Spherical Helmholtz Translation and Rotation

Release 0.3.2+3.ga28f62a

Carl Andersson

Mar 04, 2022

CONTENTS:

1 Bases **3**

2 Expansions **7**

3 Transforms **11**

4 Examples **15**
- 4.1 Simple Rotations of Surface Expansion . . . 15
  - 4.1.1 Original field . . . 17
  - 4.1.2 First rotation . . . 17
  - 4.1.3 Second rotation . . . 17
  - 4.1.4 Final rotated field . . . 17
  - 4.1.5 Reexpansion . . . 18
  - 4.1.6 Reevaluated expansion . . . 18
- 4.2 Simple Translation of a Monopole . . . 18
  - 4.2.1 Original Field . . . 19
  - 4.2.2 Translated field . . . 19
  - 4.2.3 Reexpanded field . . . 20
  - 4.2.4 Reevalulated field . . . 20

5 Indices and tables **21**

Python Module Index **23**

Index **25**

Python implementations of translation and rotation theorems for spherical solutions of the Helmholtz equation in three dimensions. This is heavily based on the book "Fast Multipole Methods for the Helmholtz Equation in Three Dimensions", by N. A. Gumerov and R. Duraiswami, Elsevier 2005.

- **Documentattion**: https://spherical-helmholtz-translation-and-rotation.readthedocs.io/en/latest/
- **Source code and issue tracker**: https://github.com/AppliedAcousticsChalmers/Spherical-Helmholtz-Translation-and-Rotation

CHAPTER

ONE

BASES

Base functions used in spherical coordinates.

This module contains classes used to calculate and organize the required base functions for the Helmholtz equation in spherical coordinates. Theese can be used to reuse the calculation of base functions for repeated evaluation of some expansion coefficients at specific points in space.

LegendrePolynomials	Order only Legendre polynomials.
AssociatedLegendrePolynomials	Associated Legendre polynomials with both order and mode.
RegularRadialBase	Regular radial base, for interior problems.
SingularRadialBase	Singular radial base, for exterior problems.
DualRadialBase	Dual radial base, for interior and problems.
SphericalHarmonics	Spherical harmonics.
RegularBase	Regular multipole base, for interior problems.
SingularBase	Singular multipole base, for exterior problems.
DualBase	Dual multipole base, for interior and exterior problems.

class shetar.bases.**LegendrePolynomials**(*order*, *position=None*, *colatitude=None*, *x=None*, *defer_evaluation=False*)

Order only Legendre polynomials.

Parameters

- **order** (*int*) – The highest order included for the base.
- **position** (*None, optional*) – Position specifier, see `coordinates` for more info.
- **colatitude** (*None, optional*) – Colatitude of a spatial position. Will be used as cos(colatitude) as the input to the base function.
- **x** (*None, optional*) – Arbitrary input directly to the base function.
- **defer_evaluation** (*bool, optional*) – Do not calculate the values upon initialization of the object.

class shetar.bases.**AssociatedLegendrePolynomials**(*order*, *position=None*, *colatitude=None*, *x=None*, *defer_evaluation=False*)

Associated Legendre polynomials with both order and mode.

Note that this is using an orthonormal definition of the basis functions, i.e. not the one traditionally found in textbooks.

See *LegendrePolynomials* for details on parameters.

class shetar.bases.**RadialBaseClass**(*order*, *position=None*, *radius=None*, *wavenumber=None*, *defer_evaluation=False*)

Parent class for all radial bases.

This class should not be instantiated, only inherited from.

Parameters

- **order** (*int*) – The highest order included for the base.
- **position** (*None, optional*) – Position specifier, see `coordinates` for more info.
- **radius** (*None, optional*) – Radius where the base is evaluated, in meters.
- **wavenumber** (*None, optional*) – Wavenumber where the base is evaluated, in rad/m.
- **defer_evaluation** (*bool, optional*) – Do not calculate the values upon initialization of the object.

class shetar.bases.**RegularRadialBase**(*order*, *position=None*, *radius=None*, *wavenumber=None*, *defer_evaluation=False*)

Regular radial base, for interior problems.

This is also known as the spherical Bessel function. See *RadialBaseClass* for details on the parameters.

class shetar.bases.**SingularRadialBase**(*order*, *position=None*, *radius=None*, *wavenumber=None*, *defer_evaluation=False*)

Singular radial base, for exterior problems.

This is also known as the spherical Hankel function. See *RadialBaseClass* for details on the parameters.

class shetar.bases.**DualRadialBase**(*order*, *position=None*, *radius=None*, *wavenumber=None*, *defer_evaluation=False*)

Dual radial base, for interior and problems.

This calculates both the regular and singular radial base functions. See *RadialBaseClass* for details on the parameters.

class shetar.bases.**SphericalHarmonics**(*order*, *position=None*, *colatitude=None*, *azimuth=None*, *defer_evaluation=False*, **args*, ***kwargs*)

Spherical harmonics.

This evaluate the (surface) spherical harmonics. The values are with the same phase and scaling conventions as e.g. scipy.

Parameters

- **order** (*int*) – The highest order included for the base.
- **position** (*None, optional*) – Position specifier, see `coordinates` for more info.
- **colatitude** (*None, optional*) – Colatitude angle where to evaluate the spherical harmonics, in [0,].
- **azimuth** (*None, optional*) – Azimuth angle where to evaluate the spherical harmonics, in radians.
- **defer_evaluation** (*bool, optional*) – Do not calculate the values upon initialization of the object.

class shetar.bases.**MultipoleBase**(*order*, *position=None*, *wavenumber=None*, *radius=None*, *colatitude=None*, *azimuth=None*, *defer_evaluation=False*)

Parent class for all multipole bases.

This class should not be instantiated, only inherited from.

Parameters

- **order** (*int*) – The highest order included for the base.
- **position** (*None, optional*) – Position specifier, see `coordinates` for more info.
- **wavenumber** (*None, optional*) – Wavenumber where the base is evaluated, in rad/m.
- **radius** (*None, optional*) – Radius where the base is evaluated, in meters.
- **colatitude** (*None, optional*) – Colatitude angle where to evaluate the spherical harmonics, in [0,].
- **azimuth** (*None, optional*) – Azimuth angle where to evaluate the spherical harmonics, in radians.
- **defer_evaluation** (*bool, optional*) – Do not calculate the values upon initialization of the object.

class shetar.bases.**RegularBase**(*order*, *position=None*, *wavenumber=None*, *radius=None*, *colatitude=None*, *azimuth=None*, *defer_evaluation=False*)

Regular multipole base, for interior problems.

See *MultipoleBase* for details on the parameters.

class shetar.bases.**SingularBase**(*order*, *position=None*, *wavenumber=None*, *radius=None*, *colatitude=None*, *azimuth=None*, *defer_evaluation=False*)

Singular multipole base, for exterior problems.

See *MultipoleBase* for details on the parameters.

class shetar.bases.**DualBase**(**args*, ***kwargs*)

Dual multipole base, for interior and exterior problems.

This calculates both the singular and regular multipole bases. See *MultipoleBase* for details on the parameters.

CHAPTER

TWO

EXPANSIONS

Organization of the various expansions coefficients.

This provides convenience classes for storing expansion coefficients with the cirrect indexing conventions for the package. There's also a few functions which can create some commonly used expansions, e.g. monopoles or plane waves. The classes with an explicit domain also has methods to conveniently evaluate the expansion at some positions in space, or to apply certain transforms.

Expansion	Basic expansion without notion of domain.
SphericalSurfaceExpansion	Holds spherical harmonics expansions.
InteriorExpansion	Holds interior expansions.
ExteriorExpansion	Holds exterior expansions.
plane_wave	Create an expansion of a plane wave.
monopole	Create a monopole source expansion.
circular_ring	Create a circular ring source expansion.

class shetar.expansions.**Expansion**(*order=None*, *data=None*, *wavenumber=None*, *shape=None*)

Basic expansion without notion of domain.

This can be used as a plain holder for the coefficients which has both order and mode, e.g. spherical harmonics expansion coefficients. Instances of this class has no inherent notion of what is the correct base to use, so it cannot create them automatically.

Parameters

- **order** (*int*) – The highest order included in the expansion. If instantiated with raw data, no order is required for the instantiated since it is calculated from the shape of the data. If `data` if not given, `order` is required and an all-zero expansion is created. If both `data` and `order` are given, the have to agree on the order.
- **data** (*array_like*) – Raw data for expansion coefficients. Has to have the correct indexing conventions with the coefficient axis last.
- **wavenumber** (*None, optional*) – Optional wavenumber of the expansion. This has to be broadcastable with the spatial shape of the data.
- **shape** (*None, optional*) – Spatial shape for all-zero expansions. If both `data` and `shape` are given, the have to agree on the spatial shape.

class shetar.expansions.**SphericalSurfaceExpansion**(*order=None*, *data=None*, *wavenumber=None*, *shape=None*)

Holds spherical harmonics expansions.

Expansion corresponding to `SphericalHarmonics` basis functions. These are defined on the surface of a sphere, and can be used to expand arbitrary functions.

See *Expansion* for parameters and more methods.

class shetar.expansions.**InteriorExpansion**(*order=None*, *data=None*, *wavenumber=None*, *shape=None*)
Holds interior expansions.

Expansion corresponding to `RegularBase` basis functions. These expansions are well defined when all sources are outside the region of evaluation. This means that the evaluation radius is smaller than the distance to the closest source, e.g. an incident sound field.

See *Expansion* for parameters and more methods.

class shetar.expansions.**ExteriorExpansion**(*order=None*, *data=None*, *wavenumber=None*, *shape=None*)
Holds exterior expansions.

Expansion corresponding to `SingularBase` basis functions. These expansions are well defined when all sources are contained within the region closer to the origin than the evaluation domain. This means that the evaluation radius has to be larger than the distance to the closest source, e.g. a source at the origin.

See *Expansion* for parameters and more methods.

shetar.expansions.**plane_wave**(*order*, *strength=1*, *colatitude=None*, *azimuth=None*, *wavenumber=None*, *wavevector=None*)
Create an expansion of a plane wave.

A plane wave has the field

$$q \exp\left(i\vec{k} \cdot \vec{r}\right)$$

where $\vec{r}$ is the position in the field, and $\vec{k}$ is the wavevector.

The propagation direction of the plane wave can be specified using either a wavevector, or by colatitude, azimuth, and wavenumber.

Parameters

- **order** (*int*) – The highest order included in the expansion.
- **strength** (*numerical, default 1*) – The strength of the source, q in the above definition.
- **colatitude** (*None, optional*) – Colatitude propagation angle of the wave.
- **azimuth** (*None, optional*) – Azimuth propagation direction of the wave.
- **wavenumber** (*None, optional*) – Wavenumber of the propagating wave, k in the above definition.
- **wavevector** (*None, optional*) – Wavevector, i.e. both wavenumber and propagation direction for the wave. This will override the separate parameters if given.

Returns *InteriorExpansion* – The expansion of a plane wave is always an interior sound field.

shetar.expansions.**monopole**(*strength=1*, *order=0*, *wavenumber=1*, *position=None*, *domain='exterior'*)
Create a monopole source expansion.

A monopole has the field

$$\frac{q}{4\pi r} \exp\left(ikr\right)$$

where r is the distance between the source and the evaluation position.

Parameters

- **strength** (*numerical, default 1*) – Strength or the source, q in the above definition.

- **order** (*int, default 0*) – This can be used to return higher orders than needed for untranslated expansions, or to controll the output order of the translation.
- **wavenumber** (*numerical, default 1*) – The wavenumber of the source, k in the above definition.
- **position** (*None, optional*) – Position specifier for translation, see `coordinates` for more info. If given, the expansion will represent a monopole at this position.
- **domain** (*str, optional*) – Domain of translated expansions. Depending on where the resulting expansion should be evaluated, the target domain should be specified. If `domain='exterior'` (default), an exterior expansion will be returned, i.e. an expansion which is valid further away from the origin than the source location. Thus, the expansion should be evaluated using the singular bases, and in the exterior domain. If `domain='interior'`, an interior expansion will be returned, i.e. an expansion which is valid closer to from the origin than the source location. Thus, the expansion should be evaluated using the regular bases, and in the interior domain. A monopole at the origin (`position=None`) is always an exterior expansion.

Returns *ExteriorExpansion* or *InteriorExpansion* – See the `domain` documentation.

shetar.expansions.**circular_ring**(*order*, *radius*, *strength=1*, *colatitude=None*, *azimuth=None*, *wavenumber=None*, *wavevector=None*, *position=None*, *domain='exterior'*, *source_order=None*)

Create a circular ring source expansion.

The returns the expansion coefficients corresponding to a circular ring. In the far-field, this has the directivity

$$\frac{q}{4\pi} J_0(ka \sin\theta) \exp ikr$$

where J_0 is the zeroth order Bessel function, r is the distance from the source, and θ is the angle between the position vector and the source normal.

Parameters

- **order** (*int*) – The output order for the source.
- **radius** (*float*) – The effective radius of the source, a in the above definition.
- **strength** (*numerical, default 1*) – Strength of the source, q in the above definition.
- **colatitude** (*None, optional*) – Colatitude angle of the normal of the source.
- **azimuth** (*None, optional*) – Azimuth angle of the normal of the source.
- **wavenumber** (*float, optional*) – Wavenumber if the sound field, k in the above definition.
- **wavevector** (*None, optional*) – Wavevector, i.e. both wavenumber and propagation direction for the wave. This will override the separate parameters if given.
- **position** (*None, optional*) – Position specifier for translation, see `coordinates` for more info. If given, the expansion will represent a source at this position.
- **domain** (*str, optional*) – Domain of translated expansions. Depending on where the resulting expansion should be evaluated, the target domain should be specified. If `domain='exterior'` (default), an exterior expansion will be returned, i.e. an expansion which is valid further away from the origin than the source location. Thus, the expansion should be evaluated using the singular bases, and in the exterior domain. If `domain='interior'`, an interior expansion will be returned, i.e. an expansion which is valid closer to from the origin than the source location. Thus, the expansion should be evaluated using the regular bases, and in the interior domain. A monopole at the origin (`position=None`) is always an exterior expansion.

- **source_order** (*int, optional*) – Only used for translated expansions. The source order argument can be used in combination with the position argument to use a different expansion order for the initial source expansion before the translation is applied.

Returns *ExteriorExpansion* or *InteriorExpansion* – See the `domain` documentation.

CHAPTER

THREE

TRANSFORMS

Spatial transforms for expansions.

Spatial transforms are operations that work on expansions to express the same field but with a different coordinate system. The can be used in two ways: 1) To find expansion coefficients of a physical field after undergoing some physical transform, e.g. by moving a source. 2) To find the expansion coefficients of the same physical field but in a different mathematical coordinate system, e.g. as measured at a different point in space. These wto operations are closely linked, and are often the inverse of each other. This means that the expansion coefficients of an "input" expansion after undergoing a movement of the field by $\vec{x}$ are the same as the expansion coefficients of the same "input" field expanded at a new origin $-\vec{x}$. The default operation here is that of translating the field, e.g. a field created by a source at [1, 2, 3] translated by [4, 5, 6] will have the same expansion coefficients at the field from the source placed at [5, 7, 9].

The transforms are handled with a number of classes listed below. Rotations are either just a colatitude rotation, or a full rotation of the field. Translations are implemented along the z-axis only, for lower computational cost. General translations are handled as a sequence of rotation->coaxial translation->rotation. For full controll of an arbitrary transform, use the rotation and coaxial transslation classes instead of the translation classes.

A translation can be done either within a domain, or from the exterior domain to the interior domain. Take care to use the correct class to preserve the validity of the input expansion at the desired evaluation domain for the output expansion. There is often larger errors in the translations near the boundary of the region of validity. Take extra care if this region is of interest.

ColatitudeRotation	Organizes rotations for colatitude directions.
Rotation	Organizes arbitrary rotations.
CoaxialTranslation	Patent class for coaxial translations.
InteriorCoaxialTranslation	Handles translations in the interior domain.
ExteriorCoaxialTranslation	Handles translations in the exterior domain.
ExteriorInteriorCoaxialTranslation	Handles translations in the between domains.
Translation	Parent class for translations.
InteriorTranslation	Handles translations in the interior domain.
ExteriorTranslation	Handles translations in the exterior domain.
ExteriorInteriorTranslation	Handles translations in the between domains.

class shetar.transforms.**ColatitudeRotation**(*order*, *position=None*, *colatitude=None*, *defer_evaluation=False*)

Organizes rotations for colatitude directions.

Parameters

- **order** (*int*) – The highest order for the rotation. Rotations are performed at constant order, so the order of the input expansion and the output expansion are kept the same.
- **position** (*optional*) – Position specifier for the rotation, see `shetar.coordinates.`

Rotation.

- **colatitude** (*float, optional*) – Angle in radians by which to rotate.
- **defer_evaluation** (*bool, optional*) – Do not calculate the values upon initialization of the object.

class shetar.transforms.**Rotation**(*order*, *position=None*, *colatitude=None*, *azimuth=None*, *secondary_azimuth=None*, *new_z_axis=None*, *old_z_axis=None*, *defer_evaluation=False*)

Organizes arbitrary rotations.

This handles rotations with both a colatitude and two azimuth parts. For details on how the two azimuth angles interplay with the colatitude angle, refer to the examples.

See *ColatitudeRotation* and shetar.coordinates.Rotation for details on the parameters.

class shetar.transforms.**CoaxialTranslation**(*orders*, *position=None*, *radius=None*, *wavenumber=None*, *defer_evaluation=False*)

Patent class for coaxial translations.

This class should not be instantiated, only inherited from.

Parameters

- **orders** (*int or (int, int)*) – The orders between which the translation takes place. If given as a single value, it is used for both orders. If given as two values, a mixed order translation is evaluated.
- **position** – Position specifier, see shetar.coordinates.Translation.
- **radius** (*float*) – Distance to translate, see shetar.coordinates.Translation.
- **wavenumber** (*float*) – The wavenumber that the translation operates at.
- **defer_evaluation** (*bool, optional*) – Do not calculate the values upon initialization of the object.

class shetar.transforms.**InteriorCoaxialTranslation**(*orders*, *position=None*, *radius=None*, *wavenumber=None*, *defer_evaluation=False*)

Handles translations in the interior domain.

This applies translations from an interior translation to an interior translation.

See *CoaxialTranslation* and shetar.coordinates.Translation for details on parameters.

class shetar.transforms.**ExteriorCoaxialTranslation**(*orders*, *position=None*, *radius=None*, *wavenumber=None*, *defer_evaluation=False*)

Handles translations in the exterior domain.

This applies translations from an exterior translation to an exterior translation.

See *CoaxialTranslation* and shetar.coordinates.Translation for details on parameters.

class shetar.transforms.**ExteriorInteriorCoaxialTranslation**(*orders*, *position=None*, *radius=None*, *wavenumber=None*, *defer_evaluation=False*)

Handles translations in the between domains.

This applies translations from an exterior translation to an interior translation.

See *CoaxialTranslation* and shetar.coordinates.Translation for details on parameters.

class shetar.transforms.**Translation**(*orders*, *position=None*, *wavenumber=None*, *radius=None*, *colatitude=None*, *azimuth=None*, *defer_evaluation=False*)

Parent class for translations.

This class should not be instantiated, only inherited from.

See `shetar.coordinates.Translation` and *CoaxialTranslation* for details on parameters.

class shetar.transforms.**InteriorTranslation**(*orders*, *position=None*, *wavenumber=None*, *radius=None*, *colatitude=None*, *azimuth=None*, *defer_evaluation=False*)

Handles translations in the interior domain.

This applies translations from an interior translation to an interior translation.

See *CoaxialTranslation* and `shetar.coordinates.Translation` for details on parameters.

class shetar.transforms.**ExteriorTranslation**(*orders*, *position=None*, *wavenumber=None*, *radius=None*, *colatitude=None*, *azimuth=None*, *defer_evaluation=False*)

Handles translations in the exterior domain.

This applies translations from an exterior translation to an exterior translation.

See *CoaxialTranslation* and `shetar.coordinates.Translation` for details on parameters.

class shetar.transforms.**ExteriorInteriorTranslation**(*orders*, *position=None*, *wavenumber=None*, *radius=None*, *colatitude=None*, *azimuth=None*, *defer_evaluation=False*)

Handles translations in the between domains.

This applies translations from an exterior translation to an interior translation.

See *CoaxialTranslation* and `shetar.coordinates.Translation` for details on parameters.

CHAPTER

FOUR

EXAMPLES

4.1 Simple Rotations of Surface Expansion

This example show how a rotation can be applied to an expansion. We apply the rotation to a spherical surface expansion, but the same technique can be used for any type of spherical expansion.

[1]:
```
import numpy as np
import shetar
import plotly.graph_objects as go
```

[2]:
```
def show_axes(x, y, z, **kwargs):
    return go.Scatter3d(
        x=[0, x[0], 0, y[0], 0, z[0]],
        y=[0, x[1], 0, y[1], 0, z[1]],
        z=[0, x[2], 0, y[2], 0, z[2]],
        hovertext=['', 'x', '', 'y', '', 'z'],
        **kwargs
    )


def show_point(x, y, z, **kwargs):
    return go.Scatter3d(x=[0, x], y=[0, y], z=[0, z], **kwargs)

surf_layout = dict(showscale=False, colorscale='Spectral', cmid=0)
fig_layout = {
    'showlegend': False,
    'scene': {
        'xaxis': {'range': (-1.1, 1.1)},
        'yaxis': {'range': (-1.1, 1.1)},
        'zaxis': {'range': (-1.1, 1.1)},
        'aspectratio': {'x': 1, 'y': 1, 'z': 1},
        'camera': {'eye': {'x': 1, 'y': 1, 'z': 0.8}},
    }
}
```

We define the rotation using three rotation angles, , , and . The field will be rotated first around the z-axis with , then around the y-axis by , and finally by around the z-axis again.

[3]:
```
beta, alpha, mu = np.deg2rad([30, -45, 60])
order = 12
```

(continues on next page)

(continued from previous page)

```
expansion = shetar.expansions.SphericalSurfaceExpansion(order)
beam_a = shetar.bases.SphericalHarmonics(order, colatitude=np.deg2rad(30), azimuth=np.
↪deg2rad(30))
beam_b = shetar.bases.SphericalHarmonics(order, colatitude=np.deg2rad(60), azimuth=np.
↪deg2rad(45))
for n in range(order+1):
    for m in range(-n, n+1):
        expansion[n, m] = beam_a[n, m].conj() - beam_b[n, m].conj()

coordinate_rotation = shetar.coordinates.Rotation(colatitude=beta, azimuth=alpha,␣
↪secondary_azimuth=mu)
first_step = expansion.rotate(colatitude=0, azimuth=0, secondary_azimuth=mu)
second_step = expansion.rotate(colatitude=beta, azimuth=0, secondary_azimuth=mu)
rotated_expansion = expansion.rotate(colatitude=beta, azimuth=alpha, secondary_
↪azimuth=mu)
```

We define two meshes of spherical surfaces. The first mesh is in the original coordinate system. The second mesh will be used to show how the rotation can be interpreted as a reexpansion. Note how the rotation matrix Q is used to transform coordinates between the two systems. As a first step, we can note that the original expansion evaluated at the original coordinates give the same values as the rotated expansion evaluated at the corresponding rotared coordinates.

[4]:
```
old_x = np.array([1, 0, 0])
old_y = np.array([0, 1, 0])
old_z = np.array([0, 0, 1])
# The coordinates of the old base in the new base
new_x = coordinate_rotation.apply(old_x).xyz
new_y = coordinate_rotation.apply(old_y).xyz
new_z = coordinate_rotation.apply(old_z).xyz
# The coordinates of the new base in the new new base
new_xi = np.array([1, 0, 0])
new_eta = np.array([0, 1, 0])
new_zeta = np.array([0, 0, 1])
# The coordinates of the new base in the old base
old_xi = coordinate_rotation.apply(new_xi, inverse=True).xyz
old_eta = coordinate_rotation.apply(new_eta, inverse=True).xyz
old_zeta = coordinate_rotation.apply(new_zeta, inverse=True).xyz

old_xyz = show_axes(old_x, old_y, old_z, name='Old axes', line=dict(color='blue'))
old_xietazeta = show_axes(old_xi, old_eta, old_zeta, name='New axes', line=dict(color=
↪'red'))
new_xyz = show_axes(new_x, new_y, new_z, name='Old axes', line=dict(color='blue'))
new_xietazeta = show_axes(new_xi, new_eta, new_zeta, name='New axes', line=dict(color=
↪'red'))

res = 5
old_theta = np.linspace(0, np.pi, 180 // res + 1)
old_phi = np.linspace(0, 2 * np.pi, 360 // res + 1)
old_mesh = shetar.coordinates.SpatialCoordinate.parse_args(radius=0.8, colatitude=old_
↪theta, azimuth=old_phi, automesh=True)
new_mesh = old_mesh.rotate(colatitude=beta, azimuth=alpha, secondary_azimuth=mu)
old_x_mesh, old_y_mesh, old_z_mesh = old_mesh.xyz
```

(continues on next page)

(continued from previous page)

```
new_x_mesh, new_y_mesh, new_z_mesh = new_mesh.xyz
print(np.allclose(expansion.evaluate(old_mesh), rotated_expansion.evaluate(new_mesh)))
```

```
True
```

4.1.1 Original field

The expansion is that of two order limited point sources on the sphere. Note the orientation of the two maxima in relateion to the original coordinate axes in blue, and the rotated axes in red.

[5]:
```
go.Figure([
    go.Surface(x=old_x_mesh, y=old_y_mesh, z=old_z_mesh, surfacecolor=expansion.
↪evaluate(old_mesh).real, **surf_layout),
    old_xyz, old_xietazeta
], fig_layout).show('svg')
```

4.1.2 First rotation

The field after rotating by around the z-axis.

[6]:
```
go.Figure([
    go.Surface(x=old_x_mesh, y=old_y_mesh, z=old_z_mesh, surfacecolor=first_step.
↪evaluate(old_mesh).real, **surf_layout),
    old_xyz
], fig_layout).show('svg')
```

4.1.3 Second rotation

The field after also rotating with around the y-axis.

[7]:
```
go.Figure([
    go.Surface(x=old_x_mesh, y=old_y_mesh, z=old_z_mesh, surfacecolor=second_step.
↪evaluate(old_mesh).real, **surf_layout),
    old_xyz
], fig_layout).show('svg')
```

4.1.4 Final rotated field

This is the field after rotating around the z-axis again, by .

[8]:
```
go.Figure([
    go.Surface(x=old_x_mesh, y=old_y_mesh, z=old_z_mesh, surfacecolor=rotated_expansion.
↪evaluate(old_mesh).real, **surf_layout),
    old_xyz
], fig_layout).show('svg')
```

4.1.5 Reexpansion

If we instead evaluate the rotated expansion in the rotared coordinates, we see a similar result. Note how the field is oriented compared to the original coordinate axes and the new coordinate axes both in this case and in the original unrotated case. We see that this rotation can be interpreted as reexpanding the field in the rotated coordinate system.

[9]:
```
go.Figure([
    go.Surface(x=new_x_mesh, y=new_y_mesh, z=new_z_mesh, surfacecolor=rotated_expansion.
↪evaluate(new_mesh).real, **surf_layout),
    new_xyz, new_xietazeta
], fig_layout).show('svg')
```

4.1.6 Reevaluated expansion

If we evaluate the original expansion in the new coordinates, we again see the original field. However, it is oriented with respect to the new coordinate axes instead of the old coordinate axes.

[10]:
```
go.Figure([
    go.Surface(x=new_x_mesh, y=new_y_mesh, z=new_z_mesh, surfacecolor=expansion.
↪evaluate(new_mesh).real, **surf_layout),
    new_xyz, new_xietazeta
], fig_layout).show('svg')
```

4.2 Simple Translation of a Monopole

This example show how to easy get the expansion of a translated monopole, as well as how the convention of the translation direction come into play. In this example all fields are exterior fields, i.e. the evaluation points are further away from the origin than the source expansion.

[1]:
```
import numpy as np
import shetar
import plotly.graph_objects as go
fig_layout = dict(width=800, height=800, xaxis={'title': 'x/'}, yaxis={'title': 'y/'})
heatmap_layout = dict(showscale=False, zmid=0, colorscale='Spectral', zsmooth='best')
```

We will define all coordinates in terms of the wavelength . The origial expansion is that of a monopole at the origin of the coordinate system. It will be translated to a new position, and the new expansion is at a higher order.

[2]:
```
output_order = 50
 = 1
translation = (5 *  , 3*, 0)

resolution =  / 20
x_range = (8 *  , 14 *  )
y_range = (-3 *  , 3 *  )
```

The monopole expansion can be created from the monopole generator function. We then find the expansion of the translated monopole by usig the `translate` method. Note that it is also possible to get expansions of translated monopoles directly from the monopole generator function.

[3]:
```
old_expansion = shetar.expansions.monopole(wavenumber=2*np.pi/)
new_expansion = old_expansion.translate(order=output_order, position=translation)
```

Here we create the evaluation mesh for the visualization, and evaluate the base funcitons on the mesh. We create two meshes, one with the original coordinates and one with coordinates which are translated with the same vector as the expansion.

[4]:
```
nx = int((max(x_range) - min(x_range)) / resolution)
ny = int((max(y_range) - min(y_range)) / resolution)
nx += (nx + 1) % 2
ny += (ny + 1) % 2
x = np.linspace(min(x_range), max(x_range), nx)
y = np.linspace(min(y_range), max(y_range), ny)
z = 0
, ,  = x + translation[0], y + translation[1], z + translation[2]

old_mesh = np.stack(np.meshgrid(x, y, z, indexing='ij'), axis=-1).squeeze()
new_mesh = np.stack(np.meshgrid(, , , indexing='ij'), axis=-1).squeeze()

old_bases = new_expansion.bases(old_mesh)
new_bases = new_expansion.bases(new_mesh)
```

4.2.1 Original Field

The original source expansion evaluated in the original coordinates.

[5]:
```
go.Figure([
    go.Heatmap(x=x, y=y, z=np.real(old_expansion.apply(old_bases)).T, **heatmap_layout),
], fig_layout).show('svg')
```

4.2.2 Translated field

The new expansion evaluated in the oridinal coordinates. Note how this describes a monopole source at the translation position $\vec{t}$. This could also be intrpreted as a reexpansion of the original field with a new origin at $-\vec{t}$, evaluated at the same coorinate values but in with the new origin.

[6]:
```
go.Figure([
    go.Heatmap(x=x, y=y, z=np.real(new_expansion.apply(old_bases)).T, **heatmap_layout),
], fig_layout).show('svg')
```

4.2.3 Reexpanded field

The new expansion evaluated at the shifted coordinates. Note how the values are the same as the original expansion evaluated at the original coordinates. This could also be seen as a reexpansion of the original field with a new origin at $-\vec{t}$, evaluated at the coordinates in the new origin which correspond to the same points in space at the old coordinates.

[7]:
```
go.Figure([
    go.Heatmap(x=, y=, z=np.real(new_expansion.apply(new_bases)).T, **heatmap_layout),
], fig_layout).show('svg')
```

4.2.4 Reevalulated field

The old expansion evaluated at the shifted coordinates. Note how this still represents a monopole at the origin of the system.

[8]:
```
go.Figure([
    go.Heatmap(x=, y=, z=np.real(old_expansion.apply(new_bases)).T, **heatmap_layout),
], fig_layout).show('svg')
```

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

shetar.bases, 3
shetar.expansions, 7
shetar.transforms, 11

INDEX

A

AssociatedLegendrePolynomials (*class in shetar.bases*), 3

C

circular_ring() (*in module shetar.expansions*), 9
CoaxialTranslation (*class in shetar.transforms*), 12
ColatitudeRotation (*class in shetar.transforms*), 11

D

DualBase (*class in shetar.bases*), 5
DualRadialBase (*class in shetar.bases*), 4

E

Expansion (*class in shetar.expansions*), 7
ExteriorCoaxialTranslation (*class in shetar.transforms*), 12
ExteriorExpansion (*class in shetar.expansions*), 8
ExteriorInteriorCoaxialTranslation (*class in shetar.transforms*), 12
ExteriorInteriorTranslation (*class in shetar.transforms*), 13
ExteriorTranslation (*class in shetar.transforms*), 13

I

InteriorCoaxialTranslation (*class in shetar.transforms*), 12
InteriorExpansion (*class in shetar.expansions*), 8
InteriorTranslation (*class in shetar.transforms*), 13

L

LegendrePolynomials (*class in shetar.bases*), 3

M

module
    shetar.bases, 3
    shetar.expansions, 7
    shetar.transforms, 11
monopole() (*in module shetar.expansions*), 8
MultipoleBase (*class in shetar.bases*), 4

P

plane_wave() (*in module shetar.expansions*), 8

R

RadialBaseClass (*class in shetar.bases*), 3
RegularBase (*class in shetar.bases*), 5
RegularRadialBase (*class in shetar.bases*), 4
Rotation (*class in shetar.transforms*), 12

S

shetar.bases
    module, 3
shetar.expansions
    module, 7
shetar.transforms
    module, 11
SingularBase (*class in shetar.bases*), 5
SingularRadialBase (*class in shetar.bases*), 4
SphericalHarmonics (*class in shetar.bases*), 4
SphericalSurfaceExpansion (*class in shetar.expansions*), 7

T

Translation (*class in shetar.transforms*), 12