
>>> vertices = [[cos(phi),sin(phi),0] for phi in linspace(0,2*pi,36)]
>>> cd = source.current.Line(curr=10,vertices=vertices)
>>> B = cd.getB([0,0,2])
>>> print(B)
  [-6.24500451e-17  1.73472348e-18  5.59871233e-01]
```

Note: The following Methods are available to all sources objects.

getB(*pos*)
This method returns the magnetic field vector generated by the source at the argument position *pos* in units of [mT]

Parameters **pos** (*vec3 [mm] Position or list of Positions where magnetic field*) – should be determined.

Returns **magnetic field vector** – position *pos* generated by the source in units of [mT].

Return type arr3 [mT] Magnetic field at the argument

move(*displacement*)

This method moves the source by the argument vector *displacement*. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters **displacement** (*vec3 [mm]*) – Set displacement vector

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1,pos=[1,2,3])
>>> print(pm.position)
    [1. 2. 3.]
>>> pm.move([3,2,1])
>>> print(pm.position)
    [4. 4. 4.]
```

rotate(*angle*, *axis*, *anchor='self.position'*)

This method rotates the source about *axis* by *angle*. The axis passes through the center of rotation anchor. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **angle** (*scalar [deg]*) – Set angle of rotation in units of [deg]
- **axis** (*vec3 []*) – Set axis of rotation
- **anchor** (*vec3 [mm]*) – Specify the Center of rotation which defines the position of the axis of rotation. If not specified the source will rotate about its own center.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000], dim=1)
>>> print(pm.position, pm.angle, pm.axis)
  [0. 0. 0.] 0.0 [0. 0. 1.]
>>> pm.rotate(90, [0,1,0], anchor=[1,0,0])
>>> print(pm.position, pm.angle, pm.axis)
  [1., 0., 1.] 90.0 [0., 1., 0.]
```

setOrientation(*angle*, *axis*)

This method sets a new source orientation given by *angle* and *axis*. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **angle** (*scalar [deg]*) – Set new angle of source orientation.

- **axis** (*vec3 []*) – Set new axis of source orientation.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> print([pm.angle,pm.axis])
    [0.0, array([0., 0., 1.])]
>>> pm.setOrientation(45,[0,1,0])
>>> print([pm.angle,pm.axis])
    [45.0, array([0., 1., 0.])]
```

setPosition(*newPos*)

This method moves the source to the position given by the argument vector *newPos*. Vector input format can be either list, tuple or array of any data type (float, int)

Parameters **newPos** (*vec3 [mm]*) – Set new position of the source.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> print(pm.position)
    [0. 0. 0.]
>>> pm.setPosition([5,5,5])
>>> print(pm.position)
    [5. 5. 5.]
```

magpylib.source.magnet package

This subpackage provides the permanent magnet classes that are used for field computation. They include *Box* (cuboid shape), *Cylinder* (cylindrical shape) *Sphere* (spherical shape) and *Facet* (triangular surface of magnet body).

class magpylib.source.magnet.**Box**(*mag=(0.0, 0.0, 0.0)*, *dim=(0.0, 0.0, 0.0)*, *pos=(0.0, 0.0, 0.0)*, *angle=0.0*, *axis=(0.0, 0.0, 1.0)*)

Bases: magpylib._lib.classes.base.HomoMag

A homogeneously magnetized cuboid magnet. In the canonical basis (position=[0,0,0], angle=0.0, axis=[0,0,1]) the magnet has the origin at its geometric center and the sides of the box are parallel to the basis vectors. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **mag** (*vec3 [mT]*) – Set magnetization vector of magnet in units of [mT].
- **dim** (*vec3 [mm]*) – Set the size of the box. dim=[a,b,c] which anchorresponds to the three side lenghts of the box in units of [mm].
- **pos=[0,0,0]** (*vec3 [mm]*) – Set position of the center of the magnet in units of [mm].

- **angle=0.0** (*scalar [deg]*) – Set angle of orientation of magnet in units of [deg].
- **axis=[0,0,1]** (*vec3 []*) – Set axis of orientation of the magnet.

magnetization
Magnetization vector of box in units of [mT].

Type arr3 [mT]

dimension
Magnet dimension=[a,b,c] which anchorrespond to the three side lenghts of the box in units of [mm] in x-,y- and z-direction respectively in the canonical basis.

Type arr3 [mm]

position
Position of the center of the magnet in units of [mm].

Type arr3 [mm]

angle
Angle of orientation of the magnet in units of [deg].

Type float [deg]

axis
Axis of orientation of the magnet.

Type arr3 []

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Box(mag=[0,0,1000],dim=[1,1,1])
>>> B = pm.getB([1,0,1])
>>> print(B)
  [4.29223532e+01 1.76697482e-14 1.37461635e+01]
```

Note: The following Methods are available to all sources objects.

getB(*pos*)
This method returns the magnetic field vector generated by the source at the argument position *pos* in units of [mT]

Parameters **pos** (*vec3 [mm] Position or list of Positions where magnetic field*) – should be determined.

Returns **magnetic field vector** – position *pos* generated by the source in units of [mT].

Return type arr3 [mT] Magnetic field at the argument

move(*displacement*)
This method moves the source by the argument vector *displacement*. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters **displacement** (*vec3 [mm]*) – Set displacement vector

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1,pos=[1,2,3])
>>> print(pm.position)
    [1. 2. 3.]
>>> pm.move([3,2,1])
>>> print(pm.position)
    [4. 4. 4.]
```

rotate (*angle*, *axis*, *anchor='self.position'*)

This method rotates the source about *axis* by *angle*. The axis passes through the center of rotation anchor. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **angle** (*scalar [deg]*) – Set angle of rotation in units of [deg]
- **axis** (*vec3 []*) – Set axis of rotation
- **anchor** (*vec3 [mm]*) – Specify the Center of rotation which defines the position of the axis of rotation. If not specified the source will rotate about its own center.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000], dim=1)
>>> print(pm.position, pm.angle, pm.axis)
  [0. 0. 0.] 0.0 [0. 0. 1.]
>>> pm.rotate(90, [0,1,0], anchor=[1,0,0])
>>> print(pm.position, pm.angle, pm.axis)
  [1., 0., 1.] 90.0 [0., 1., 0.]
```

setOrientation (*angle*, *axis*)

This method sets a new source orientation given by *angle* and *axis*. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **angle** (*scalar [deg]*) – Set new angle of source orientation.
- **axis** (*vec3 []*) – Set new axis of source orientation.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> print([pm.angle,pm.axis])
    [0.0, array([0., 0., 1.])]
```

(continues on next page)

(continued from previous page)

```
>>> pm.setOrientation(45,[0,1,0])
>>> print([pm.angle,pm.axis])
    [45.0, array([0., 1., 0.])]
```

setPosition(*newPos*)

This method moves the source to the position given by the argument vector *newPos*. Vector input format can be either list, tuple or array of any data type (float, int)

Parameters **newPos** (*vec3 [mm]*) – Set new position of the source.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> print(pm.position)
    [0. 0. 0.]
>>> pm.setPosition([5,5,5])
>>> print(pm.position)
    [5. 5. 5.]
```

class magpylib.source.magnet.**Cylinder**(*mag=(0.0, 0.0, 0.0)*, *dim=(0.0, 0.0)*, *pos=(0.0, 0.0, 0.0)*, *angle=0.0*, *axis=(0.0, 0.0, 1.0)*, *iterDia=50*)

Bases: magpylib._lib.classes.base.HomoMag

A homogeneously magnetized cylindrical magnet. The magnet is initialized in the canonical basis (position=[0,0,0], angle=0.0, axis=[0,0,1]) with the geometric center at the origin and the central symmetry axis pointing in z-direction so that the circular bottom lies in a plane parallel to the xy-plane. Scalar input is either integer or float and reflects a round bottom. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **mag** (*vec3 [mT]*) – Set magnetization vector of magnet in units of [mT].
- **dim** (*vec2 [mm]*) – Set the size of the cylinder. dim=[D,H] which are diameter and height of the cylinder in units of [mm] respectively.
- **pos=[0,0,0]** (*vec3 [mm]*) – Set position of the center of the magnet in units of [mm].
- **angle=0.0** (*scalar [deg]*) – Set angle of orientation of magnet in units of [deg].
- **axis=[0,0,1]** (*vec3 []*) – Set axis of orientation of the magnet.
- **iterDia=50** (*int []*) – Set number of iterations for calculation of B-field from non-axial magnetization. Lower values will make the calculation faster but less precise.

magnetization

Magnetization vector of magnet in units of [mT].

Type arr3 [mT]

dimension

Magnet dimension=[d,h] which anchorrespond to diameter and height of the cylinder in units of [mm].

Type arr2 [mm]

position
Position of the center of the magnet in units of [mm].

Type arr3 [mm]

angle
Angle of orientation of the magnet in units of [deg].

Type float [deg]

axis
Axis of orientation of the magnet.

Type arr3 []

iterDia
Number of iterations for calculation of B-field from non-axial magnetization. Lower values will make the calculation faster but less precise.

Type int []

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Cylinder(mag=[0,0,1000],dim=[1,1])
>>> B = pm.getB([1,0,1])
>>> print(B)
  [34.31662243  0.         10.16090915]
```

Note: The following Methods are available to all sources objects.

getB(*pos*)
This method returns the magnetic field vector generated by the source at the argument position *pos* in units of [mT]

Parameters **pos** (*vec3 [mm] Position or list of Positions where magnetic field*) – should be determined.

Returns **magnetic field vector** – position *pos* generated by the source in units of [mT].

Return type arr3 [mT] Magnetic field at the argument

move(*displacement*)
This method moves the source by the argument vector *displacement*. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters **displacement** (*vec3 [mm]*) – Set displacement vector

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1,pos=[1,2,3])
>>> print(pm.position)
```

(continues on next page)

(continued from previous page)

```
    [1. 2. 3.]
>>> pm.move([3,2,1])
>>> print(pm.position)
    [4. 4. 4.]
```

rotate(*angle*, *axis*, *anchor='self.position'*)

This method rotates the source about *axis* by *angle*. The axis passes through the center of rotation anchor. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **angle** (*scalar [deg]*) – Set angle of rotation in units of [deg]
- **axis** (*vec3 []*) – Set axis of rotation
- **anchor** (*vec3 [mm]*) – Specify the Center of rotation which defines the position of the axis of rotation. If not specified the source will rotate about its own center.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000], dim=1)
>>> print(pm.position, pm.angle, pm.axis)
  [0. 0. 0.] 0.0 [0. 0. 1.]
>>> pm.rotate(90, [0,1,0], anchor=[1,0,0])
>>> print(pm.position, pm.angle, pm.axis)
  [1., 0., 1.] 90.0 [0., 1., 0.]
```

setOrientation(*angle*, *axis*)

This method sets a new source orientation given by *angle* and *axis*. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **angle** (*scalar [deg]*) – Set new angle of source orientation.
- **axis** (*vec3 []*) – Set new axis of source orientation.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> print([pm.angle,pm.axis])
    [0.0, array([0., 0., 1.])]
>>> pm.setOrientation(45,[0,1,0])
>>> print([pm.angle,pm.axis])
    [45.0, array([0., 1., 0.])]
```

setPosition(*newPos*)
This method moves the source to the position given by the argument vector *newPos*. Vector input format can be either list, tuple or array of any data type (float, int)

Parameters **newPos** (*vec3 [mm]*) – Set new position of the source.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> print(pm.position)
    [0. 0. 0.]
>>> pm.setPosition([5,5,5])
>>> print(pm.position)
    [5. 5. 5.]
```

class magpylib.source.magnet.**Sphere**(*mag=(0.0, 0.0, 0.0), dim=0.0, pos=(0.0, 0.0, 0.0), angle=0.0, axis=(0.0, 0.0, 1.0)*)
Bases: magpylib._lib.classes.base.HomoMag

A homogeneously magnetized sphere. The magnet is initialized in the canonical basis (position=[0,0,0], angle=0.0, axis=[0,0,1]) with the center at the origin. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **mag** (*vec3 [mT]*) – Set magnetization vector of magnet in units of [mT].
- **dim** (*float [mm]*) – Set diameter of the sphere in units of [mm].
- **pos=[0,0,0]** (*vec3 [mm]*) – Set position of the center of the magnet in units of [mm].
- **angle=0.0** (*scalar [deg]*) – Set angle of orientation of magnet in units of [deg].
- **axis=[0,0,1]** (*vec3 []*) – Set axis of orientation of the magnet.

magnetization
Magnetization vector of magnet in units of [mT].

Type arr3 [mT]

dimension
Sphere diameter in units of [mm].

Type float [mm]

position
Position of the center of the magnet in units of [mm].

Type arr3 [mm]

angle
Angle of orientation of the magnet in units of [deg].

Type float [deg]

axis
Axis of orientation of the magnet.

Type arr3 []

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> B = pm.getB([1,0,1])
>>> print(B)
  [22.09708691  0.          7.36569564]
```

Note: The following Methods are available to all sources objects.

---

getB(*pos*)

This method returns the magnetic field vector generated by the source at the argument position *pos* in units of [mT]

Parameters **pos** (*vec3 [mm] Position or list of Positions where magnetic field*) – should be determined.

Returns **magnetic field vector** – position *pos* generated by the source in units of [mT].

Return type arr3 [mT] Magnetic field at the argument

move(*displacement*)

This method moves the source by the argument vector *displacement*. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters **displacement** (*vec3 [mm]*) – Set displacement vector

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1,pos=[1,2,3])
>>> print(pm.position)
    [1. 2. 3.]
>>> pm.move([3,2,1])
>>> print(pm.position)
    [4. 4. 4.]
```

rotate(*angle*, *axis*, *anchor='self.position'*)

This method rotates the source about *axis* by *angle*. The axis passes through the center of rotation anchor. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **angle** (*scalar [deg]*) – Set angle of rotation in units of [deg]
- **axis** (*vec3 []*) – Set axis of rotation
- **anchor** (*vec3 [mm]*) – Specify the Center of rotation which defines the position of the axis of rotation. If not specified the source will rotate about its own center.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000], dim=1)
>>> print(pm.position, pm.angle, pm.axis)
  [0. 0. 0.] 0.0 [0. 0. 1.]
>>> pm.rotate(90, [0,1,0], anchor=[1,0,0])
>>> print(pm.position, pm.angle, pm.axis)
  [1., 0., 1.] 90.0 [0., 1., 0.]
```

setOrientation(*angle*, *axis*)

This method sets a new source orientation given by *angle* and *axis*. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **angle** (*scalar [deg]*) – Set new angle of source orientation.
- **axis** (*vec3 []*) – Set new axis of source orientation.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> print([pm.angle,pm.axis])
    [0.0, array([0., 0., 1.])]
>>> pm.setOrientation(45,[0,1,0])
>>> print([pm.angle,pm.axis])
    [45.0, array([0., 1., 0.])]
```

setPosition(*newPos*)

This method moves the source to the position given by the argument vector *newPos*. Vector input format can be either list, tuple or array of any data type (float, int)

Parameters **newPos** (*vec3 [mm]*) – Set new position of the source.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> print(pm.position)
    [0. 0. 0.]
>>> pm.setPosition([5,5,5])
>>> print(pm.position)
    [5. 5. 5.]
```

class magpylib.source.magnet.**Facet**

Bases: magpylib._lib.classes.base.HomoMag

WIP

getB(*pos*)

This method returns the magnetic field vector generated by the source at the argument position *pos* in units of [mT]

Parameters **pos** (*vec3 [mm] Position or list of Positions where magnetic field*) – should be determined.

Returns **magnetic field vector** – position *pos* generated by the source in units of [mT].

Return type arr3 [mT] Magnetic field at the argument

move(*displacement*)

This method moves the source by the argument vector *displacement*. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters **displacement** (*vec3 [mm]*) – Set displacement vector

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1,pos=[1,2,3])
>>> print(pm.position)
    [1. 2. 3.]
>>> pm.move([3,2,1])
>>> print(pm.position)
    [4. 4. 4.]
```

rotate(*angle*, *axis*, *anchor='self.position'*)

This method rotates the source about *axis* by *angle*. The axis passes through the center of rotation anchor. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **angle** (*scalar [deg]*) – Set angle of rotation in units of [deg]
- **axis** (*vec3 []*) – Set axis of rotation
- **anchor** (*vec3 [mm]*) – Specify the Center of rotation which defines the position of the axis of rotation. If not specified the source will rotate about its own center.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000], dim=1)
>>> print(pm.position, pm.angle, pm.axis)
  [0. 0. 0.] 0.0 [0. 0. 1.]
>>> pm.rotate(90, [0,1,0], anchor=[1,0,0])
>>> print(pm.position, pm.angle, pm.axis)
  [1., 0., 1.] 90.0 [0., 1., 0.]
```

setOrientation(*angle*, *axis*)
This method sets a new source orientation given by *angle* and *axis*. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **angle** (*scalar [deg]*) – Set new angle of source orientation.
- **axis** (*vec3 []*) – Set new axis of source orientation.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> print([pm.angle,pm.axis])
    [0.0, array([0., 0., 1.])]
>>> pm.setOrientation(45,[0,1,0])
>>> print([pm.angle,pm.axis])
    [45.0, array([0., 1., 0.])]
```

setPosition(*newPos*)
This method moves the source to the position given by the argument vector *newPos*. Vector input format can be either list, tuple or array of any data type (float, int)

Parameters **newPos** (*vec3 [mm]*) – Set new position of the source.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> print(pm.position)
    [0. 0. 0.]
>>> pm.setPosition([5,5,5])
>>> print(pm.position)
    [5. 5. 5.]
```

magpylib.source.moment package

This subpackge includes magnetic moment classes for field computation. Currently it includes only the *Dipole* (magnetisches dipol moment) class.

class magpylib.source.moment.**Dipole**(*moment=(0.0, 0.0, 0.0)*, *pos=(0.0, 0.0, 0.0)*, *angle=0.0*, *axis=(0.0, 0.0, 1.0)*)
Bases: magpylib._lib.classes.base.MagMoment

This class represents a magnetic dipole. The dipole is constructed such that its moment $|M|$ is given in $[mT * mm^3]$ and corresponds to the moment of a cuboid magnet with remanence field Br and Volume V such that $|M| = Br * V$. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **moment** (*vec3 [mT]*) – Set magnetic dipole moment in units of [mT*mm^3].
- **pos=[0,0,0]** (*vec3 [mm]*) – Set position of the moment in units of [mm].
- **angle=0.0** (*scalar [deg]*) – Set angle of orientation of the moment in units of [deg].
- **axis=[0,0,1]** (*vec3 []*) – Set axis of orientation of the moment.

moment
Magnetic dipole moment in units of [mT*mm^3] ($|moment| = Br * V$ of a cuboid magnet.)

Type arr3 [mT]

position
Position of the moment in units of [mm].

Type arr3 [mm]

angle
Angle of orientation of the moment in units of [deg].

Type float [deg]

axis
Axis of orientation of the moment.

Type arr3 []

Examples

```
>>> magpylib as magpy
>>> mom = magpy.source.moment.Dipole(moment=[0,0,1000])
>>> B = mom.getB([1,0,1])
>>> print(B)
  [0.33761862  0.  0.11253954]
```

Note: The following Methods are available to all source objects.

getB(*pos*)
This method returns the magnetic field vector generated by the source at the argument position *pos* in units of [mT]

Parameters **pos** (*vec3 [mm] Position or list of Positions where magnetic field*) – should be determined.

Returns **magnetic field vector** – position *pos* generated by the source in units of [mT].

Return type arr3 [mT] Magnetic field at the argument

move(*displacement*)
This method moves the source by the argument vector *displacement*. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters **displacement** (*vec3 [mm]*) – Set displacement vector

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1,pos=[1,2,3])
>>> print(pm.position)
    [1. 2. 3.]
>>> pm.move([3,2,1])
>>> print(pm.position)
    [4. 4. 4.]
```

rotate (*angle*, *axis*, *anchor='self.position'*)

This method rotates the source about *axis* by *angle*. The axis passes through the center of rotation anchor. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **angle** (*scalar [deg]*) – Set angle of rotation in units of [deg]
- **axis** (*vec3 []*) – Set axis of rotation
- **anchor** (*vec3 [mm]*) – Specify the Center of rotation which defines the position of the axis of rotation. If not specified the source will rotate about its own center.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000], dim=1)
>>> print(pm.position, pm.angle, pm.axis)
  [0. 0. 0.] 0.0 [0. 0. 1.]
>>> pm.rotate(90, [0,1,0], anchor=[1,0,0])
>>> print(pm.position, pm.angle, pm.axis)
  [1., 0., 1.] 90.0 [0., 1., 0.]
```

setOrientation (*angle*, *axis*)

This method sets a new source orientation given by *angle* and *axis*. Scalar input is either integer or float. Vector input format can be either list, tuple or array of any data type (float, int).

Parameters

- **angle** (*scalar [deg]*) – Set new angle of source orientation.
- **axis** (*vec3 []*) – Set new axis of source orientation.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> print([pm.angle,pm.axis])
    [0.0, array([0., 0., 1.])]
```

(continues on next page)

(continued from previous page)

```
>>> pm.setOrientation(45,[0,1,0])
>>> print([pm.angle,pm.axis])
    [45.0, array([0., 1., 0.])]
```

setPosition(*newPos*)

This method moves the source to the position given by the argument vector *newPos*. Vector input format can be either list, tuple or array of any data type (float, int)

Parameters **newPos** (*vec3 [mm]*) – Set new position of the source.

Returns

Return type None

Example

```
>>> from magpylib import source
>>> pm = source.magnet.Sphere(mag=[0,0,1000],dim=1)
>>> print(pm.position)
    [0. 0. 0.]
>>> pm.setPosition([5,5,5])
>>> print(pm.position)
    [5. 5. 5.]
```

magpylib.vector package

The vector subpackage includes functions for vectorized computation of the magnetic field. Use these functions only when performance is an issue, when doing 10 or more evaluations of similar sources (with different parameters).

magpylib.vector.**getBv_magnet**(*type*, *MAG*, *DIM*, *POSm*, *POSo*, *ANG=[]*, *AX=[]*, *ANCH=[]*, *Nphi0=50*)

Calculate the field of magnets using vectorized performance code.

Parameters

- **type** (*string*) – source type either 'box', 'cylinder', 'sphere'.
- **MAG** (*Nx3 numpy array float [mT]*) – vector of N magnetizations.
- **DIM** (*NxY numpy array float [mm]*) – vector of N dimensions for each evaluation. The form of this vector depends on the source type. Y=3/2/1 for box/cylinder/sphere
- **POSo** (*Nx3 numpy array float [mm]*) – vector of N positions of the observer.
- **POSm** (*Nx3 numpy array float [mm]*) – vector of N initial source positions. These positions will be adjusted by the given rotation parameters.
- **ANG=[]** (*length M list of size N numpy arrays float [deg]*) – Angles of M subsequent rotation operations applied to the N-sized POSm and the implicit source orientation.
- **AX=[]** (*length M list of Nx3 numpy arrays float []*) – Axis vectors of M subsequent rotation operations applied to the N-sized POSm and the implicit source orientation.
- **ANCH=[]** (*length M list of Nx3 numpy arrays float [mm]*) – Anchor positions of M subsequent rotations applied ot the N-sized POSm and the implicit source orientation.

- **Nphi0=50** (*integer gives number of iterations used when calculating diametral*) – magnetized cylindrical magnets.

magpylib.vector.**getBv_current**(*type*, *CURR*, *DIM*, *POSm*, *POSo*, *ANG=[]*, *AX=[]*, *ANCH=[]*)

Calculate the field of currents using vectorized performance code.

Parameters

- **type** (*string*) – source type either 'circular' or 'line'
- **MAG** (*Nx3 numpy array float [mT]*) – vector of N magnetizations.
- **DIM** (*NxY numpy array float [mm]*) – vector of N dimensions for each evaluation. The form of this vector depends on the source type. Y=1/3x3 for circular/line.
- **POSo** (*Nx3 numpy array float [mm]*) – vector of N positions of the observer.
- **POSm** (*Nx3 numpy array float [mm]*) – vector of N initial source positions. These positions will be adjusted by the given rotation parameters.
- **ANG=[]** (*length M list of size N numpy arrays float [deg]*) – Angles of M subsequent rotation operations applied to the N-sized POSm and the implicit source orientation.
- **AX=[]** (*length M list of Nx3 numpy arrays float []*) – Axis vectors of M subsequent rotation operations applied to the N-sized POSm and the implicit source orientation.
- **ANCH=[]** (*length M list of Nx3 numpy arrays float [mm]*) – Anchor positions of M subsequent rotations applied ot the N-sized POSm and the implicit source orientation.

magpylib.vector.**getBv_moment**(*type*, *MOM*, *POSm*, *POSo*, *ANG=[]*, *AX=[]*, *ANCH=[]*)

Calculate the field of magnetic moments using vectorized performance code.

Parameters

- **type** (*string*) – source type: 'dipole'
- **MOM** (*Nx3 numpy array float [mT]*) – vector of N dipole moments.
- **POSo** (*Nx3 numpy array float [mm]*) – vector of N positions of the observer.
- **POSm** (*Nx3 numpy array float [mm]*) – vector of N initial source positions. These positions will be adjusted by the given rotation parameters.
- **ANG=[]** (*length M list of size N numpy arrays float [deg]*) – Angles of M subsequent rotation operations applied to the N-sized POSm and the implicit source orientation.
- **AX=[]** (*length M list of Nx3 numpy arrays float []*) – Axis vectors of M subsequent rotation operations applied to the N-sized POSm and the implicit source orientation.
- **ANCH=[]** (*length M list of Nx3 numpy arrays float [mm]*) – Anchor positions of M subsequent rotations applied ot the N-sized POSm and the implicit source orientation.

CHAPTER 2

Index

- genindex
- modindex

Python Module Index

m

Index

P

R

S

V