MMFUtils Documentation

Release 0.4.10

Michael McNeil Forbes

Sep 18, 2018

Contents

Contents:

CHAPTER 1

mmfutils

1.1 mmfutils.interface

Stand-in for zope.interface if it is not available.

class mmfutils.interface.**Attribute**(*__name__*, *__doc__*='')
Bases: zope.interface.interface.Element

Attribute descriptions

interface = None

mmfutils.interface.**implements**(**interfaces*)
Declare interfaces implemented by instances of a class

This function is called in a class definition.

The arguments are one or more interfaces or interface specifications (IDeclaration objects).

The interfaces given (including the interfaces in the specifications) are added to any interfaces previously declared.

Previous declarations include declarations for base classes unless implementsOnly was used.

This function is provided for convenience. It provides a more convenient way to call classImplements. For example:

```
implements(I1)
```

is equivalent to calling:

```
classImplements(C, I1)
```

after the class has been created.

mmfutils.interface.**classImplements**(*cls*, **interfaces*)
Declare additional interfaces implemented for instances of a class

The arguments after the class are one or more interfaces or interface specifications (`IDeclaration` objects).

The interfaces given (including the interfaces in the specifications) are added to any interfaces previously declared.

`mmfutils.interface.`**`verifyObject`**(*iface*, *candidate*, *tentative=0*)

`mmfutils.interface.`**`verifyClass`**(*iface*, *candidate*, *tentative=0*)

`mmfutils.interface.`**`describe_interface`**(*interface*, *format='ipython'*)

Return an HTML object for Jupyter notebooks that describes the interface.

Parameters

- **`interface`** (*`Interface`*) – Interface to extract documentation from.
- **`format`** (*`'rst'`, `'html'`, `'ipython'`*) – Return format. 'rst' is raw ReStructuredText, 'html' is packaged as HTML, and 'ipython' is packaged as an IPython.display.HTML() object suitable for Jupyter notebooks.

Example

```
>>> class IExample(Interface):
...     x = Attribute("Floating point number")
...     def two():
...         "Return two"
>>> print(describe_interface(IExample, format='rst').strip())
``IExample``

 Attributes:

  ``x`` -- Floating point number

Methods:

  ``two()`` -- Return two
```

You can also get this wrapped as HTML:

```
>>> print(describe_interface(IExample, format='html').strip())
<!DOCTYPE html ...
<p><tt class="docutils literal">IExample</tt></p>
<blockquote>
<p>Attributes:</p>
<blockquote>
<tt class="docutils literal">x</tt> -- Floating point number</blockquote>
<p>Methods:</p>
<blockquote>
<tt class="docutils literal">two()</tt> -- Return two</blockquote>
</blockquote>
</div>
```

In a Jupyter notebook, this will properly display:

```
>>> describe_interface(IExample)
<IPython.core.display.HTML object>
```

Other formats are not yet supported:

```
>>> describe_interface(IExample, format='WYSIWYG')
Traceback (most recent call last):
   ...
NotImplementedError: format WYSIWYG not supported
```

1.2 mmfutils.containers

Provides convenience containers that support pickling and archiving.

Archiving is supported through the interface defined by the `persist` package (though use of that package is optional and it is not a dependency).

class mmfutils.containers.**Object**

Bases: object

General base class with a few convenience methods.

Constructors: The *__init__* method should simply be used to set variables, all initialization that computes attributes etc. should be done in *init()* which will be called at the end of *__init__*.

This aids pickling which will save only those variables defined when the base *__init__* is finished, and will call *init()* upon unpickling, thereby allowing unpickable objects to be used (in particular function instances).

Note: Do not use any of the following variables:

- *_empty_state*: reserved for objects without any state
- *_independent_attributes*, *_dependent_attributes*, *initialized*: Used to flag if attributes have been changed but without *init()* being called. (See below.)

By default setting any attribute in *picklable_attributes* will set the *initialized* flag to *False*. This will be set to *True* when *init()* is called. Objects can then include an *assert self.initialized* in the appropriate places.

To allow for some variables to be set without invalidating the object we also check the set of names *_independent_attributes*.

Note: This redefines __setattr__ to provide the behaviour.

Examples

```
>>> class A(Object):
...     def __init__(self, x=0):
...         self.x = x
...         Object.__init__(self)
...     def init(self):
...         self.x1 = self.x + 1    # A dependent variable
...         Object.init(self)
...     def check(self):
...         if not self.initialized:
...             raise AssertionError("Please call init()!")
...         return self.x1 == self.x + 1
>>> a = A(x=0)
```

(continues on next page)

(continued from previous page)

```
>>> a.check()
True
>>> a.x = 2.0
>>> a.check()
Traceback (most recent call last):
...
AssertionError: Please call init()!
>>> a.init()
>>> a.check()
True
```

get_persistent_rep(*env*)
Return *(rep, args, imports)*.

Define a persistent representation *rep* of the instance self where the instance can be reconstructed from the string rep evaluated in the context of dict args with the specified imports = list of *(module, iname, uiname)* where one has either *import module as uiname*, *from module import iname* or *from module import iname as uiname*.

This satisfies the `IArchivable` interface for the `persist` package.

init()
Define any computed attributes here.

Don't forget to call *Object.init()*

initialized = False

picklable_attributes = ()

class mmfutils.containers.**Container**(**argv*, ***kw*)
Bases: *mmfutils.containers.Object*, _abcoll.Sized, _abcoll.Iterable, _abcoll.Container

Simple container object.

Attributes can be specified in the constructor. These will form the representation of the object as well as picking. Additional attributes can be assigned, but will not be pickled.

Examples

```
>>> c = Container(b='Hi', a=1)
>>> c                       # Note: items sorted for consistent repr
Container(a=1, b='Hi')
>>> c.a
1
>>> c.a = 2
>>> c.a
2
>>> tuple(c)                # Order is lexicographic
(2, 'Hi')
>>> c.x = 6                 # Will not be pickled: only for temp usage
>>> c.x
6
>>> 'a' in c
True
>>> 'x' in c
```

(continues on next page)

(continued from previous page)

```
False
>>> import pickle
>>> c1 = pickle.loads(pickle.dumps(c))
>>> c1
Container(a=2, b='Hi')
>>> c1.x
Traceback (most recent call last):
...
AttributeError: 'Container' object has no attribute 'x'
```

class mmfutils.containers.**ContainerList**(**argv*, ***kw*)

Bases: *mmfutils.containers.Container*, _abcoll.Sequence

Simple container object that behaves like a list.

Examples

```
>>> c = ContainerList(b='Hi', a=1)
>>> c                          # Note: items sorted for consistent repr
ContainerList(a=1, b='Hi')
>>> c[0]
1
>>> c[0] = 2
>>> c.a
2
>>> tuple(c)                   # Order is lexicographic
(2, 'Hi')
```

class mmfutils.containers.**ContainerDict**(**argv*, ***kw*)

Bases: *mmfutils.containers.Container*, _abcoll.MutableMapping

Simple container object that behaves like a dict.

Attributes can be specified in the constructor. These will form the representation of the object as well as picking. Additional attributes can be assigned, but will not be pickled.

Examples

```
>>> from collections import OrderedDict
>>> c = ContainerDict(b='Hi', a=1)
>>> c                          # Note: items sorted for consistent repr
ContainerDict(a=1, b='Hi')
>>> c['a']
1
>>> c['a'] = 2
>>> c.a
2
>>> OrderedDict(c)
OrderedDict([('a', 2), ('b', 'Hi')])
```

1.3 mmfutils.math

1.3.1 mmfutils.math.integrate

Integration Utilities.

mmfutils.math.integrate.**quad**(*f*, *a*, *b*, *epsabs=1e-12*, *epsrel=1e-08*, *limit=1000*, *points=None*, ***kwargs*)

An improved version of integrate.quad that does some argument checking and deals with points properly.

Return (ans, err).

Examples

```
>>> def f(x): return 1./x**2
>>> (ans, err) = quad(f, 1, np.inf, points=[])
>>> abs(ans - 1.0) < err
True
>>> (ans, err) = quad(f, 1, np.inf, points=[3.0, 2.0])
>>> abs(ans - 1.0) < err
True
```

mmfutils.math.integrate.**mquad**(*f*, *a*, *b*, *abs_tol=1e-12*, *verbosity=0*, *fa=None*, *fb=None*, *save_fx=False*, *res_dict=None*, *max_fcnt=10000*, *min_step_size=None*, *norm=<function <lambda>>*, *points=None*)

Return (res, err) where res is the numerically evaluated integral using adaptive Simpson quadrature.

mquad tries to approximate the integral of function f from a to b to within an error of abs_tol using recursive adaptive Simpson quadrature. mquad allows the function y = f(x) to be array-valued. In the matrix valued case, the infinity norm of the matrix is used as it's "absolute value".

Parameters

- **f** (*function*) – Possibly array valued function to integrate. If this emits a NaN, then an AssertionError is raised to allow the user to optimize this check away (as it exists in the core of the loops)
- **b** (*a,*) – Integration range (a, b)
- **fb** (*fa,*) – f(a) and f(b) respectively (if already computed)
- **abs_tol** (*float*) – Approximate absolute tolerance on integral
- **verbosity** (*int*) – Display info if greater than zero. Shows the values of [fcnt a b-a Q] during the iteration.
- **save_fx** (*bool*) – If True, then save the abscissa and function values in res_dict.
- **res_dict** (*dict*) – Details are stored here. Pass a dictionary to access these. The dictionary will be modified.

  fcnt : Number of function evaluations. xy : List of pairs (x, f(x)) if save_fx is defined.
- **max_fcnt** (*int*) – Maximum number of function evaluations.
- **min_step_size** (*float*) – Minimum step size to limit recursion.
- **norm** (*function*) – Norm to use to determine convergence. The absolute error is determined as *norm(f(x) - F)*.

- **points** (*[float]*) – List of special points to be included in abscissa.

Notes

Based on "adaptsim" by Walter Gander. Ref: W. Gander and W. Gautschi, "Adaptive Quadrature Revisited", 1998. http://www.inf.ethz.ch/personal/gander

Examples

Orthogonality of planewaves on [0, 2pi]

```
>>> def f(x):
...     v = np.exp(1j*np.array([[1.0, 2.0, 3.0]])*x)
...     return v.T.conj()*v/2.0/np.pi
>>> ans = mquad(f, 0, 2*np.pi)
>>> abs(ans - np.eye(ans.shape[0])).max() < _abs_tol
True
```

```
>>> res_dict = {}
>>> def f(x): return x**2
>>> ans = mquad(f, -2, 1, res_dict=res_dict, save_fx=True)
>>> abs(ans - 3.0) < _abs_tol
True
>>> x = np.array([xy[0] for xy in res_dict['xy']])
>>> y = np.array([xy[1] for xy in res_dict['xy']])
>>> abs(y - f(x)).max()
0.0
```

This works, but triggers a warning because of the singular # endpoints. >>> logger = logging.getLogger() >>> logger.disabled = True >>> def f(x): return 1.0/np.sqrt(x) + 1.0/np.sqrt(1.0-x) >>> abs(mquad(f, 0, 1, abs_tol=1e-8) - 4.0) < 1e-8 True >>> logger.disabled = False

```
>>> def f(x):
...     if x < 0:
...         return 0.0
...     else:
...         return 1.0
>>> abs(mquad(f, -2.0, 1.0) - 1.0) < 1e-10
True
```

```
>>> def f(x): return 1./x
>>> mquad(f, 1, np.inf)
Traceback (most recent call last):
    ...
ValueError: Infinite endpoints not supported.
```

mmfutils.math.integrate.**Richardson**(*f*, *ps=None*, *l=2*, *n0=1*)

Compute the Richardson extrapolation of f given the function

$$f(N) = f + \sum_{n=0}^{\infty} \frac{\alpha_n}{N^{p_n}}$$

The extrapolants are stored in the array *S'[n, s] where 'S[n, 0] = f(n0*l**n)* and *S[n, s]* is the s'th extrapolant.

Note: It is crucial for performance that the powers p_n be properly characterized. If you do not know the form of the errors, then consider using a non-linear acceleration technique such as `levin_sum()`.

Parameters **ps** (*iterable*) – Iterable returning the powers p_n. To generate the sequence $p_0 + m d_p$ for example, use `itertools.count``(p0, dp)()`.

Examples

Here we consider

$$f(N) = \sum_{n=1}^{N} \frac{1}{n^2} = \frac{\pi^2}{6} + (N^{-1})$$

```
>>> def f(N): return sum(np.arange(1, N+1, dtype=float)**(-2))
>>> r = Richardson(f, l=3, n0=2)
>>> for n in range(9):
...     x = next(r)
>>> err = abs(x - np.pi**2/6.0)
>>> assert err < 1e-14, 'err'
```

Now some other examples with different p values:

$$f(N) = \sum_{n=1}^{N} \frac{1}{n^4} = \frac{\pi^4}{90} + (N^{-3})$$

```
>>> def f(N): return sum(np.arange(1, N+1, dtype=float)**(-4))
>>> r = Richardson(f, ps=itertools.count(3,1))
>>> for n in range(8):
...     x = next(r)
>>> err = abs(x - np.pi**4/90.0)
>>> assert err < 1e-14, 'err'
```

$$f(N) = \sum_{n=1}^{N} \frac{1}{n^6} = \frac{\pi^6}{945} + (N^{-5})$$

```
>>> def f(N): return sum(np.arange(1, N+1, dtype=float)**(-6))
>>> r = Richardson(f, ps=itertools.count(5))
>>> for n in range(7):
...     x = next(r)
>>> err = abs(x - np.pi**6/945.0)
>>> assert err < 1e-14, 'err'
```

Richardson works with array valued functions:

```
>>> def f(N): return np.array([sum(np.arange(1, N+1, dtype=float)**(-2)),
...                            sum(np.arange(1, N+1, dtype=float)**(-4))])
>>> r = Richardson(f, l=3, n0=2)
>>> for n in range(7):
...     x = next(r)
>>> err = abs(x - np.array([np.pi**2/6.0, np.pi**4/90.0])).max()
>>> assert err < 1e-13, 'err'
```

It also works for complex valued functions:

```
>>> def f(N): return (sum(np.arange(1, N+1, dtype=float)**(-2)) +
...                   1j*sum(np.arange(1, N+1, dtype=float)**(-4)))
>>> r = Richardson(f, l=3, n0=2)
>>> for n in range(7):
...     x = next(r)
>>> err = abs(x - (np.pi**2/6.0 + 1j*np.pi**4/90.0))
>>> assert err < 1e-13, 'err'
```

mmfutils.math.integrate.**rsum**(*f*, *N0=0*, *ps=None*, *l=2*, *abs_tol=1e-12*, *rel_tol=1e-08*, *verbosity=0*)

Sum f using Richardson extrapolation.

Examples

```
>>> def f(n):
...     return 1./(n+1)**2
>>> res, err = rsum(f)
>>> res
1.6449340668...
>>> abs(res - np.pi**2/6.0) < err
True
```

1.3.2 mmfutils.math.differentiate

Differentiation.

mmfutils.math.differentiate.**differentiate**(*f, x=0.0, d=1, h0=1.0, l=1.4, nmax=10, dir=0, p0=1, err=[0]*)

Evaluate the numerical dth derivative of f(x) using a Richardson extrapolation of the finite difference formula.

Parameters

- **f** (*function*) – The function to compute the derivative of.
- **x** (*{float, array}*) – The derivative is computed at this point (or at these points if the function is vectorized.
- **d** (*int, optional*) – Order of derivative to compute. *d=0* is the function *f(x)*, *d=1* is the first derivative etc.
- **h0** (*float, optional*) – Initial stepsize. Should be on about a factor of 10 smaller than the typical scale over which *f(x)* varies significantly.
- **l** (*float, optional*) – Richardson extrapolation factor. Stepsizes used are *h0/l**n*
- **nmax** (*int, optional*) – Maximum number of extrapolation steps to take.
- **dir** (*int, optional*) – If *dir < 0*, then the function is only evaluated to the left, if positive, then only to the right, and if zero, then centered form is used.

Returns **df** – Order *d* derivative of *f* at *x*.

Return type {float, array}

Other Parameters

- **p0** (*int, optional*) – This is the first non-zero term in the taylor expansion of either the difference formula. If you know that the first term is zero (because of the coefficient), then you should set *p0=2* to skip that term.

  Note: This is not the power of the term, but the order. For centered difference formulae, *p0=1* is the *h**2* term, which would vanish if third derivative vanished at *x* while for the forward difference formulae this is the *h* term which is absent if the second derivative vanishes.

- **err[0]** (*float*) – This is mutated to provide an error estimate.

Notes

This implementation uses the Richardson extrapolation to extrapolate the answer. This is based on the following Taylor series error formulae:

$$\begin{aligned} f'(x) &= \frac{f(x+h) - f(x)}{h} - h\frac{f'')}{2} + \cdots \\ &= \frac{f(x+h) - f(x-h)}{2h} - h^2\frac{f''}{3!} + \cdots \\ f''(x) &= \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2} - hf^{(3)} + \cdots \\ &= \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} - 2h^2\frac{f^{(4)}}{4!} + \cdots \end{aligned}$$

If we let \$h = 1/N\$ then these formula match the expected error model for the Richardson extrapolation

$$S(h) = S(0) + ah^p + (h^{p+1})$$

with \$p=1\$ for the one-sided formulae and \$p=2\$ for the centered difference formula respectively.

See `mmf.math.integrate.Richardson`

See also:

mmfutils.math.integrate.Richardson() Richardson extrapolation

Examples

```
>>> from math import sin, cos
>>> x = 100.0
>>> assert(abs(differentiate(sin, x, d=0)-sin(x))<1e-15)
>>> assert(abs(differentiate(sin, x, d=1)-cos(x))<1e-14)
>>> assert(abs(differentiate(sin, x, d=2)+sin(x))<1e-13)
>>> assert(abs(differentiate(sin, x, d=3)+cos(x))<1e-11)
>>> assert(abs(differentiate(sin, x, d=4)-sin(x))<1e-9)
>>> differentiate(abs, 0.0, d=1, dir=1)
1.0
>>> differentiate(abs, 0.0, d=1, dir=-1)
-1.0
>>> differentiate(abs, 0.0, d=1, dir=0)
0.0
```

Note that the Richardson extrapolation assumes that *h0* is small enough that the truncation errors are controlled by the taylor series and that the taylor series properly describes the behaviour of the error. For example, the following will not converge well, even though the derivative is well defined:

```
>>> def f(x):
...     return np.sign(x)*abs(x)**(1.5)
>>> df = differentiate(f, 0.0)
>>> abs(df) < 0.1
True
>>> abs(df) < 0.01
False
>>> abs(differentiate(f, 0.0, nmax=100)) < 3e-8
True
```

Sometimes, one may compensate by increasing nmax. (One could in principle change the Richardson parameter p, but this is optimized for analytic functions.)

The *differentiate()* also works over arrays if the function *f* is vectorized:

```
>>> x = np.linspace(0, 100, 10)
>>> assert(max(abs(differentiate(np.sin, x, d=1) - np.cos(x))) < 3e-15)
```

mmfutils.math.differentiate.**hessian**(*f*, *x*, ***kw*)

Return the gradient Hessian matrix of *f(x)* at *x* using *differentiate()*. This is not efficient.

Parameters

- **f** (*function*) – Scalar function of an array.
- **x** (*array-like*) – Derivatives evaluated at this point.
- **kw** (*dict*) – See *differentiate()* for options.

Examples

```
>>> def f(x): return np.arctan2(*x)
>>> def df(x): return np.array([x[1], -x[0]])/np.sum(x**2)
>>> def ddf(x):
...     return np.array([[-2.*x[0]*x[1], -np.diff(x**2)],
...                      [-np.diff(x**2), 2.*x[0]*x[1]]])/np.sum(x**2)**2
>>> x = [0.1, 0.2]
>>> D, H = hessian(f, x, h0=0.1)
>>> x= np.asarray(x)
>>> D, df(x)
(array([ 4., -2.]), array([ 4., -2.]))
>>> H, ddf(x)
(array([[-16., -12.],
        [-12.,  16.]]),
 array([[-16., -12.],
        [-12.,  16.]]))
```

1.4 mmfutils.optimize

Optimization tools.

mmfutils.optimize.**bracket_monotonic**(*f*, *x0=0.0*, *x1=1.0*, *factor=2.0*)
Return *(x0, x1)* where *f(x0)*f(x1) < 0*.

Assumes that *f* is monotonic and that the root exists.

Proceeds by increasing the size of the interval by *factor* in the direction of the root until the root is found.

Examples

```
>>> import math
>>> bracket_monotonic(lambda x:10 - math.exp(x))
(0.0, 3.0)
>>> bracket_monotonic(lambda x:10 - math.exp(-x), factor=1.5)
(4.75, -10.875)
```

mmfutils.optimize.**ubrentq**(*f*, *a*, *b*, **v*, ***kw*)
Version of *scipy.optimize.brentq* with uncertainty processing using the uncertainties package.

mmfutils.optimize.**usolve**(*f*, *a*, **v*, ***kw*)
Return the root of *f(x) = 0* with uncertainties propagated.

Parameters

- **f** (*function*) – Function to find root of *f(x) = 0*. Note: this must work with only a single argument even if the solver supports *args* etc. Thus, use *lambda x: f(x, ...)* or *functools.partial* if needed.
- **solver** (*function*) – Solver function (default is scipy.optimize.brentq).
- **kw** (*v,*) – Remaining arguments will be passed as *solver(f, a, *v, **kw)*.

1.5 mmfutils.performance

1.5.1 mmfutils.performance.blas

1.5.2 mmfutils.performance.fft

FFTW wrappers for high-performance computing.

This module requires you to have installed the fftw libraries and `pyfftw`. Note that you must build the fftw with all precisions using something like:

```
PREFIX=/data/apps/fftw
VER=3.3.4
for opt in " " "--enable-sse2 --enable-single"                "--enable-long-double"
↪"--enable-quad-precision"; do
  ./configure --prefix="${PREFIX}/${VER}"                  --enable-threads
↪       --enable-shared                  $opt
  make -j8 install
done
```

Note: The FFTW library does not work with negative indices for axis. Indices should first be normalized by `inds % len(shape)`.

mmfutils.performance.fft.**fft**(*Phi*, *axis=-1*)

mmfutils.performance.fft.**ifft**(*Phit*, *axis=-1*)

mmfutils.performance.fft.**fftn**(*Phi*, *axes=None*)

mmfutils.performance.fft.**ifftn**(*Phit*, *axes=None*)

mmfutils.performance.fft.**fftfreq**(*n*, *d=1.0*)

Return the Discrete Fourier Transform sample frequencies.

The returned float array *f* contains the frequency bin centers in cycles per unit of the sample spacing (with zero at the start). For instance, if the sample spacing is in seconds, then the frequency unit is cycles/second.

Given a window length *n* and a sample spacing *d*:

```
f = [0, 1, ...,   n/2-1,     -n/2, ..., -1] / (d*n)   if n is even
f = [0, 1, ..., (n-1)/2, -(n-1)/2, ..., -1] / (d*n)   if n is odd
```

Parameters

- **n** (*int*) – Window length.
- **d** (*scalar, optional*) – Sample spacing (inverse of the sampling rate). Defaults to 1.

Returns **f** – Array of length *n* containing the sample frequencies.

Return type ndarray

Examples

```
>>> signal = np.array([-2, 8, 6, 4, 1, 0, 3, 5], dtype=float)
>>> fourier = np.fft.fft(signal)
>>> n = signal.size
>>> timestep = 0.1
>>> freq = np.fft.fftfreq(n, d=timestep)
>>> freq
array([ 0.  ,  1.25,  2.5 ,  3.75, -5.  , -3.75, -2.5 , -1.25])
```

mmfutils.performance.fft.**resample**(*f*, *N*)

Resample f to a new grid of size N.

This uses the FFT to resample the function *f* on a new grid with *N* points. Note: this assumes that the function *f* is periodic. Resampling non-periodic functions to finer lattices may introduce aliasing artifacts.

Parameters

- **f** (*array*) – The function to be resampled. May be n-dimensional
- **N** (*int or array*) – The number of lattice points in the new array. If this is an integer, then all dimensions of the output array will have this length.

Examples

```
>>> def f(x, y):
...     "Function with only low frequencies"
...     return (np.sin(2*np.pi*x)-np.cos(4*np.pi*y))
>>> L = 1.0
>>> Nx, Ny = 16, 13   # Small grid
>>> NX, NY = 31, 24   # Large grid
>>> dx, dy = L/Nx, L/Ny
>>> dX, dY = L/NX, L/NY
```

(continues on next page)

(continued from previous page)

```
>>> x = (np.arange(Nx)*dx - L/2)[:, None]
>>> y = (np.arange(Ny)*dy - L/2)[None, :]
>>> X = (np.arange(NX)*dX - L/2)[:, None]
>>> Y = (np.arange(NY)*dY - L/2)[None, :]
>>> f_XY = resample(f(x,y), (NX, NY))
>>> np.allclose(f_XY, f(X,Y))                          # To larger grid
True
>>> np.allclose(resample(f_XY, (Nx, Ny)), f(x,y))   # Back down
True
```

1.5.3 mmfutils.performance.numexpr

Tools for working with Numexp.

At present all this module provides is a safe way of importing `numexpr`. This prevents a hard crash (i.e. segfault) when the MKL is enabled but cannot be found. Just go:

```
>>> from mmfutils.performance.numexpr import numexpr
```

1.5.4 mmfutils.performance.threads

Thread Control

This module provides control of the number of threads used by the MKL and numexpr. It uses the global set SET_THREAD_HOOKS which should contain functions that take a single argument and set the number of threads for that particular part of the system.

Use *set_num_threads(nthreads)* to call all of these hooks.

`mmfutils.performance.threads.`**`set_num_threads`**(*nthreads*)

Set the maximum number of threads to use.

Calls all the hooks in *mmfutils.performance.threads.SET_THREAD_HOOKS*

Tools for high-performance computing.

This module may rely on many other packages that are not easy to install such as pyfftw and the corresponding fftw implementation.

1.6 mmfutils.plot

1.6.1 mmfutils.plot.viridis

Plotting utilities for matplotlib.

`mmfutils.plot.`**`contourf`**(**v*, ***kw*)

Replacement for `matplotlib.pyplot.contourf()` that supports the *rasterized* keyword.

`mmfutils.plot.`**`imcontourf`**(*x*, *y*, *z*, *interpolate=True*, *diverging=False*, **v*, ***kw*)

Like `matplotlib.pyplot.contourf()` but does not actually find contours. Just displays z using `matplotlib.pyplot.imshow()` which is much faster and uses exactly the information available.

Parameters

- **y, z** (*x,*) – Assumes that *z* is ordered as *z[x, y]*. If *x* and *y* have the same shape as *z*, then *x = x[:, 0]* and *y = y[0, :]* are used. Otherwise, *z.shape == (len(x), len(y))*. *x* and *y* must be equally spaced.
- **interpolate** (*bool*) – If *True*, then interpolate the function onto an evenly spaced set of abscissa using cublic splines.
- **diverging** (*bool*) – If *True*, then the output is normalized so that diverging colormaps will have 0 in the middle. This is done by setting *vmin* and *vmax* symmetrically.

mmfutils.plot.**phase_contour**(*x*, *y*, *z*, *N=10*, *colors='k'*, *linewidths=0.5*, ***kw*)
Specialized contour plot for plotting the contours of constant phase for the complex variable z. Plots *4*N* contours in total. Note: two sets of contours are returned, and, due to processing, these do not have the correct values.

The problem this solves is that plotting the contours of *np.angle(z)* gives a whole swath of contours at the discontinuity between *-pi* and *pi*. We get around this by doing two things:

1. We plot the contours of *abs(angle(z))*. This almost fixes the problem, but can give rise to spurious closed contours near zero and *pi*. To deal with this:
2. We plot only the contours between *pi/4* and *3*pi/4*. We do this twice, multiplying *z* by *exp(0.5j*pi)*.
3. We carefully choose the contours so that they have even spacing.

mmfutils.plot.**plot_errorbars**(*x*, *y*, *dx=None*, *dy=None*, *colour=''*, *linestyle=''*, *pointstyle='.'*, *barwidth=0.5*, ***kwargs*)

mmfutils.plot.**plot_err**(*x*, *y*, *yerr=None*, *xerr=None*, ***kwarg*)
Plot x vs. y with errorbars.

Right now we support the following cases: x = 1D, y = 1D

mmfutils.plot.**error_line**(*x*, *y*, *dy*, *fgc='k'*, *bgc='w'*, *N=20*, *fill=True*)
Plots a curve (x, y) with gaussian errors dy represented by shading out to 5 dy.

mmfutils.plot.**MidpointNormalize**

1.7 mmfutils.debugging

Some debugging tools.

Most of these are implemented as decorators.

class mmfutils.debugging.**persistent_locals**(*func*)
Bases: object

Decorator that stores the function's local variables.

Examples

```
>>> @persistent_locals
... def f(x):
...     y = x**2
...     z = 2*y
...     return z
>>> f(1)
2
```

(continues on next page)

(continued from previous page)

```
>>> sorted(f.locals.items())
[('x', 1), ('y', 1), ('z', 2)]
>>> f.clear_locals()
>>> f.locals
{}
```

clear_locals()

locals

mmfutils.debugging.**debug**(*v, **kw)

Decorator to wrap a function and dump its local scope.

Parameters **(or env)** (*locals*) – Function's local variables will be updated in this dict. Use locals() if desired.

Examples

```
>>> env = {}
>>> @debug(env)
... def f(x):
...     y = x**2
...     z = 2*y
...     return z
>>> f(1)
2
>>> sorted(env.items())
[('x', 1), ('y', 1), ('z', 2)]
```

This will put the local variables directly in the global scope:

```
>>> @debug(locals())
... def f(x):
...     y = x**2
...     z = 2*y
...     return z
>>> f(1)
2
>>> x, y, z
(1, 1, 2)
>>> f(2)
8
>>> x, y, z
(2, 4, 8)
```

If an exception is raised, you still have access to the results:

```
>>> env = {}
>>> @debug(env)
... def f(x):
...     y = 2*x
...     z = 2/y
...     return z
>>> f(0)
Traceback (most recent call last):
  ...
```

(continues on next page)

(continued from previous page)

```
  File "<doctest mmfutils.debugging.debug[14]>", line 1, in <module>
    f(0)
  File "<doctest mmfutils.debugging.debug[13]>", line 4, in f
    z = 2/y
ZeroDivisionError: division by zero
>>> sorted(env.items())
[('x', 0), ('y', 0)]
```

1.8 mmfutils.monkeypatches

1.9 mmfutils.parallel

CHAPTER 2

MMF Utils

Small set of utilities: containers and interfaces.

This package provides some utilities that I tend to rely on during development. Presently it includes some convenience containers, plotting tools, and a patch for including zope.interface documentation in a notebook.

(Note: If this file does not render properly, try viewing it through nbviewer.org)

Documentation: http://mmfutils.readthedocs.org

Source: https://bitbucket.org/mforbes/mmfutils

Issues: https://bitbucket.org/mforbes/mmfutils/issues

Build Status:

Main

Fork

mmfutils Build Status

mmfutils-fork Build Status

Table of Contents

2.1 Installing

This package can be installed from from the bitbucket project:

```
pip install hg+https://bitbucket.org/mforbes/mmfutils
```

CHAPTER 3

Usage

3.1 Containers

3.1.1 Object

The `Object` object provides a base class to satisfy the following use-case.

Serialization and Deferred Initialization: Consider a problem where a class is defined through a few parameters, but requires extensive initialization before it can be properly used. An example is a numerical simulation where one passes the number of grid points N and a length L, but the initialization must generate large grids for efficient use later on. These grids should not be pickled when the object is serialized: instead, they should be generated at the end of initialization. By default, everything in `__dict__` will be pickled, leading to bloated pickles. The solution here is to split initialization into two steps: `__init__()` should initialize everything that is picklable, then `init()` should do any further initialization, defining the grid points based on the values of the picklable attributes. To do this, the semantics of the `__init__()` method are changed slightly here. `Object.__init__()` registers all keys in `__dict__` as `self.picklable_attributes`. These and only these attributes will be pickled (through the provided `__getstate__` and `__setstate__` methods).

The intended use is for subclasses to set and defined all attributes that should be pickled in the `__init__()` method, then call `Object.__init__(self)`. Any additional initialization can be done after this call, or in the `init()` method (see below) and attributes defined after this point will be treated as temporary. Note, however, that unpickling an object will not call `__init__()` so any additional initialization required should be included in the `init()` method.

Deferred initialization via the ‘‘init()‘‘ method: The idea here is to defer any expensive initialization – especially that which creates large temporary data that should not be pickled – until later. This method is automatically called at the end of `Object.__init__()` and after restoring a pickle. A further use-case is to allow one to change many parameters, then reinitialize the object once with an explicit call to `init()`.

Object Example

```
__init__() called
init() called
State(L=1.0, N=256)
```

One feature is that a nice `repr()` of the object is produced. Now let's do a calculation:

```
True
```

Here we demonstrate pickling. Note that the pickle is very small, and when unpickled, `init()` is called to re-establish `s.x` and `s.k`.

```
169
init() called
```

Another use case applies when `init()` is expensive. If x and k were computed in `__init__()`, then using properties to change both N and L would trigger two updates. Here we do the updates, then call `init()`. Good practice is to call `init()` automatically before any serious calculation to ensure that the object is brought up to date before the computation.

```
init() called
```

Finally, we demonstrate that `Object` instances can be archived using the `persist` package:

```
__init__() called
init() called
```

```
State(L=2.0, N=64)
```

3.1.2 Container

The `Container` object is a slight extension of `Object` that provides a simple container for storing data with attribute and iterative access. These implement some of the Collections Abstract Base Classes from the python standard library. The following containers are provided:

- `Container`: Bare-bones container extending the `Sized`, `Iterable`, and `Container` abstract ase classes (ABCs) from the standard `containers` library.
- `ContainerList`: Extension that acts like a tuple/list satisfying the `Sequence` ABC from the `containers` library (but not the `MutableSequence` ABC. Although we allow setting and deleting items, we do not provide a way for insertion, which breaks this interface.)
- `ContainerDict`: Extension that acts like a dict satisfying the `MutableMapping` ABC from the `containers` library.

These were designed with the following use cases in mind:

- Returning data from a function associating names with each data. The resulting `ContainerList` will act like a tuple, but will support attribute access. Note that the order will be lexicographic. One could use a dictionary, but attribute access with tab completion is much nicer in an interactive session. The `containers.nametuple` generator could also be used, but this is somewhat more complicated (though might be faster). Also, named tuples are immutable - here we provide a mutable object that is picklable etc. The choice between `ContainerList` and `ContainerDict` will depend on subsequent usage. Containers can be converted from one type to another.

Container Examples

```
Container(a=1, b='Hi there', c=2)
(1, 'Hi there', 2)
```

```
Container(a=1, b='Ho there', c=2)
```

```
Container(a=1, b='Ho there', c=2)
[[ 1.  1.  1. ...,  1.  1.  1.]
 [ 1.  1.  1. ...,  1.  1.  1.]
 [ 1.  1.  1. ...,  1.  1.  1.]
 ...,
 [ 1.  1.  1. ...,  1.  1.  1.]
 [ 1.  1.  1. ...,  1.  1.  1.]
 [ 1.  1.  1. ...,  1.  1.  1.]]
```

```
Container(a=1, b='Ho there', c=2)
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-9-c6cad315ac19> in <module>()
      2 c1 = pickle.loads(pickle.dumps(c))
      3 print c1
----> 4 c1.large_temporary_array


AttributeError: 'Container' object has no attribute 'large_temporary_array'
```

3.2 Contexts

The `mmfutils.contexts` module provides two useful contexts:

`NoInterrupt`: This can be used to susspend `KeyboardInterrupt` exceptions until they can be dealt with at a point that is convenient. A typical use is when performing a series of calculations in a loop. By placing the loop in a `NoInterrupt` context, one can avoid an interrupt from ruining a calculation:

Note: One can nest `NoInterrupt` contexts so that outer loops are also interrupted.

3.3 Interfaces

The interfaces module collects some useful zope.interface tools for checking interface requirements. Interfaces provide a convenient way of communicating to a programmer what needs to be done to used your code. This can then be checked in tests.

Here is a broken implementation. We muck up the arguments to `add`:

```
BrokenMethodImplementation: The implementation of add violates its contract
        because implementation requires too many arguments.
```

Now we get `add` right, but forget to define `value`. This is only caught when we have an object since the attribute is supposed to be defined in `__init__()`:

```
BrokenImplementation: An object has failed to implement interface <InterfaceClass __
↪main__.IAdder>

        The value attribute was not provided.
```

Finally, a working instance:

```
True
```

3.3.1 Interface Documentation

We also monkeypatch `zope.interface.documentation.asStructuredText()` to provide a mechanism for documentating interfaces in a notebook.

3.4 Parallel

The `mmfutils.parallel` module provides some tools for launching and connecting to IPython clusters. The `parallel.Cluster` class represents and controls a cluster. The cluster is specified by the profile name, and can be started or stopped from this class:

```
Waiting for connection file: ~/.ipython/profile_default/security/ipcontroller-client.
↪json
```

```
INFO:root:Starting cluster: ipcluster start --daemonize --quiet --profile=default --
↪n=3
```

```
Waiting for connection file: ~/.ipython/profile_default/security/ipcontroller-client.
↪json
```

```
INFO:root:waiting for 3 engines
INFO:root:0 of 3 running
INFO:root:3 of 3 running
INFO:root:Stopping cluster: ipcluster stop --profile=default
```

```
True
Waiting for connection file: ~/.ipython/profile_default/security/ipcontroller-client.
↪json
```

If you only need a cluster for a single task, it can be managed with a context. Be sure to wait for the result to be computed before exiting the context and shutting down the cluster!

```
Waiting for connection file: ~/.ipython/profile_default/security/ipcontroller-client.
↪json
```

```
INFO:root:Starting cluster: ipcluster start --daemonize --quiet --profile=default --
↪n=3
```

```
Waiting for connection file: ~/.ipython/profile_default/security/ipcontroller-client.
↪json
```

```
INFO:root:waiting for 3 engines
INFO:root:0 of 3 running
INFO:root:3 of 3 running
INFO:root:Stopping cluster: ipcluster stop --profile=default
```

```
Waiting for connection file: ~/.ipython/profile_default/security/ipcontroller-client.
↪json
True
```

If you just need to connect to a running cluster, you can use `parallel.get_client()`.

3.5 Performance

The `mmfutils.performance` module provides some tools for high performance computing. Note: this module requires some additional packages including numexp, pyfftw, and the `mkl` package installed by anaconda. Some of these require building system libraries (i.e. the FFTW). However, the various components will not be imported by default.

Here is a brief description of the components:

- `mmfutils.performance.blas`: Provides an interface to a few of the scipy BLAS wrappers. Very incomplete (only things I currently need).
- `mmfutils.performance.fft`: Provides an interface to the FFTW using `pyfftw` if it is available. Also enables the planning cache and setting threads so you can better control your performance.
- `mmfutils.performance.numexpr`: Robustly imports numexpr and disabling the VML. (If you don't do this carefully, it will crash your program so fast you won't even get a traceback.)
- `mmfutils.performance.threads`: Provides some hooks for setting the maximum number of threads in a bunch of places including the MKL, numexpr, and fftw.

3.6 Plotting

Several tools are provided in `mmfutils.plot`:

3.6.1 Fast Filled Contour Plots

`mmfutils.plot.imcontourf` is similar to matplotlib's `plt.contourf` function, but uses `plt.imshow` which is much faster. This is useful for animations and interactive work. It also supports my idea of saner array-shape processing (i.e. if `x` and `y` have different shapes, then it will match these to the shape of `z`). Matplotlib now provies `plt.pcolourmesh` which is similar, but has the same interface issues.

```
CPU times: user 9.77 ms, sys: 50 µs, total: 9.82 ms
Wall time: 9.86 ms
CPU times: user 43.5 ms, sys: 1.19 ms, total: 44.6 ms
Wall time: 44.7 ms
CPU times: user 426 ms, sys: 34.1 ms, total: 460 ms
Wall time: 450 ms
CPU times: user 2.39 ms, sys: 346 µs, total: 2.73 ms
Wall time: 2.74 ms
```

```
<matplotlib.collections.QuadMesh at 0x1209acd10>
```

3.7 Angular Variables

A couple of tools are provided to visualize angular fields, such as the phase of a complex wavefunction.

```
(<matplotlib.contour.QuadContourSet at 0x11558d8d0>,
 <matplotlib.contour.QuadContourSet at 0x115630b10>)
```

3.8 Debugging

A couple of debugging tools are provided. The most useful is the `debug` decorator which will store the local variables of a function in a dictionary or in your global scope.

```
(1.0, 2.0, 2.8284271247461903, 1.0)
```

3.9 Mathematics

We include a few mathematical tools here too. In particular, numerical integration and differentiation. Check the API documentation for details.

CHAPTER 4

Developer Instructions

If you are a developer of this package, there are a few things to be aware of.

1. If you modify the notebooks in `docs/notebooks` then you may need to regenerate some of the `.rst` files and commit them so they appear on bitbucket. This is done automatically by the `pre-commit` hook in `.hgrc` if you include this in your `.hg/hgrc` file with a line like:

```
%include ../.hgrc
```

Security Warning: if you do this, be sure to inspect the `.hgrc` file carefully to make sure that no one inserts malicious code.

This runs the following code:

```
[NbConvertApp] Converting notebook doc/README.ipynb to rst
[NbConvertApp] Support files will be in README_files/
[NbConvertApp] Making directory README_files
[NbConvertApp] Making directory README_files
[NbConvertApp] Writing 29492 bytes to README.rst
```

We also run a comprehensive set of tests, and the pre-commit hook will fail if any of these do not pass, or if we don't have complete code coverage. This uses nosetests and flake8. To run individal tests do one of:

```
python setup.py nosetests
python setup.py flake8
python setup.py check
python setup.py test   # This runs them all using a custom command defined in setup.py
```

Here is an example:

```
/data/apps/anaconda/envs/work/lib/python2.7/site-packages/setuptools-19.1.1-
↪py2.7.egg/setuptools/dist.py:284: UserWarning: Normalizing '0.4.7dev' to '0.
↪4.7.dev0'
running test
running nosetests
running egg_info
```

```
writing requirements to mmfutils.egg-info/requires.txt
writing mmfutils.egg-info/PKG-INFO
writing top-level names to mmfutils.egg-info/top_level.txt
writing dependency_links to mmfutils.egg-info/dependency_links.txt
reading manifest file 'mmfutils.egg-info/SOURCES.txt'
writing manifest file 'mmfutils.egg-info/SOURCES.txt'
nose.config: INFO: Set working dir to /Users/mforbes/work/mmfbb/mmfutils
nose.config: INFO: Ignoring files matching ['^\.', '^_', '^setup\.py$']
nose.plugins.cover: INFO: Coverage report will include only packages:␣
↪['mmfutils']
INFO:root:Patching zope.interface.document.asStructuredText to format code
INFO:root:Patching flake8 for issues 39 and 40
Doctest: mmfutils.containers.Container ... ok
Doctest: mmfutils.containers.ContainerDict ... ok
Doctest: mmfutils.containers.ContainerList ... ok
Doctest: mmfutils.containers.Object ... ok
Doctest: mmfutils.debugging.debug ... ok
Doctest: mmfutils.debugging.persistent_locals ... ok
Doctest: mmfutils.interface.describe_interface ... ok
Doctest: mmfutils.math.differentiate.differentiate ... ok
Doctest: mmfutils.math.differentiate.hessian ... ok
Test the Richardson extrapolation for the correct scaling behaviour. ... ok
Doctest: mmfutils.math.integrate.Richardson ... ok
Doctest: mmfutils.math.integrate.exact_add ... ok
Doctest: mmfutils.math.integrate.exact_sum ... ok
Doctest: mmfutils.math.integrate.mquad ... /Users/mforbes/work/mmfbb/mmfutils/
↪mmfutils/math/integrate/__init__.py:1: RuntimeWarning: divide by zero␣
↪encountered in double_scalars
  """Integration Utilities.
WARNING:root:mquad:MinStepSize: Minimum step size reached. (5.94368304574e-19
↪< 6.50521303491e-19) Singularity possible (err = 0.0).
WARNING:root:mquad:MinStepSize: Minimum step size reached. (5.94368304574e-19
↪< 6.50521303491e-19) Singularity possible (err = 1.98122768191e-19).
ok
Doctest: mmfutils.math.integrate.quad ... ok
Doctest: mmfutils.math.integrate.rsum ... ok
Doctest: mmfutils.math.integrate.ssum_inline ... ok
Doctest: mmfutils.math.integrate.ssum_python ... ok
Test directional first derivatives ... ok
Test directional second derivatives ... ok
Doctest: mmfutils.optimize.bracket_monotonic ... ok
Doctest: mmfutils.performance.fft.resample ... ok
Doctest: mmfutils.performance.numexpr ... ok
mmfutils.tests.test_containers.TestContainer.test_container_delattr ... ok
Test persistent representation of object class ... ok
Check that the order of attributes defined by ... ok
mmfutils.tests.test_containers.TestContainerConversion.test_conversions ... ok
mmfutils.tests.test_containers.TestContainerDict.test_container_del ... ok
mmfutils.tests.test_containers.TestContainerDict.test_container_setitem ... ok
mmfutils.tests.test_containers.TestContainerList.test_container_delitem ... ok
mmfutils.tests.test_containers.TestObject.test_empty_object ... ok
Test persistent representation of object class ... ok
mmfutils.tests.test_containers.TestPersist.test_archive ... ok
Doctest: mmfutils.tests.test_containers.Issue4 ... ok
```

```
mmfutils.tests.test_debugging.TestCoverage.test_coverage_1 ... ok
mmfutils.tests.test_debugging.TestCoverage.test_coverage_2 ... ok
mmfutils.tests.test_debugging.TestCoverage.test_coverage_3 ... ok
mmfutils.tests.test_debugging.TestCoverage.test_coverage_exception ... ok
Test 3rd order differentiation ... ok
mmfutils.tests.test_interface.TestInterfaces.test_verifyBrokenClass ... ok
mmfutils.tests.test_interface.TestInterfaces.test_verifyBrokenObject1 ... ok
mmfutils.tests.test_interface.TestInterfaces.test_verifyBrokenObject2 ... ok
mmfutils.tests.test_interface.TestInterfaces.test_verifyClass ... ok
mmfutils.tests.test_interface.TestInterfaces.test_verifyObject ... ok
Doctest: mmfutils.tests.test_interface.Doctests ... ok
mmfutils.tests.test_monkeypatchs.TestCoverage.test_cover_monkeypatchs ...␣
↪INFO:root:Patching flake8 for issues 39 and 40
ok
mmfutils.tests.test_monkeypatchs.TestCoverage.test_flake8_patch_err ...␣
↪INFO:root:Patching flake8 for issues 39 and 40
ok
[ProfileCreate] Generating default config file: u'/var/folders/m7/
↪dnr91tjs4gn58_t3k8zp_g000000gn/T/tmp9itx0A/profile_testing/ipython_config.
↪py'
[ProfileCreate] Generating default config file: u'/var/folders/m7/
↪dnr91tjs4gn58_t3k8zp_g000000gn/T/tmp9itx0A/profile_testing/ipython_kernel_
↪config.py'
[ProfileCreate] Generating default config file: u'/var/folders/m7/
↪dnr91tjs4gn58_t3k8zp_g000000gn/T/tmp9itx0A/profile_testing/ipcontroller_
↪config.py'
[ProfileCreate] Generating default config file: u'/var/folders/m7/
↪dnr91tjs4gn58_t3k8zp_g000000gn/T/tmp9itx0A/profile_testing/ipengine_config.
↪py'
[ProfileCreate] Generating default config file: u'/var/folders/m7/
↪dnr91tjs4gn58_t3k8zp_g000000gn/T/tmp9itx0A/profile_testing/ipcluster_config.
↪py'
INFO:root:Starting cluster: ipcluster start --daemonize --quiet --
↪profile=testing1 --n=7 --ipython-dir="/var/folders/m7/dnr91tjs4gn58_t3k8zp_
↪g000000gn/T/tmp9itx0A"
WARNING:root:No ipcontroller-client.json, waiting...
WARNING:root:No ipcontroller-client.json, waiting...
WARNING:root:No ipcontroller-client.json, waiting...
WARNING:root:No ipcontroller-client.json, waiting...
WARNING:root:No ipcontroller-client.json, waiting...
WARNING:root:No ipcontroller-client.json, waiting...
WARNING:root:No ipcontroller-client.json, waiting...
WARNING:root:No ipcontroller-client.json, waiting...
INFO:root:waiting for 1 engines
INFO:root:0 of 1 running
INFO:root:7 of 1 running
INFO:root:Starting cluster: ipcluster start --daemonize --quiet --
↪profile=testing_pbs --n=3 --ipython-dir="/var/folders/m7/dnr91tjs4gn58_
↪t3k8zp_g000000gn/T/tmp9itx0A"
WARNING:root:No ipcontroller-client.json, waiting...
WARNING:root:No ipcontroller-client.json, waiting...
WARNING:root:No ipcontroller-client.json, waiting...
WARNING:root:No ipcontroller-client.json, waiting...
WARNING:root:No ipcontroller-client.json, waiting...
```

```
WARNING:root:No ipcontroller-client.json, waiting...
WARNING:root:No ipcontroller-client.json, waiting...
WARNING:root:No ipcontroller-client.json, waiting...
INFO:root:waiting for 1 engines
INFO:root:0 of 1 running
INFO:root:3 of 1 running
Simple test connecting to a cluster. ... INFO:root:waiting for 1 engines
INFO:root:7 of 1 running
ok
Test deleting of cluster objects ... ok
Test that starting a running cluster does nothing. ... ok
Test that the PBS_NODEFILE is used if defined ... INFO:root:waiting for 1␣
↪engines
INFO:root:3 of 1 running
INFO:root:waiting for 3 engines
INFO:root:3 of 3 running
INFO:root:Stopping cluster: ipcluster stop --profile=testing_pbs --ipython-
↪dir="/var/folders/m7/dnr91tjs4gn58_t3k8zp_g000000gn/T/tmp9itx0A"
2016-01-05 12:16:55.566 [IPClusterStop] Stopping cluster [pid=17497] with␣
↪[signal=2]
ok
Test timeout (coverage) ... ok
mmfutils.tests.test_parallel.TestCluster.test_views ... DEBUG:root:Importing␣
↪canning map
ok
INFO:root:Stopping cluster: ipcluster stop --profile=testing1 --ipython-dir="/
↪var/folders/m7/dnr91tjs4gn58_t3k8zp_g000000gn/T/tmp9itx0A"
2016-01-05 12:16:56.330 [IPClusterStop] Stopping cluster [pid=17461] with␣
↪[signal=2]
mmfutils.tests.test_performance_blas.Test_BLAS.test_daxpy ... ok
mmfutils.tests.test_performance_blas.Test_BLAS.test_ddot ... ok
mmfutils.tests.test_performance_blas.Test_BLAS.test_dnorm ... ok
mmfutils.tests.test_performance_blas.Test_BLAS.test_zaxpy ... ok
mmfutils.tests.test_performance_blas.Test_BLAS.test_zdotc ... ok
mmfutils.tests.test_performance_blas.Test_BLAS.test_znorm ... ok
mmfutils.tests.test_performance_fft.Test_FFT.test_fft ... ok
mmfutils.tests.test_performance_fft.Test_FFT.test_fftn ... ok
mmfutils.tests.test_performance_fft.Test_FFT_pyfftw.test_fft ... ok
mmfutils.tests.test_performance_fft.Test_FFT_pyfftw.test_fft_pyfftw ... ok
mmfutils.tests.test_performance_fft.Test_FFT_pyfftw.test_fftn ... ok
mmfutils.tests.test_performance_fft.Test_FFT_pyfftw.test_fftn_pyfftw ... ok
mmfutils.tests.test_performance_fft.Test_FFT_pyfftw.test_get_fft_pyfftw ... ok
mmfutils.tests.test_performance_fft.Test_FFT_pyfftw.test_get_fftn_pyfftw ...␣
↪ok
mmfutils.tests.test_performance_threads.TestThreads.test_hook_mkl ... ok
mmfutils.tests.test_performance_threads.TestThreads.test_hooks_fft ... ok
mmfutils.tests.test_performance_threads.TestThreads.test_hooks_numexpr ... ok
mmfutils.tests.test_performance_threads.TestThreads.test_set_threads_fft ...␣
↪ok
mmfutils.tests.test_performance_threads.TestThreads.test_set_threads_mkl ...␣
↪ok
mmfutils.tests.test_performance_threads.TestThreads.test_set_threads_numexpr .
↪.. ok
```

```
Name                              Stmts   Miss  Cover   Missing
---------------------------------------------------------------
mmfutils                              1      0   100%
mmfutils.containers                  85      0   100%
mmfutils.debugging                   47      0   100%
mmfutils.interface                   70      0   100%
mmfutils.math                         0      0   100%
mmfutils.math.differentiate          61      0   100%
mmfutils.math.integrate             193      0   100%
mmfutils.monkeypatches               14      0   100%
mmfutils.optimize                    13      0   100%
mmfutils.parallel                   124      2    98%   15-16
mmfutils.performance                  0      0   100%
mmfutils.performance.blas            58      0   100%
mmfutils.performance.fft             61      0   100%
mmfutils.performance.numexpr         10      0   100%
mmfutils.performance.threads         10      0   100%
---------------------------------------------------------------
TOTAL                               747      2    99%
----------------------------------------------------------------------
Ran 73 tests in 19.302s

OK
```

Complete code coverage information is provided in `build/_coverage/index.html`.

4.1 Releases

We try to keep the repository clean with the following properties:

1. The default branch is stable: i.e. if someone runs `hg clone`, this will pull the latest stable release.
2. Each release has its own named branch so that e.g. `hg up 0.4.6` will get the right thing. Note: this should update to the development branch, *not* the default branch so that any work committed will not pollute the development branch (which would violate the previous point).

To do this, we advocate the following proceedure.

1. **"hg up <version>"**: Make sure this is the correct development branch, not the default branch. (Check by `hg up default` which should take you to the default branch.)
2. **Work**: Do your work, committing as required with messages as shown in the repository with the following keys:
   - `DOC`: Documentation changes.
   - `API`: Changes to the exising API. This could break old code.
   - `EHN`: Enhancement or new functionality. Without an `API` tag, these should not break existing codes.
   - `BLD`: Build system changes (`setup.py`, `requirements.txt` etc.)
   - `TST`: Update tests, code coverage, etc.
   - `BUG`: Address an issue as filed on the issue tracker.
   - `BRN`: Start a new branch (see below).
   - `REL`: Release (see below).

- `WIP`: Work in progress. Do not depend on these! They will be stripped. This is useful when testing things like the rendering of documentation on bitbucket etc. where you need to push an incomplete set of files. Please collapse and strip these eventually when you get things working.
- `CHK`: Checkpoints. These should not be pushed to bitbucket!

3. **"python setup.py test"**: Make sure the tests pass. (`hg com` will do this automatically if you have linked the `.hgrc` file as discussed above.
4. **Update Docs**: Update the documentation if needed. To generate new documentation run:

```
cd doc
sphinx-apidoc -eTE ../mmfutils -o source
rm source/mmfutis.tests.*
```

   Include any changes at the bottom of this file (`doc/README.ipynb`).

   Edit any new files created (titles often need to be added) and check that this looks good with

```
make html
open build/html/index.html
```

   Look especially for errors of the type `WARNING: document isn't included in any toctree`. This indicates that you probably need to add the module to an upper level `.. toctree::`. Also look for `WARNING: toctree contains reference to document u'...' that doesn't have a title: no link will be generated`. This indicates you need to add a title to a new file. For example, when I added the `mmf.math.optimize` module, I needed to update the following:

```
.. doc/source/mmfutils.rst
mmfutils
========

.. toctree::
    ...
    mmfutils.optimize
    ...
```

```
.. doc/source/mmfutils.optimize.rst
mmfutils.optimize
=================

.. automodule:: mmfutils.optimize
    :members:
    :undoc-members:
    :show-inheritance:
```

5. **"hg histedit"**: (or `hg rebase`, or `hg strip` as needed) Clean up the repo before you push. Branches should generally be linear unless there is an exceptional reason to split development.
6. **Release**: First edit `mmfutils/__init__.py` and update the version number by removing the `dev` part of the version number. Commit only this change and then push only the branch you are working on:

```
hg com -m "REL: <version>"
hg push -b .
```

7. Create a pull request on the development fork from your branch to `default` on the release project bitbucket. Review it, fix anything, then accept the PR and close the branch.
8. **Start new branch**: On the same development branch (not `default`), increase the version number in `mmfutils/__init__.py` and add `dev`: i.e.:

```
__version__ = '0.4.7dev'
```

Then create this branch and commit this:

```
hg branch "0.4.7"
hg com -m "BRN: Started branch 0.4.7"
```

9. Update MyPI index.
10. Optional: Update any `setup.py` files that depend on your new features/fixes etc.

CHAPTER 5

Change Log

5.1 REL: 0.4.7

API changes:

- Added `mmfutils.interface.describe_interface()` for inserting interfaces into documentation.
- Added some DVR basis code to `mmfutils.math.bases`.
- Added a diverging colormap and some support in `mmfutils.plot`.
- Added a Wigner Ville distribution computation in `mmfutils.math.wigner`
- Added `mmfutils.optimize.usolve` and `ubrentq` for finding roots with `` `uncertanties <https://pythonhosted.org/uncertainties/>`__ `` support.

Issues:

- Resolve issue #8: Use `` `ipyparallel <https://github.com/ipython/ipyparallel>`__ `` now.
- Resolve issue #9: Use pytest rather than `nose` (which is no longer supported).
- Resolve issue #10: PYFFTW wrappers now support negative `axis` and `axes` arguments.
- Address issue #11: Preliminary version of some DVR basis classes.
- Resolve issue #12: Added solvers with `` `uncertanties <https://pythonhosted.org/uncertainties/>`__ `` support.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

R

S

U

V