BPPP Documentation

Release 1.0

MW

March 25, 2016

CONTENTS

CHAPTER

ONE

INTRODUCTION

This is something I want to say that is not in the docstring.

1.1 Installing

Installing is easy via pip:

```
pip install bpm
```

1.2 Basic Usage

Here is how to use it:

```
m = myClass("hello")
```

meshgrid(**xi*, ***kwargs*)

Return coordinate matrices from coordinate vectors.

Make N-D coordinate arrays for vectorized evaluations of N-D scalar/vector fields over N-D grids, given one-dimensional coordinate arrays x1, x2,..., xn.

Changed in version 1.9: 1-D and 0-D cases are allowed.

Parameters

- **x2, ..., xn** (*x1,*) – 1-D arrays representing the coordinates of a grid.
- **indexing** (*{'xy', 'ij'}, optional*) – Cartesian ('xy', default) or matrix ('ij') indexing of output. See Notes for more details.

  New in version 1.7.0.
- **sparse** (*bool, optional*) – If True a sparse grid is returned in order to conserve memory. Default is False.

  New in version 1.7.0.
- **copy** (*bool, optional*) – If False, a view into the original arrays are returned in order to conserve memory. Default is True. Please note that `sparse=False, copy=False` will likely return non-contiguous arrays. Furthermore, more than one element of a broadcast array may refer to a single memory location. If you need to write to the arrays, make copies first.

  New in version 1.7.0.

Returns **X1, X2,..., XN** – For vectors *x1*, *x2*,..., 'xn' with lengths `Ni=len(xi)` , return `(N1, N2, N3,...Nn)` shaped arrays if indexing='ij' or `(N2, N1, N3,...Nn)` shaped arrays if indexing='xy' with the elements of *xi* repeated to fill the matrix along the first dimension for *x1*, the second for *x2* and so on.

Return type ndarray

Notes

This function supports both indexing conventions through the indexing keyword argument. Giving the string 'ij' returns a meshgrid with matrix indexing, while 'xy' returns a meshgrid with Cartesian indexing. In the 2-D case with inputs of length M and N, the outputs are of shape (N, M) for 'xy' indexing and (M, N) for 'ij' indexing. In the 3-D case with inputs of length M, N and P, outputs are of shape (N, M, P) for 'xy' indexing and (M, N, P) for 'ij' indexing. The difference is illustrated by the following code snippet:

```
xv, yv = meshgrid(x, y, sparse=False, indexing='ij')
for i in range(nx):
    for j in range(ny):
        # treat xv[i,j], yv[i,j]

xv, yv = meshgrid(x, y, sparse=False, indexing='xy')
for i in range(nx):
    for j in range(ny):
        # treat xv[j,i], yv[j,i]
```

In the 1-D and 0-D case, the indexing and sparse keywords have no effect.

See also:

`index_tricks.mgrid()` Construct a multi-dimensional "meshgrid" using indexing notation.

`index_tricks.ogrid()` Construct an open multi-dimensional "meshgrid" using indexing notation.

Examples

```
>>> nx, ny = (3, 2)
>>> x = np.linspace(0, 1, nx)
>>> y = np.linspace(0, 1, ny)
>>> xv, yv = meshgrid(x, y)
>>> xv
array([[ 0. ,  0.5,  1. ],
       [ 0. ,  0.5,  1. ]])
>>> yv
array([[ 0.,  0.,  0.],
       [ 1.,  1.,  1.]])
>>> xv, yv = meshgrid(x, y, sparse=True)  # make sparse output arrays
>>> xv
array([[ 0. ,  0.5,  1. ]])
>>> yv
array([[ 0.],
       [ 1.]])
```

meshgrid is very useful to evaluate functions on a grid.

```
>>> x = np.arange(-5, 5, 0.1)
>>> y = np.arange(-5, 5, 0.1)
>>> xx, yy = meshgrid(x, y, sparse=True)
```

```
    >>> z = np.sin(xx**2 + yy**2) / (xx**2 + yy**2)
    >>> h = plt.contourf(x,y,z)
```

1.3 Documentation

enumerate(*sequence*[, *start=0*])
Return an iterator that yields tuples of an index and an item of the *sequence*. (And so on.)

1.3.1 myclass

add(*x*, *y*)

Parameters

- **x** – the first value to be added
- **y** – the second, optional

Returns the sum of the two

Example add(1.,2.) # == 3

See also:

myclass.public_service

get_class(*x*)
returns a member of myclass right away

Parameters **x** (*array_like*) – means something, but forgot...

Returns **u** – the result

Return type array

and we have a snippet for you!

```
    a = get_class()
    a.kiss()
```

class **myclass**
a very fine class indeed!

startme()
starts the class and makes it run

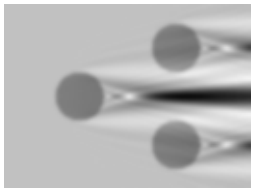
public_service()
sowas

Parameters

- **x2, ..., xn** (*x1,*) – 1-D arrays representing the coordinates of a grid.
- **indexing** (*{'xy', 'ij'}, optional*) – Cartesian ('xy', default) or matrix ('ij') indexing of output. See Notes for more details.

  New in version 1.7.0.
- **sparse** (*bool, optional*) – If True a sparse grid is returned in order to conserve memory. Default is False.

1.4 Beam propagation



psf(*shape*, *units*, *lam*, *NA*, *n0=1.0*, *n_integration_steps=200*, *return_field=False*)

Parameters

- **shape** (*Nx, Ny, Nz*) – the shape of the geometry
- **units** (*dx, dy, dz*) – the pixel sizes in microns
- **lam** (*float*) – the wavelength
- **NA** –
- **n0** –
- **n_integration_steps** –
- **return_field** –

Returns

- *calculates the 3d psf for a perfect, aberration free optical system*
- *via the vectorial debye diffraction integral*
- *the psf is centered at a grid of given size with voxelsizes units*

see [1]

returns: u, the (not normalized) intensity

or if return_field = True u,ex,ey,ez

NA can be either a single number or an even length list of NAs (for bessel beams), e.g. NA = [.1,.2,.5,.6] lets light through the annulus .1<.2 and .5<.6

References

psf_u0(*shape*, *units*, *zfoc=0*, *NA=0.4*, *lam=0.5*, *n0=1.0*, *n_integration_steps=200*)
calculates initial plane u0 of a beam focused at zfoc shape = (Nx,Ny) units = (dx,dy) NAs = e.g. (0,.6)

bpm_3d(*size*, *units*, *lam=0.5*, *u0=None*, *dn=None*, *subsample=1*, *n_volumes=1*, *n0=1.0*, *return_scattering=False*, *return_g=False*, *return_full=True*, *absorbing_width=0*, *use_fresnel_approx=False*, *scattering_plane_ind=0*)
simulates the propagation of monochromativ wave of wavelength lam with initial conditions u0 along z in a media filled with dn

size - the dimension of the image to be calulcated in pixels (Nx,Ny,Nz) units - the unit lengths of each dimensions in microns lam - the wavelength u0 - the initial field distribution, if u0 = None an incident plane wave is assumed dn - the refractive index of the medium (can be complex) n0 - refractive index of surrounding medium return_full - if True, returns the complex field in volume otherwise only last plane

[1] Matthew R. Foreman, Peter Toeroek, *Computational methods in vectorial imaging*, Journal of Modern Optics, 2011, 58, 5-6, 339

1.5 Docstring examples

Huhu

citing_me()
please cite [1]

and we do have footnotes [2] as well!

$$e^{-\alpha x} = \int_0^1 dk f(k)$$

References

google_style(*x*, *y=None*, *fname=''*)

Parameters

- **x** (*int*) – the first value
- **y** (*float*) – nothing
- **fname** (*str*) – obvious

Returns everything you never cared about

numpy_style()
Return a new array of given shape and type, filled with zeros.

Parameters

- **shape** (*int or sequence of ints*) – Shape of the new array, e.g., `(2, 3)` or `2`.
- **dtype** (*data-type, optional*) – The desired data-type for the array, e.g., *numpy.int8*. Default is *numpy.float64*.
- **order** (*{'C', 'F'}, optional*) – Whether to store multidimensional data in C- or Fortran-contiguous (row- or column-wise) order in memory.

Returns **out** – Array of zeros with the given shape, dtype, and order.

Return type ndarray

See also:

zeros_like() Return an array of zeros with shape and type of input.

ones_like() Return an array of ones with shape and type of input.

empty_like() Return an empty array with shape and type of input.

ones() Return a new array setting values to one.

empty() Return a new uninitialized array.

Examples

[1] Hugo Beierthal (2013) *fastsomething: We wish you a merry christmas*
[2] Text fo the footnote

```
    >>> np.zeros(5)
    array([ 0.,  0.,  0.,  0.,  0.])
```

```
    >>> np.zeros((5,), dtype=np.int)
    array([0, 0, 0, 0, 0])
```

```
    >>> np.zeros((2, 1))
    array([[ 0.],
           [ 0.]])
```

```
    >>> s = (2,2)
    >>> np.zeros(s)
    array([[ 0.,  0.],
           [ 0.,  0.]])
```

```
    >>> np.zeros((2,), dtype=[('x', 'i4'), ('y', 'i4')]) # custom dtype
    array([(0, 0), (0, 0)],
          dtype=[('x', '<i4'), ('y', '<i4')])
```

rst_style(*x*, *y=None*, *fname=''*)

Parameters

- **path** (*str*) – The path of the file to wrap
- **field_storage** (*FileStorage*) – The FileStorage instance to wrap
- **temporary** – Whether or not to delete the file when the File

Returns A buffered writable file descriptor

Return type BufferedFileStorage

sphynx_style(*x*, *y=None*, *fname=''*)

Parameters

- **x** (*str*) – the first value
- **y** (*float, int, ndarray*) – the second value
- **fname** (*str*) – the name tow rite to

Returns 1 on sucess

Every great project starts with a line

- genindex
- modindex
- search

a

A

B

C

E

G

M

N

P

R

S