dispersion

Aug 09, 2021

Contents

1 Getting Started **3**

2 Contents **5**

2.1 The Catalogue Class 5
2.2 The Material Class 8
2.3 The SpectralData Class 14
2.4 The Spectrum Class 24
2.5 Configuration 26
2.6 Modules 27
2.7 File Format 27
2.8 Change Log 29

Python Module Index **31**

Index **33**

> *In optics, the phenomenon that the refractive index depends upon the frequency is called the phenomenon of dispersion, because it is the basis of the fact that light is "dispersed" by a prism into a spectrum.*
>
> *Feynman Lectures in physics* [1]

The `dispersion` Python package provides a way of loading and evaluating files containing the dispersion of the refractive index of materials.

CHAPTER 1

Getting Started

Python is required to install and use the `dispersion` package. It is recommended to use a package manager such as pip to install the package.

```
> pip install dispersion
```

now we need to tell the package where you are going to store the material data files. To do this we run the script that comes with the package

```
> dispersion_setup
```

This script will ask you to type in the path to a folder where the database file structure will be installed. Secondly, you will be asked to name the database. Finally you will be asked if you would like to install the available modules. See *Modules* for more information.

Now that the database has been setup, we can start using the package. For examples and further documentation, see the related pages.

CHAPTER 2

Contents

2.1 The Catalogue Class

This page describes the basic usage of the `Catalogue` class. This class is used for interfacing with the catalogue file. The catalogue provides meta data on all of the materials is in the database. These materials can then be loaded as `Material` objects into Python using the catalogue.

```
from dispersion import Catalogue
cat = Catalogue()
```

The catalogue relies on a specific file system structure in order to find the files in the database. An example of the file structure is the following,

```
database_root/
    catalogue.csv
    UserData/
        userfile1.txt
        userfile2.yml
        ...
    Module1/
        Module1_subdir1/
            mat1.yml
            mat2.yml
        Module1_subdir2/
            anothermat.yml
        ...
    Module2/
    ...
```

The catalogue files sits at the root level of the database filesystem. The other folders present in the root directory are the installed modules. By default a module called UserData is installed for the user to keep their data. Other modules consist of literature material databases available for download. See modules for a list of supported modules.

2.1.1 Building the Catalogue

Whenever new files are added to the database file system, the catalogue needs to be rebuilt.

To build the catalogue you can use the script included in this package

```
> dispersion_catalogue_rebuild
```

or alternatively you can rebuild from inside python

```
from dispersion import Catalogue
cat = Catalogue(rebuild='All')
```

When rebuilding the catalogue, you can choose to rebuild either some or all of the modules.

2.1.2 Setting an Alias

Many materials have different data files associated with them. In order to uniquely identify materials in the catalogue, a unique alias can be assigned to each material. This alias will be used later to extract the material from the database. By default all materials do not have any alias defined.

In order to set an alias, the catalogue file needs to be edited. This can be done in three ways: Externally, using python or interactively using iPython with the `qgrid` extension.

The following shows how to edit an alias using python.

```
row = cat.database[(cat.database.Name == name)]
cat.register_alias(row, "alias")
```

This will not work if there are multiple materials in the catalogue with the same "Name" attribute. In order to set the alias in this case, either use the path to the file on the inside the module folder as the filter,

```
row = cat.database[(cat.database.Path == path/to/file.txt)]
cat.register_alias(row, "alias")
cat.save_to_file()
```

or use the interactive editing. For this the qgrid IPython extension needs to be installed. Note that after installing the package with a package manager, iPython extensions need to be installed using,

```
jupyter nbextension enable --py --sys-prefix widgetsnbextension
jupyter nbextension enable --py --sys-prefix qgrid
```

After installation, open the catalogue in an IPython like environment (such as a Jupyter notebook) and use,

```
cat.edit_interactive()
```

note that "Interactive" must be set to true in the package configuration file to use interactive editing. After the aliases have been set in the alias column for the appropriate materials, the catalogue must be saved via,

```
cat.save_interactive()
cat.save_to_file()
```

The following animation shows the process of interactively editing the catalogue.

2.1.3 Accessing the Catalogue

To get a material from the database using the catalogue, use:

```
mat = cat.get_material("<mat_alias>")
```

this returns a Material object. If the argument of get_material is a string, then it must refer to the alias of the material in the catalogue. If the argument is an integer, it refers to the row number in the catalogue.

2.1.4 Full API

class catalogue.**Catalogue**(*config=None*, *rebuild='None'*)

administers the set of material data files

this class is used to find and load data files from disk which describe spectrally resolved refractive index or permittivity data into Material objects.

Parameters

config: dict or None configuration data

rebuild: list or None which modules which should be rebuilt

base_path: str where the database file structure is stored

file_name: str name of the catalogue file

database: pandas.DataFrame the catalogue of material files

qgrid_widget: qgrid.widget iPython widget for interactive editing of the catalogue

make_grid: function qgrid function for refreshing the interactive interface

reference_spectrum: Spectrum catalogue will provide n/k values at the reference spectrum value

rii_loader: dict temporary dict used in constructing the refractive index info catalogue

Methods

build_catalogue(self, df, rebuild)	read specified modules and return a dataframe combining all modules
edit_interactive(self)	returns an editable qgrid instance for interactively viewing the catalogue.
get_database(self)	returns the pandas data frame
get_material(self, identifier)	get a material from the catalogue using its alias or row number
make_reference_spectrum(self, config)	make the spectrum with which every material in the catalogue will be evaluated
read_filmetrics_db(self, db_path)	read the file structure provided my filmetrics.com
read_ri_info_db(self, db_path)	read the file structure provided by the refractiveindex.info website
read_user_data_db(self, db_path)	read user data files
register_alias(self, row_id, alias)	create an alias for a material to easily access it from the catalogue

Continued on next page

Table 1 – continued from previous page

save_interactive(self)	saves changes made to the qgrid when interfactive edit mode has been used
save_to_file(self)	save the pandas dataframe to the root path of the catalogue file structure
set_database(self, database)	set the pandas data frame
view_interactive(self)	returns a read only qgrid instance for interactively viewing the catalogue

build_catalogue (*self*, *df*, *rebuild*)
read specified modules and return a dataframe combining all modules

edit_interactive (*self*)
returns an editable qgrid instance for interactively viewing the catalogue. Call the method save_interactive to save any changes made

get_database (*self*)
returns the pandas data frame

get_material (*self*, *identifier*)
get a material from the catalogue using its alias or row number

make_reference_spectrum (*self*, *config*)
make the spectrum with which every material in the catalogue will be evaluated

read_filmetrics_db (*self*, *db_path*)
read the file structure provided my filmetrics.com

read_ri_info_db (*self*, *db_path*)
read the file structure provided by the refractiveindex.info website

read_user_data_db (*self*, *db_path*)
read user data files

register_alias (*self*, *row_id*, *alias*)
create an alias for a material to easily access it from the catalogue

save_interactive (*self*)
saves changes made to the qgrid when interfactive edit mode has been used

save_to_file (*self*)
save the pandas dataframe to the root path of the catalogue file structure

set_database (*self*, *database*)
set the pandas data frame

view_interactive (*self*)
returns a read only qgrid instance for interactively viewing the catalogue

2.2 The Material Class

This class defines the optical parameters for a given material. The optical parameters are defined via the complex refractive index or equivalently the complex permittivity. As a shorthand, we will refer to the real part of the refractive index as n, the imaginary part as k and the complex refractive index as nk. Similarly, the shorthand for permittivity is eps_r, eps_i and eps

There are multiple ways to initalise this class. Typically the objects of this class will be generated using the catalogue.

It can also be initalised in the following ways,

- using the path to a file
- using a constant value for n, nk, eps_r or eps.
- using tabulated data for n, nk or eps
- using a predefined model

Apart from the file path, these different definitions can be mixed and matched. For example a model or tabulated data for n, while taking a constant value for k.

A material is full defined when either both the real and imaginary parts of the refractive index or permittivity are defined, or alternatively a complex value for either refractive index or permittivity is defined.

Once a material has been fully defined, the refractive index or permittivity can be evaluated on a given spectrum. The spectrum can be wavelength, frequency, angular frequency or energy. For more information see Spectrum

```
import numpy as np
from dispersion import Material, Spectrum
mat = Material(fixed_n = 1.5) # k will be set = 0
spm = Spectrum( np.arange(380, 750, 10 ),
                spectrum_type= 'Wavelength',
                unit = 'nanometer')
nk_values = mat.get_nk_data(spm)
eps = mat.get_permittivity(spm)
```

2.2.1 Interpolation

If tabulated data is used to initalise the MaterialData, it will be automatically interpolated. The order of interpolation can be set by passing interp_order as a keyword to the MaterialData constructor. The order must be an integer. The default value is 1 (linear interpolation).

2.2.2 Extrapolation

Data can be extrapolated outside of the range in which it is defined. This should be done with great care, as extrapolated values may not even be qualitatively correct. However in circumstances where the material dispersion is very low, it may be practical to extrapolate the close to the spectrum of known values. Note that since the real and imaginary parts are extrapolated separately they must be independent of one another. Therefore, the material must be defined via separate real and imaginary parts, rather than via a complex value.

Extrapolation is achieved as follows,

```
new_spectrum = Spectrum(800., spectrum_type='Wavelength', unit='nm')
mat.extrapolate(new_spectrum, spline_order=2)
```

Due to extrapolation using splines, the results can vary greatly depending on the spline order used. For this reason it is recommended to verify the results of extrapolation before using the results for further calculations.

2.2.3 Removing Absorption

Sometimes it is useful to remove the absorption by setting the imaginary part of material data to a constant zero value. This can be achieved by,

```
mat.remove_absorption()
```

Note that this is only valid for data which have independent real and imaginary parts.

2.2.4 Full API

holds the complex refactive index or permittivity data

material_data implements the Material class which can hold different representations of spectral data (e.g. refractive index or permittivity). The data is in the form of either a constant value, tabulated data or a model. These different representations can be combined e.g. model for n (real part of refractive index) and constant value for k (imaginary part of refractive index)

Functions

_check_table_shape validate that a numpy array has a given shape

Classes

Material processes and interfaces refractive index data.

class material.**Bruggeman**(*spectrum*, *material1*, *material2*, *filling_fraction*)

Methods

create_effective_data(self)	this must be implemented in a subclass
dataset_to_dict(self)	generate a file_data type dictionary from this object
extrapolate(self, new_spectrum[, spline_order])	extrpolates the material data
get_maximum_valid_range(self)	find maximum spectral range that spans real and imaginary data.
get_nk_data(self, spectrum[, spectrum_type, ...])	return complex refractive index for a given input spectrum.
get_permittivity(self, spectrum_values[, ...])	return complex permittivity for a given input spectrum.
get_sample_spectrum(self)	spectrum which covers the maximum valid range of the material data
plot_nk_data(self, **kwargs)	plots the real and imaginary part of the refractive index
plot_permittivity(self, **kwargs)	plots the real and imaginary part of the permittivity
print_comment(self)	print material comment
print_reference(self)	print material reference
remove_absorption(self)	sets loss (k or epsi) to constant zero value
utf8_to_ascii(string)	converts a string from utf8 to ascii

add_dtype_suffix	
collapse_datasets	
prepare_file_dict	

create_effective_data(*self*)
this must be implemented in a subclass

class material.**EffectiveMedium**(*spectrum*, *material1*, *material2*, *filling_fraction*)

Methods

create_effective_data(self)	this must be implemented in a subclass
dataset_to_dict(self)	generate a file_data type dictionary from this object
extrapolate(self, new_spectrum[, spline_order])	extrpolates the material data
get_maximum_valid_range(self)	find maximum spectral range that spans real and imaginary data.
get_nk_data(self, spectrum[, spectrum_type, ...])	return complex refractive index for a given input spectrum.
get_permittivity(self, spectrum_values[, ...])	return complex permittivity for a given input spectrum.
get_sample_spectrum(self)	spectrum which covers the maximum valid range of the material data
plot_nk_data(self, **kwargs)	plots the real and imaginary part of the refractive index
plot_permittivity(self, **kwargs)	plots the real and imaginary part of the permittivity
print_comment(self)	print material comment
print_reference(self)	print material reference
remove_absorption(self)	sets loss (k or epsi) to constant zero value
utf8_to_ascii(string)	converts a string from utf8 to ascii

add_dtype_suffix	
collapse_datasets	
prepare_file_dict	

create_effective_data(*self*)
this must be implemented in a subclass

class material.**Material**(***kwargs*)
Class for processing refractive index and permittivity data

Parameters

file_path: str file path from which to load data

fixed_n: float fixed real part of refractive index

fixed_nk: complex fixed complex refractive index

fixed_eps_r: float fixed real part of permittivity

fixed_eps: complex fixed complex permittivity

tabulated_n: Nx2 array table of real part of refractive index to interpolate

tabulated_nk: Nx3 array table of real and imaginary refractive index values to interpolate

tabulated_eps: Nx3 array table of real and imaginary permittivity values to interpolate

model_kw: dict model parameters

spectrum_type: str sets the default spectrum type

unit: str sets the default unit

meta_data: dict contains the meta data for the material

data: dict holds one or two SpectralData objects to describe the data

options: dict holds options for the material object

defaults: dict default values for spectrum data

Warning: the parameters file_path, fixed_n, fixed_nk, fixed_eps_r, fixed_eps, tabulated_n, tabulated_nk, tabulated_eps and model_kw are mututally exclusive.

Methods

dataset_to_dict(self)	generate a file_data type dictionary from this object
extrapolate(self, new_spectrum[, spline_order])	extrpolates the material data
get_maximum_valid_range(self)	find maximum spectral range that spans real and imaginary data.
get_nk_data(self, spectrum[, spectrum_type, ...])	return complex refractive index for a given input spectrum.
get_permittivity(self, spectrum_values[, ...])	return complex permittivity for a given input spectrum.
get_sample_spectrum(self)	spectrum which covers the maximum valid range of the material data
plot_nk_data(self, **kwargs)	plots the real and imaginary part of the refractive index
plot_permittivity(self, **kwargs)	plots the real and imaginary part of the permittivity
print_comment(self)	print material comment
print_reference(self)	print material reference
remove_absorption(self)	sets loss (k or epsi) to constant zero value
utf8_to_ascii(string)	converts a string from utf8 to ascii

add_dtype_suffix	
collapse_datasets	
prepare_file_dict	

dataset_to_dict (*self*)
generate a file_data type dictionary from this object

Parameters

material_data: dict keys: name, real, imag, complex

Returns

dict a list of dicts that has a format suitable for writing to file

extrapolate (*self*, *new_spectrum*, *spline_order=2*)
extrpolates the material data

extrapolates the material data to cover the range defined by the spectrum new_spectrum. if new_spectrum has only one element, the data will be extrapolated from the relevant end of its valid range up to the value given by new_spectrum. spline_order defines the order of the spline used for extrapolation. The results of the extrapolation depend heavily on the order chosen, so please check the end result to make sure it make physical sense.

Parameters

new_spectrum: Spectrum the values to exrapolate to

spline_order: int the order of spline to use for interpolation -> extrpolation

Raises

NotImplementedError if the material is defined as via a complex value

get_maximum_valid_range (*self*)
find maximum spectral range that spans real and imaginary data.

Checks both real and imaginary parts of spectral data and finds the maximum spectral range which is valid for both parts.

Returns

2x1 np.array the maximum valid range

get_nk_data (*self*, *spectrum*, *spectrum_type='wavelength'*, *unit='meter'*)
return complex refractive index for a given input spectrum.

Parameters

spectrum: np.array or Spectrum the spectral values to evaluate

spectrum_type: str {'wavelength', 'frequency', 'energy'} type of spectrum

unit: str {'meter', 'nanometer', 'micrometer', 'hertz', 'electronvolt'} unit of spectrum (must match spectrum type)

Returns

np.complex128 the complex n/k values (if input spectrum has size == 1)

np.array with np.complex128 dtype the complex n/k values (if input spectrum has size > 1)

get_permittivity (*self*, *spectrum_values*, *spectrum_type='wavelength'*, *unit='meter'*)
return complex permittivity for a given input spectrum.

Parameters

spectrum: np.array or Spectrum the spectral values to evaluate

spectrum_type: str {'wavelength', 'frequency', 'energy'} type of spectrum

unit: str {'meter', 'nanometer', 'micrometer', 'hertz', 'electronvolt'} unit of spectrum (must match spectrum type)

Returns

np.complex128 the complex permittivity values (if input spectrum has size == 1)

np.array with np.complex128 dtype the complex permittivity values (if input spectrum has size > 1)

get_sample_spectrum (*self*)
spectrum which covers the maximum valid range of the material data

plot_nk_data (*self*, ***kwargs*)
plots the real and imaginary part of the refractive index

plot_permittivity (*self*, ***kwargs*)
plots the real and imaginary part of the permittivity

print_comment (*self*)
print material comment

print_reference (*self*)
print material reference

remove_absorption (*self*)
sets loss (k or epsi) to constant zero value

> **Warning:** has no effect if the material is defined as via complex data instead of separate real and imaginary parts.

static utf8_to_ascii (*string*)
converts a string from utf8 to ascii

class material.**MaxwellGarnett** (*spectrum*, *material1*, *material2*, *filling_fraction*)

Methods

create_effective_data(self)	this must be implemented in a subclass
dataset_to_dict(self)	generate a file_data type dictionary from this object
extrapolate(self, new_spectrum[, spline_order])	extrpolates the material data
get_maximum_valid_range(self)	find maximum spectral range that spans real and imaginary data.
get_nk_data(self, spectrum[, spectrum_type, ...])	return complex refractive index for a given input spectrum.
get_permittivity(self, spectrum_values[, ...])	return complex permittivity for a given input spectrum.
get_sample_spectrum(self)	spectrum which covers the maximum valid range of the material data
plot_nk_data(self, **kwargs)	plots the real and imaginary part of the refractive index
plot_permittivity(self, **kwargs)	plots the real and imaginary part of the permittivity
print_comment(self)	print material comment
print_reference(self)	print material reference
remove_absorption(self)	sets loss (k or epsi) to constant zero value
utf8_to_ascii(string)	converts a string from utf8 to ascii

add_dtype_suffix	
collapse_datasets	
prepare_file_dict	

create_effective_data (*self*)
this must be implemented in a subclass

2.3 The SpectralData Class

Spectral data is use to define data with an associated spectrum. Typically objects of this class will be generated by a parent MaterialData class which handles one or more spectral data classes in order to provide both real and imaginary parts of optical parameters.

2.3.1 Full API

describes spectrally dependent data

spectral_data implements the abstract base class SpectralData which defines a material parameter which has a spectral dependence (e.g. refractive index, permittivity). Each of the subclasses must implement the evaluate method which returns the material parameter for a given Spectrum object.

Classes

SpectralData abstract base class

Constant: SpectralData for values that are independent of the spectrum

Interpolation: SpectralData for tabulated data values

Model: SpectralData abstract base class for values generated from a particular model

Sellmeier: Model implements the Sellmeier model for refractive index

Sellmeier2: Model implements the modified Sellmeier model for refractive index

Polynomial: Model implements a polynomial model for refractive index

RefractiveIndexInfo: Model implements the RefractiveIndexInfo model for refractive index

Cauchy: Model implements the Cauchy model for refractive index

Gases: Model implements the Gas model for refractive index

Herzberger: Model implements the Herzberger model for refractive index

Retro: Model implements the Retro model for refractive index

Exotic: Model implements the Exotic model for refractive index

Drude: Model implements the Drude model for complex permittivity

DrudeLorentz: Model implements the Drude-Lorentz model for complex permittivity

TaucLorentz: Model implements the Tauc-Lorentz model for complex permittivity

Notes

for more information on models see https://refractiveindex.info/about

class spectral_data.**Cauchy**(*model_parameters*, *valid_range*, *spectrum_type='wavelength'*, *unit='m'*)

requires wavelength input in micrometers returns real part of refractive index only

Methods

dict_repr(self)	return a dictionary representation of the object
evaluate(self, spectrum)	returns the value of the spectral data for the given spectrum
input_output(self)	defines the required inputs and the output spectrum type

Continued on next page

Table 6 – continued from previous page

preprocess(self, spectrum)	check range of spectrum, convert to correct sType and unit and return an object with the same tensor order (scalar|vector) with values set to 1.0
suggest_spectrum(self)	for plotting the spectral data we take a geometrically spaced set of values

validate_spectrum_type

evaluate(*self*, *spectrum*)
returns the value of the spectral data for the given spectrum

class spectral_data.**Constant**(*constant*, *valid_range=(0, inf)*, *spectrum_type='wavelength'*, *unit='m'*)
for spectral data values that are independent of the spectrum

Methods

dict_repr(self)	return a dictionary representation of the object
evaluate(self, spectrum)	returns the value of the spectral data for the given spectrum
suggest_spectrum(self)	for plotting the spectral data we take a geometrically spaced set of values

dict_repr(*self*)
return a dictionary representation of the object

evaluate(*self*, *spectrum*)
returns the value of the spectral data for the given spectrum

class spectral_data.**Drude**(*model_parameters*, *valid_range*, *spectrum_type='wavelength'*, *unit='m'*)
requires energy input in eV returns real and imaginary parts of permittivity

Methods

dict_repr(self)	return a dictionary representation of the object
evaluate(self, spectrum)	returns the value of the spectral data for the given spectrum
input_output(self)	defines the required inputs and the output spectrum type
preprocess(self, spectrum)	check range of spectrum, convert to correct sType and unit and return an object with the same tensor order (scalar|vector) with values set to 1.0
suggest_spectrum(self)	for plotting the spectral data we take a geometrically spaced set of values

validate_spectrum_type

evaluate(*self*, *spectrum*)

returns the value of the spectral data for the given spectrum

class spectral_data.**DrudeLorentz**(*model_parameters*, *valid_range*, *spectrum_type='wavelength'*, *unit='m'*)

requires energy input in eV returns real and imaginary parts of permittivity

Methods

dict_repr(self)	return a dictionary representation of the object
evaluate(self, spectrum)	returns the value of the spectral data for the given spectrum
input_output(self)	defines the required inputs and the output spectrum type
preprocess(self, spectrum)	check range of spectrum, convert to correct sType and unit and return an object with the same tensor order (scalar|vector) with values set to 1.0
suggest_spectrum(self)	for plotting the spectral data we take a geometrically spaced set of values

validate_spectrum_type	

evaluate(*self*, *spectrum*)

returns the value of the spectral data for the given spectrum

class spectral_data.**Exotic**(*model_parameters*, *valid_range*, *spectrum_type='wavelength'*, *unit='m'*)

requires wavelength input in micrometers returns real part of refractive index only

Methods

dict_repr(self)	return a dictionary representation of the object
evaluate(self, spectrum)	returns the value of the spectral data for the given spectrum
input_output(self)	defines the required inputs and the output spectrum type
preprocess(self, spectrum)	check range of spectrum, convert to correct sType and unit and return an object with the same tensor order (scalar|vector) with values set to 1.0
suggest_spectrum(self)	for plotting the spectral data we take a geometrically spaced set of values

validate_spectrum_type	

evaluate(*self*, *spectrum*)

returns the value of the spectral data for the given spectrum

class spectral_data.**Extrapolation**(*spectral_data*, *extended_spectrum*, *spline_order=2*)

for extending spectral data outside of the valid range. Use with caution

Methods

evaluate(self, spectrum)	returns the value of the spectral data for the fiven spectrum
extrapolate_data(self)	makes a spline base on the base data for future lookup
get_extrap_spectrum(self, extended_spectrum)	takes a Spectrum object with one or two values possibly lying outside the base spectral range.
suggest_spectrum(self)	for plotting the spectral data we take a geometrically spaced set of values
validate_extrap_val(self, extrap_val, base_range)	checks if extrap_val lies outside base_range and replaces the relevant value in base_range with extrap_val.

evaluate(*self*, *spectrum*)
returns the value of the spectral data for the fiven spectrum

extrapolate_data(*self*)
makes a spline base on the base data for future lookup

get_extrap_spectrum(*self*, *extended_spectrum*)
takes a Spectrum object with one or two values possibly lying outside the base spectral range. Raises an error if the values do not lie outside the base spectral range. returns a new length two spectrum that gives the lower and upper bound for an extrapolation

validate_extrap_val(*self*, *extrap_val*, *base_range*)
checks if extrap_val lies outside base_range and replaces the relevant value in base_range with extrap_val.

class spectral_data.**Fano**(*model_parameters*, *valid_range*, *spectrum_type='wavelength'*, *unit='m'*)
this model can be applied to scattering cross sections requires energy input in eV returns real and imaginary parts of scattering cross section

Methods

dict_repr(self)	return a dictionary representation of the object
evaluate(self, spectrum)	returns the value of the spectral data for the given spectrum
input_output(self)	defines the required inputs and the output spectrum type
preprocess(self, spectrum)	check range of spectrum, convert to correct sType and unit and return an object with the same tensor order (scalar|vector) with values set to 1.0
suggest_spectrum(self)	for plotting the spectral data we take a geometrically spaced set of values

validate_spectrum_type	

evaluate(*self*, *spectrum*)
returns the value of the spectral data for the given spectrum

input_output(*self*)
defines the required inputs and the output spectrum type

class spectral_data.**Gases**(*model_parameters*, *valid_range*, *spectrum_type='wavelength'*, *unit='m'*)

requires wavelength input in micrometers returns real part of refractive index only

Methods

dict_repr(self)	return a dictionary representation of the object
evaluate(self, spectrum)	returns the value of the spectral data for the given spectrum
input_output(self)	defines the required inputs and the output spectrum type
preprocess(self, spectrum)	check range of spectrum, convert to correct sType and unit and return an object with the same tensor order (scalar|vector) with values set to 1.0
suggest_spectrum(self)	for plotting the spectral data we take a geometrically spaced set of values

validate_spectrum_type	

evaluate(*self*, *spectrum*)

returns the value of the spectral data for the given spectrum

class spectral_data.**Herzberger**(*model_parameters*, *valid_range*, *spectrum_type='wavelength'*, *unit='m'*)

requires wavelength input in micrometers returns real part of refractive index only

Methods

dict_repr(self)	return a dictionary representation of the object
evaluate(self, spectrum)	returns the value of the spectral data for the given spectrum
input_output(self)	defines the required inputs and the output spectrum type
preprocess(self, spectrum)	check range of spectrum, convert to correct sType and unit and return an object with the same tensor order (scalar|vector) with values set to 1.0
suggest_spectrum(self)	for plotting the spectral data we take a geometrically spaced set of values

validate_spectrum_type	

evaluate(*self*, *spectrum*)

returns the value of the spectral data for the given spectrum

class spectral_data.**Interpolation**(*data*, *spectrum_type='wavelength'*, *unit='m'*, *interp_order=1*)

for spectral data values that are from tabulated data

Methods

dict_repr(self)	return a dictionary representation of the object
evaluate(self, spectrum)	returns the value of the spectral data for the fiven spectrum
interpolate_data(self)	interpolates the data for future lookup
suggest_spectrum(self)	for plotting the spectral data we take a geometrically spaced set of values

dict_repr (*self*)
return a dictionary representation of the object

evaluate (*self*, *spectrum*)
returns the value of the spectral data for the fiven spectrum

interpolate_data (*self*)
interpolates the data for future lookup

class spectral_data.**Model** (*model_parameters*, *valid_range*, *spectrum_type='wavelength'*, *unit='m'*)
for spectral data values depending on model parameters

Methods

dict_repr(self)	return a dictionary representation of the object
evaluate(self, spectrum)	returns the value of the spectral data for the given spectrum
input_output(self)	defines the required inputs and the output spectrum type
preprocess(self, spectrum)	check range of spectrum, convert to correct sType and unit and return an object with the same tensor order (scalar|vector) with values set to 1.0
suggest_spectrum(self)	for plotting the spectral data we take a geometrically spaced set of values

validate_spectrum_type	

dict_repr (*self*)
return a dictionary representation of the object

evaluate (*self*, *spectrum*)
returns the value of the spectral data for the given spectrum

input_output (*self*)
defines the required inputs and the output spectrum type

preprocess (*self*, *spectrum*)
check range of spectrum, convert to correct sType and unit and return an object with the same tensor order (scalar|vector) with values set to 1.0

class spectral_data.**Polynomial** (*model_parameters*, *valid_range*, *spectrum_type='wavelength'*, *unit='m'*)
requires wavelength input in micrometers returns real part of refractive index only

Methods

dict_repr(self)	return a dictionary representation of the object
evaluate(self, spectrum)	returns the value of the spectral data for the given spectrum
input_output(self)	defines the required inputs and the output spectrum type
preprocess(self, spectrum)	check range of spectrum, convert to correct sType and unit and return an object with the same tensor order (scalar|vector) with values set to 1.0
suggest_spectrum(self)	for plotting the spectral data we take a geometrically spaced set of values

validate_spectrum_type	

evaluate(*self*, *spectrum*)
returns the value of the spectral data for the given spectrum

class spectral_data.**RefractiveIndexInfo**(*model_parameters*, *valid_range*, *spectrum_type='wavelength'*, *unit='m'*)
requires wavelength input in micrometers returns real part of refractive index only

Methods

dict_repr(self)	return a dictionary representation of the object
evaluate(self, spectrum)	returns the value of the spectral data for the given spectrum
input_output(self)	defines the required inputs and the output spectrum type
preprocess(self, spectrum)	check range of spectrum, convert to correct sType and unit and return an object with the same tensor order (scalar|vector) with values set to 1.0
suggest_spectrum(self)	for plotting the spectral data we take a geometrically spaced set of values

validate_spectrum_type	

evaluate(*self*, *spectrum*)
returns the value of the spectral data for the given spectrum

class spectral_data.**Retro**(*model_parameters*, *valid_range*, *spectrum_type='wavelength'*, *unit='m'*)
requires wavelength input in micrometers returns real part of refractive index only

Methods

dict_repr(self)	return a dictionary representation of the object

Continued on next page

Table 19 – continued from previous page

evaluate(self, spectrum)	returns the value of the spectral data for the given spectrum
input_output(self)	defines the required inputs and the output spectrum type
preprocess(self, spectrum)	check range of spectrum, convert to correct sType and unit and return an object with the same tensor order (scalar|vector) with values set to 1.0
suggest_spectrum(self)	for plotting the spectral data we take a geometrically spaced set of values

validate_spectrum_type

evaluate(*self*, *spectrum*)
returns the value of the spectral data for the given spectrum

class spectral_data.**Sellmeier**(*model_parameters*, *valid_range*, *spectrum_type='wavelength'*, *unit='m'*)
requires wavelength input in micrometers returns real part of refractive index only

Methods

dict_repr(self)	return a dictionary representation of the object
evaluate(self, spectrum)	returns the value of the spectral data for the given spectrum
input_output(self)	defines the required inputs and the output spectrum type
preprocess(self, spectrum)	check range of spectrum, convert to correct sType and unit and return an object with the same tensor order (scalar|vector) with values set to 1.0
suggest_spectrum(self)	for plotting the spectral data we take a geometrically spaced set of values

validate_spectrum_type

evaluate(*self*, *spectrum*)
returns the value of the spectral data for the given spectrum

class spectral_data.**Sellmeier2**(*model_parameters*, *valid_range*, *spectrum_type='wavelength'*, *unit='m'*)
requires wavelength input in micrometers returns real part of refractive index only

Methods

dict_repr(self)	return a dictionary representation of the object
evaluate(self, spectrum)	returns the value of the spectral data for the given spectrum
input_output(self)	defines the required inputs and the output spectrum type

Continued on next page

Table 21 – continued from previous page

preprocess(self, spectrum)	check range of spectrum, convert to correct sType and unit and return an object with the same tensor order (scalar|vector) with values set to 1.0
suggest_spectrum(self)	for plotting the spectral data we take a geometrically spaced set of values

validate_spectrum_type	

evaluate(*self*, *spectrum*)
returns the value of the spectral data for the given spectrum

class spectral_data.**SpectralData**(*valid_range*, *spectrum_type='wavelength'*, *unit='nm'*)
Base class for defining a quantity (e.g. refactive index) which is defined over a given spectrum (see class Spectrum).

Methods

evaluate(self, spectrum)	returns the value of the spectral data for the given spectrum
suggest_spectrum(self)	for plotting the spectral data we take a geometrically spaced set of values

evaluate(*self*, *spectrum*)
returns the value of the spectral data for the given spectrum

suggest_spectrum(*self*)
for plotting the spectral data we take a geometrically spaced set of values

class spectral_data.**TaucLorentz**(*model_parameters*, *valid_range*, *spectrum_type='wavelength'*, *unit='m'*)
requires energy input in eV returns real and imaginary parts of permittivity

Methods

dict_repr(self)	return a dictionary representation of the object
evaluate(self, spectrum)	returns the value of the spectral data for the given spectrum
input_output(self)	defines the required inputs and the output spectrum type
preprocess(self, spectrum)	check range of spectrum, convert to correct sType and unit and return an object with the same tensor order (scalar|vector) with values set to 1.0
suggest_spectrum(self)	for plotting the spectral data we take a geometrically spaced set of values

validate_spectrum_type	

evaluate(*self*, *spectrum*)
returns the value of the spectral data for the given spectrum

2.4 The Spectrum Class

Spectrum objects are used link values in an array to a specific physical quantity and associated units. Conversion between different quantities and units is automated via this class.

2.4.1 spectrum_type

- wavelength
- frequency
- energy
- angularfrequency
- wavenumber

2.4.2 unit

- meter [wavelength]
- micrometer [wavelength]
- nanometer [wavelength]
- hertz [frequency]
- electronvolt [energy]
- 1/s [angular frequency]
- 1/cm [wavenumber]

2.4.3 Full API

spectrum is an array of data which defines a physical spectrum

A Spectrum is a numpy array with an associated type which describes the physical quantity (e.g. wavelength, energy, etc.) and a unit for specifiying the SI unit of the spectrum (e.g. meter, electronVolt)

class spectrum.**Spectrum**(*values*, *spectrum_type='wavelength'*, *unit='m'*)
Class for converting between wavelength/frequency/energy when defining spectral quantities

Methods

check_type(type_to_check, is_type)	test if two spectrum_type strings are equal taking aliases into account
check_unit(unit, is_unit)	test if two unit strings are equal taking aliases into account
contains(self, other)	detects if the values of another Spectrum object are completely contained within the range of the values defined in this Spectrum object
convert_from(spectrum_type, unit, values)	converts units from given type into wavelength/meter representation

Continued on next page

Table 24 – continued from previous page

convert_to(self, spectrum_type, unit[, in_place])	converts units from wavelength/meter representation to given type/unit
display_string(unit)	gets the SI standard form of a unit string for printing
get_type_unit_string(self)	formats the unit and spectrum_type for display as an axis label
standardise_unit(spectrum_type, unit)	takes a spectrum_type and unit string and returns them in a standardised form
standardise_values(self, values)	check if we need to convert the value in order to obtain the wavelength/meter representation

static check_type(*type_to_check*, *is_type*)
test if two spectrum_type strings are equal taking aliases into account

static check_unit(*unit*, *is_unit*)
test if two unit strings are equal taking aliases into account

contains(*self*, *other*)
detects if the values of another Spectrum object are completely contained within the range of the values defined in this Spectrum object

static convert_from(*spectrum_type*, *unit*, *values*)
converts units from given type into wavelength/meter representation

convert_to(*self*, *spectrum_type*, *unit*, *in_place=False*)
converts units from wavelength/meter representation to given type/unit

static display_string(*unit*)
gets the SI standard form of a unit string for printing

get_type_unit_string(*self*)
formats the unit and spectrum_type for display as an axis label

static standardise_unit(*spectrum_type*, *unit*)
takes a spectrum_type and unit string and returns them in a standardised form

standardise_values(*self*, *values*)
check if we need to convert the value in order to obtain the wavelength/meter representation

spectrum.**ang_freq_to_standard**(*unit*, *values*)
converts angular frequency values to standardised representation

spectrum.**energy_to_standard**(*unit*, *values*)
converts energy values to standardised representation

spectrum.**frequency_to_standard**(*unit*, *values*)
converts frequency values to standardised representation

spectrum.**safe_inverse**(*values*)
returns the inverse of a scalar or array while converting values of 0 to inf and values of inf to 0

spectrum.**safe_inverse_array**(*values*)
takes the inverse of an array while converting values of 0 to inf and values of inf to 0

spectrum.**safe_inverse_scalar**(*value*)
takes the inverse of a scalar while converting values of 0 to inf and values of inf to 0

spectrum.**to_ang_freq**(*unit*, *values*)
converts values from standard to angular frequency representation

spectrum.**to_energy**(*unit*, *values*)
converts values from standard to energy representation

`spectrum.`**`to_frequency`**(*unit*, *values*)
converts values from standard to frequency representation

`spectrum.`**`to_wavelength`**(*unit*, *values*)
converts values from standard to wavelength representation

`spectrum.`**`to_wavenumber`**(*unit*, *values*)
converts values from standard to wavenumber representation

`spectrum.`**`wavelength_to_standard`**(*unit*, *values*)
converts wavelength values to standardised representation

`spectrum.`**`wavenumber_to_standard`**(*unit*, *values*)
converts wavenumber values to standardised representation

2.5 Configuration

Some of the features of the package are configurable. These can be changed in configuration file. In order to use the `Catalogue` class, a valid configuration file (config.yaml) needs to be used. when setting up the package (see *Getting Started*) a new configuration file with default values will be created.

2.5.1 Location

The config file can exist in two different locations. The first location is a user specific directory, for linux systems this is,

```
~/.config/refractive_index_database
```

whereas on windows systems this is,

```
%LOCALAPPDATA%\refractive_index_database
```

this is also the default directory used when creating a new configuration if no config file exists. If no config file is found in this location, the package looks in the package directory.

2.5.2 Values

Path **Type**: *str*

The path to the root of the database file system

File **Type**: *str*

Name of the catalogue file (will be located in directory defined by Path)

Interactive **Type**: *bool*

Set to true for interactive editing of the database in IPython (requires `qgrid`)

Plotting **Type**: *bool*

Set to true for plotting of material data (requires `matplotlib`)

Modules **Type**: *dict*

modules to be included in the database

ReferenceSpectrum **Type**: *dict*

use Value, SpectrumType and Unit to define the spectral value at which the reference spectrum will be calculated when building the database

2.6 Modules

The `dispersion` package keeps different literature databases internally separate as so called Modules. This page lists the currently available modules.

2.6.1 UserData

The user data is the only module that is installed by default. It is empty apart from example file types. This is where the user can place their own files.

2.6.2 RefractiveIndexInfo

The refractive index database hosted at refractive index info is made freely available via their github project.

installing this module using the setup script requires the `gitlab` python package.

2.7 File Format

Files in the database can be in two different formats. Either as a text file (.txt, .csv or .nk) or as a YAML file (.yml). An example of each format is generated in the UserData module when installing the package.

2.7.1 Text File

This is an example of the text file format,

```
# This is a multiline meta-comment
# which provides information not
# in metadata
# REFERENCES: Literature reference to the data
# AUTHOR: The author of this data file
# FULLNAME: Full name of the material
# NAME: Short name of the material
# COMMENTS: Any additional information goes here
# SPECTRUMTYPE: wavelength
# UNIT: nanometer
# DATATYPE: tabulated nk
#
400.00000000  1.70000000      0.10000000
500.00000000  1.60000000      0.05000000
600.00000000  1.50000000      0.00000000
700.00000000  1.40000000      0.00000000
```

- The optional metacomment comes at the beginning and can span multiple lines, each line must begin with # for .txt and .csv or ; for .nk files.

- The optional metadata is written in key:value pairs, one per line, beginning with a # (.txt, .csv) or ; (.nk). For a list of valid metadata categories, see *MetaData*.
- Text files are restricted to datasets of tabulated data. For data defined via model parameters, use the YAML format.
- .txt and .nk files expect tab separated columns, .csv files should use comma separated columns.

2.7.2 YAML File

This is an example of the yaml file format,

```
# This is a multiline meta-comment
# which provides information not
# in metadata

REFERENCES: Literature reference to the data
COMMENTS: Any additional information goes here
NAME: Short name of the material
FULLNAME: Full name of the material
AUTHOR: The author of this data file
DATA:
  - DataType: model Sellmeier
    ValidRange: 0.35 2.
    SpectrumType: wavelength
    Unit: micrometer
    Yields: n
    Parameters: 0.    1.    0.05  2.    0.1  10.   25.
  - DataType: tabulated k
    ValidRange: 400. 600.
    SpectrumType: wavelength
    Unit: nm
    Data: |-
        4.e+02 1.e-01
        5.e+02 5.e-02
        6.e+02 0.e+00
```

- The optional metacomment comes at the beginning and can span multiple lines, each line must begin with #.
- The optional metadata is written in key:value pairs, one per line. For a list of valid metadata categories, see *MetaData*.
- The DATA section can contain one or more data sets. Each data set after DATA: begins with a "-".
- Each type of data set can contain different types of metadata.

2.7.3 MetaData

Metadata is divided into two kinds, metadata for the file, and metadata for each dataset.

File Metadata

REFERENCES Literature reference to the data

COMMENTS Any additional information

NAME Short name of the material

FULLNAME Full name of the material

AUTHOR The author of the data file

Dataset Metadata

DataType {tabulated_n, tabulated_k, tabulated_nk, tabulated_eps, model} For valid model names see *The SpectralData Class*

SpectrumType The type of spectrum. For more information see *The Spectrum Class*

Unit The physical unit of the spectrum. For more information see *The Spectrum Class*

ValidRange The spectral range that this data set covers

Data For tabulated data sets, the table of data.

Yields For model datasets, what values the model returns

Parameters For model datasets, the parameters or coefficients for the model

2.8 Change Log

- 1.0.0
  - first full release.
  - added config option to supress YAML package warnings.
  - fixed incorrect name of the setup script in the documentation.
  - added angstrom as a valid unit for wavelength spectrums.
  - added support for loading .nk files.
  - .txt and .csv files can now have colons in their metadata.
  - updated PyYAML dependency to v5.4
- 0.1.0.beta5
  - order of elements no longer reversed when inverting spectrum
  - Bruggeman effective medium model should provide physical solutions for non-absorbing media.
- 0.1.0.beta4
  - automatic github integration with pypi database.
- 0.1.0.beta3
  - automatic github integration with test pypi database.
- 0.1.0.beta2
  - numerous typos fixed
  - enabled plotting using the matplotlib package
  - when taking the inverse of a spectrum, the order of the data is now reversed
  - added qgrid installation information to documentation
  - added rounding to 9 decimal places when converting to energy
  - added the Tauc Lorentz Model for permittivity
  - added the EffectiveMedium class for effective mediums of multiple materials
  - MaxwellGarnett and Bruggeman provide the respective effective media

  - updated the DrudeLorentz model definition to include oscillator strength
  - when converting spectrum_type and unit using inplace=True, the spectrum_type and unit will not be converted as well as the values

- genindex
- modindex
- search

Python Module Index

m

material, 10

s

spectral_data, 15
spectrum, 24

Index

A

ang_freq_to_standard() (*in module spectrum*), 25

B

Bruggeman (*class in material*), 10
build_catalogue() (*catalogue.Catalogue method*), 8

C

Catalogue (*class in catalogue*), 7
Cauchy (*class in spectral_data*), 15
check_type() (*spectrum.Spectrum static method*), 25
check_unit() (*spectrum.Spectrum static method*), 25
Constant (*class in spectral_data*), 16
contains() (*spectrum.Spectrum method*), 25
convert_from() (*spectrum.Spectrum static method*), 25
convert_to() (*spectrum.Spectrum method*), 25
create_effective_data() (*material.Bruggeman method*), 10
create_effective_data() (*material.EffectiveMedium method*), 11
create_effective_data() (*material.MaxwellGarnett method*), 14

D

dataset_to_dict() (*material.Material method*), 12
dict_repr() (*spectral_data.Constant method*), 16
dict_repr() (*spectral_data.Interpolation method*), 20
dict_repr() (*spectral_data.Model method*), 20
display_string() (*spectrum.Spectrum static method*), 25
Drude (*class in spectral_data*), 16
DrudeLorentz (*class in spectral_data*), 17

E

edit_interactive() (*catalogue.Catalogue method*), 8
EffectiveMedium (*class in material*), 10
energy_to_standard() (*in module spectrum*), 25
evaluate() (*spectral_data.Cauchy method*), 16
evaluate() (*spectral_data.Constant method*), 16
evaluate() (*spectral_data.Drude method*), 16
evaluate() (*spectral_data.DrudeLorentz method*), 17
evaluate() (*spectral_data.Exotic method*), 17
evaluate() (*spectral_data.Extrapolation method*), 18
evaluate() (*spectral_data.Fano method*), 18
evaluate() (*spectral_data.Gases method*), 19
evaluate() (*spectral_data.Herzberger method*), 19
evaluate() (*spectral_data.Interpolation method*), 20
evaluate() (*spectral_data.Model method*), 20
evaluate() (*spectral_data.Polynomial method*), 21
evaluate() (*spectral_data.RefractiveIndexInfo method*), 21
evaluate() (*spectral_data.Retro method*), 22
evaluate() (*spectral_data.Sellmeier method*), 22
evaluate() (*spectral_data.Sellmeier2 method*), 23
evaluate() (*spectral_data.SpectralData method*), 23
evaluate() (*spectral_data.TaucLorentz method*), 23
Exotic (*class in spectral_data*), 17
extrapolate() (*material.Material method*), 12
extrapolate_data() (*spectral_data.Extrapolation method*), 18
Extrapolation (*class in spectral_data*), 17

F

Fano (*class in spectral_data*), 18
frequency_to_standard() (*in module spectrum*), 25

G

Gases (*class in spectral_data*), 18
get_database() (*catalogue.Catalogue method*), 8
get_extrap_spectrum() (*spectral_data.Extrapolation method*), 18
get_material() (*catalogue.Catalogue method*), 8
get_maximum_valid_range() (*material.Material method*), 13

get_nk_data() (*material.Material method*), 13
get_permittivity() (*material.Material method*), 13
get_sample_spectrum() (*material.Material method*), 13
get_type_unit_string() (*spectrum.Spectrum method*), 25

H

Herzberger (*class in spectral_data*), 19

I

input_output() (*spectral_data.Fano method*), 18
input_output() (*spectral_data.Model method*), 20
interpolate_data() (*spectral_data.Interpolation method*), 20
Interpolation (*class in spectral_data*), 19

M

make_reference_spectrum() (*catalogue.Catalogue method*), 8
Material (*class in material*), 11
material (*module*), 10
MaxwellGarnett (*class in material*), 14
Model (*class in spectral_data*), 20

P

plot_nk_data() (*material.Material method*), 13
plot_permittivity() (*material.Material method*), 13
Polynomial (*class in spectral_data*), 20
preprocess() (*spectral_data.Model method*), 20
print_comment() (*material.Material method*), 13
print_reference() (*material.Material method*), 13

R

read_filmetrics_db() (*catalogue.Catalogue method*), 8
read_ri_info_db() (*catalogue.Catalogue method*), 8
read_user_data_db() (*catalogue.Catalogue method*), 8
RefractiveIndexInfo (*class in spectral_data*), 21
register_alias() (*catalogue.Catalogue method*), 8
remove_absorption() (*material.Material method*), 14
Retro (*class in spectral_data*), 21

S

safe_inverse() (*in module spectrum*), 25
safe_inverse_array() (*in module spectrum*), 25
safe_inverse_scalar() (*in module spectrum*), 25
save_interactive() (*catalogue.Catalogue method*), 8
save_to_file() (*catalogue.Catalogue method*), 8
Sellmeier (*class in spectral_data*), 22
Sellmeier2 (*class in spectral_data*), 22
set_database() (*catalogue.Catalogue method*), 8
spectral_data (*module*), 15
SpectralData (*class in spectral_data*), 23
Spectrum (*class in spectrum*), 24
spectrum (*module*), 24
standardise_unit() (*spectrum.Spectrum static method*), 25
standardise_values() (*spectrum.Spectrum method*), 25
suggest_spectrum() (*spectral_data.SpectralData method*), 23

T

TaucLorentz (*class in spectral_data*), 23
to_ang_freq() (*in module spectrum*), 25
to_energy() (*in module spectrum*), 25
to_frequency() (*in module spectrum*), 26
to_wavelength() (*in module spectrum*), 26
to_wavenumber() (*in module spectrum*), 26

U

utf8_to_ascii() (*material.Material static method*), 14

V

validate_extrap_val() (*spectral_data.Extrapolation method*), 18
view_interactive() (*catalogue.Catalogue method*), 8

W

wavelength_to_standard() (*in module spectrum*), 26
wavenumber_to_standard() (*in module spectrum*), 26