Main

April 26, 2021

1 Python for HPC

1.1 Summary of beginner training

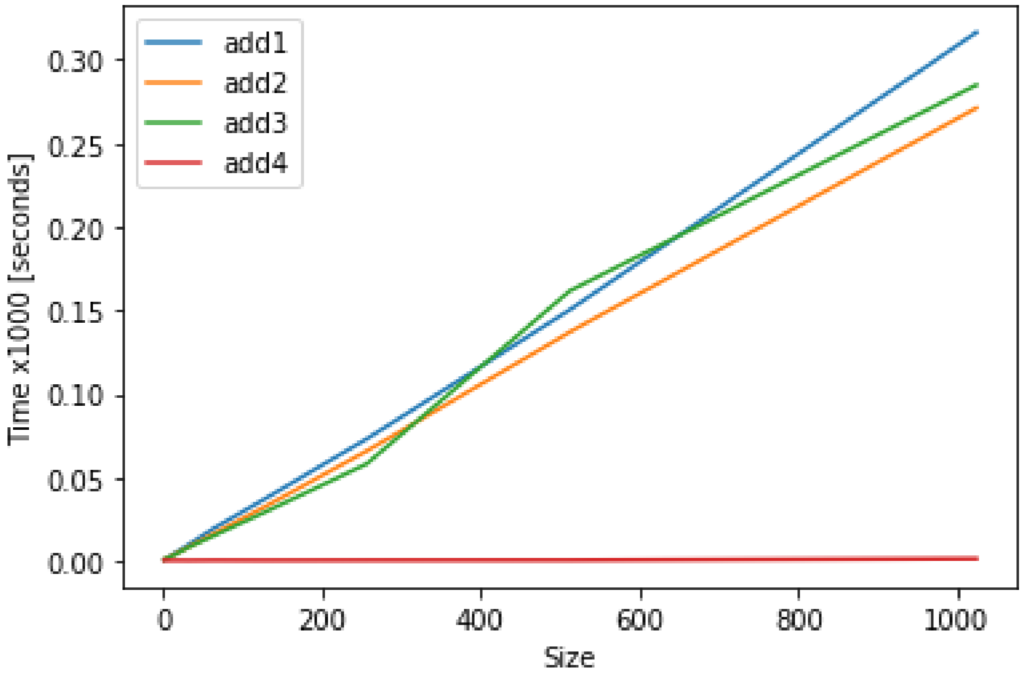
Pure python is SLOW

[1]:
```
def add1(x1, x2):
    "Add with direct memory access and list extension"
    y = []
    for i in range(len(x1)):
        y.append(x1[i] + x2[i])
    return y


def add2(x1, x2):
    "Add with indirect memory access and list extension"
    y = []
    for i1, i2 in zip(x1, x2):
        y.append(i1 + i2)
    return y


def add3(x1, x2):
    "Add with indirect memory access and list generation"
    return [i1 + i2 for i1, i2 in zip(x1, x2)]


def add4(x1, x2):
    "Add with buit-in numpy function"
    return x1 + x2
```

[2]:
```
import numpy as np
from timeit import timeit
from pandas import DataFrame

times = DataFrame(index=[2 ** i for i in range(11)])
```

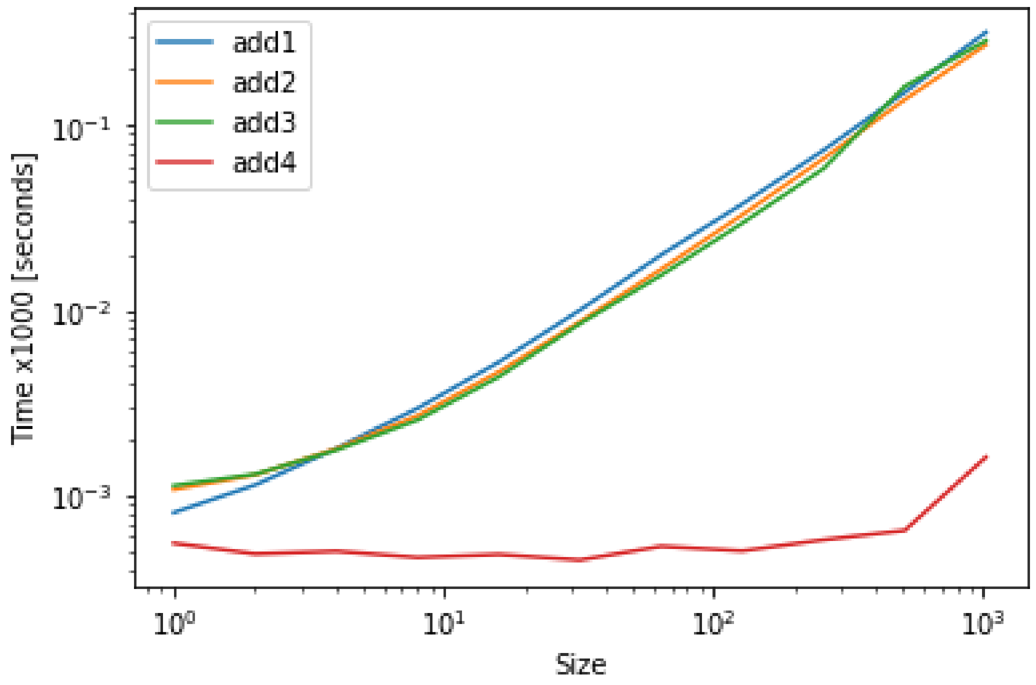
```
for size in times.index:
    x1, x2 = np.random.rand(2, size)
    for add in [
        add1,
        add2,
        add3,
        add4,
    ]:
        times.at[size, add.__name__] = timeit(lambda: add(x1, x2), number=1000)

times.plot(ylabel="Time x1000 [seconds]", xlabel="Size")
```

[2]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fefc8a53110>
```



[3]:
```
times.plot(logx=True, logy=True, ylabel="Time x1000 [seconds]", xlabel="Size")
```

[3]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fefc7925b10>
```



1.2 Laplace operator

Focus of the morning session: **optimization of the laplace operator**

Reference: https://en.wikipedia.org/wiki/Laplace_operator

$$\Delta f = \nabla^2 f = \nabla \cdot \nabla f = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2}$$

$$\Delta f(x,y) \approx \frac{f(x-h,y) + f(x+h,y) + f(x,y-h) + f(x,y+h) - 4f(x,y)}{h^2}$$

[8]:
```
import numpy as np

def laplace(arr):
    assert len(arr.shape)==2
    out = np.zeros_like(arr)
    X,Y = arr.shape
    for x in range(X):
        for y in range(Y):
            out[x,y] = arr[x-1,y] + arr[(x+1)%X,y] + arr[x,y-1] +␣
 ↪arr[x,(y+1)%Y] - 4*arr[x,y]
    return out
```

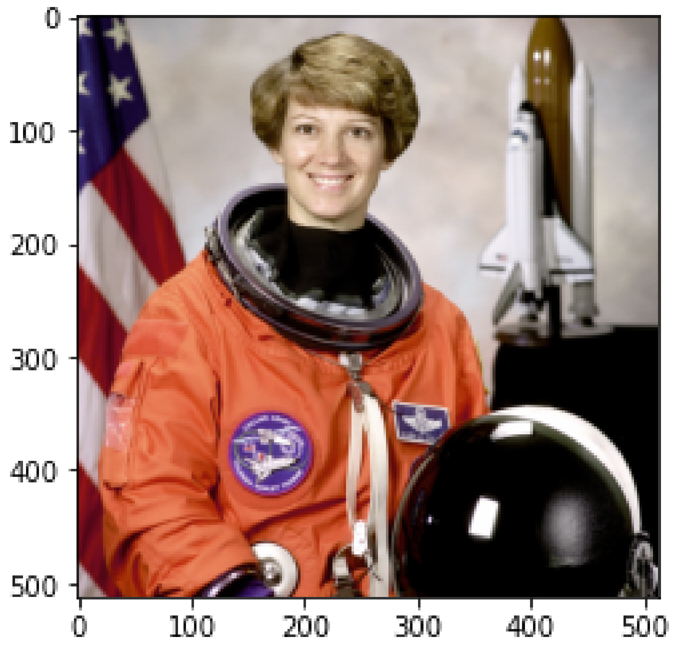
1.2.1 Example for Laplacian operator

[9]:
```
import skimage.data
import skimage.color

image = skimage.data.astronaut()
image.shape
```

[9]:
```
(512, 512, 3)
```

[10]:
```
from matplotlib import pyplot as plt

plt.imshow(image)
```

[10]:
```
<matplotlib.image.AxesImage at 0x2ba0099bc760>
```

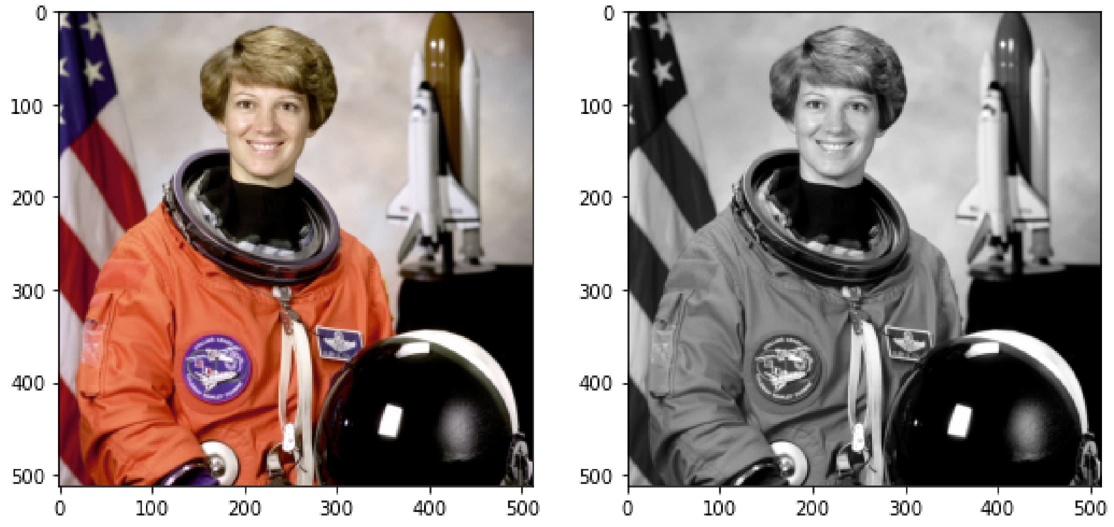


[11]:
```
color = image
image = skimage.color.rgb2gray(image)
image.shape
```

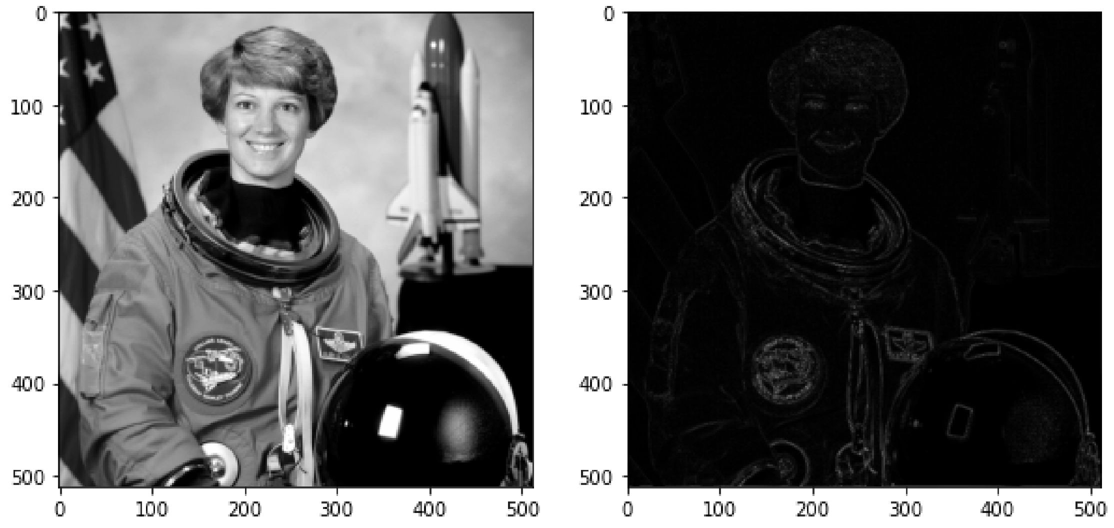
[11]:
```
(512, 512)
```

[12]:
```
from matplotlib import pyplot as plt

def compare(left, right, cmap="gray"):
    """Compares two images, left and right."""
    fig, ax = plt.subplots(1, 2, figsize=(10, 5))
    ax[0].imshow(left, cmap=cmap)
    ax[1].imshow(right, cmap=cmap)
```

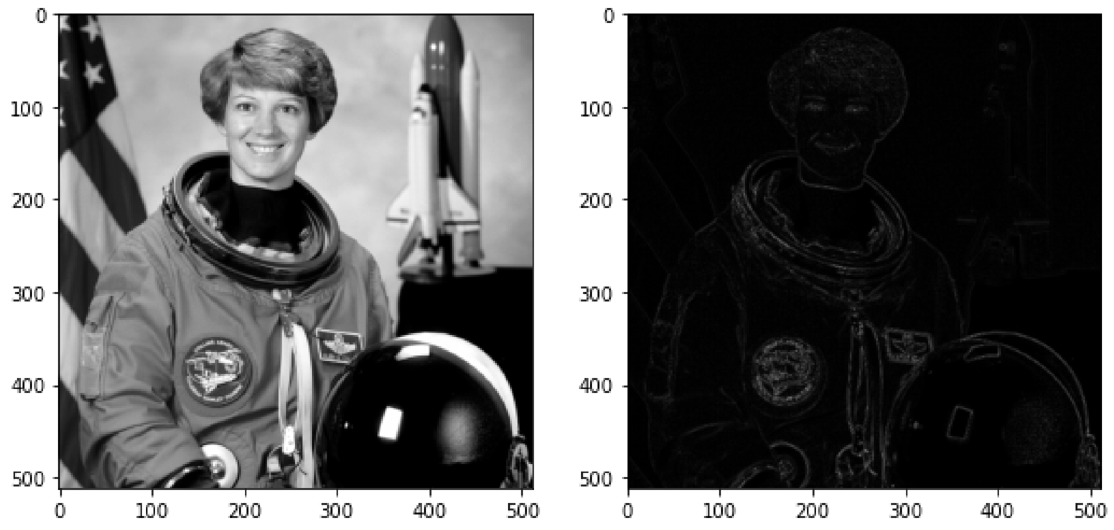
[13]:
```
compare(color, image)
```



[14]:
```
edges = laplace(image)
compare(image, np.abs(edges), cmap="gray")
```



[15]:
```python
from scipy import ndimage

edges = ndimage.laplace(image)
compare(image, np.abs(edges), cmap="gray")
```



[16]:
```python
arr = np.random.rand(10,10)
np.allclose(laplace(arr), ndimage.laplace(arr, mode="wrap"))
```

[16]:
```
True
```

[17]:
```python
help(ndimage.laplace)
```

```
Help on function laplace in module scipy.ndimage.filters:

laplace(input, output=None, mode='reflect', cval=0.0)
    N-D Laplace filter based on approximate second derivatives.

    Parameters
    ----------
    input : array_like
        The input array.
    output : array or dtype, optional
        The array in which to place the output, or the dtype of the
        returned array. By default an array of the same dtype as input
        will be created.
    mode : str or sequence, optional
        The `mode` parameter determines how the input array is extended
        when the filter overlaps a border. By passing a sequence of modes
        with length equal to the number of dimensions of the input array,
        different modes can be specified along each axis. Default value is
        'reflect'. The valid values and their behavior is as follows:

        'reflect' (`d c b a | a b c d | d c b a`)
            The input is extended by reflecting about the edge of the last
            pixel.

        'constant' (`k k k k | a b c d | k k k k`)
            The input is extended by filling all values beyond the edge with
            the same constant value, defined by the `cval` parameter.

        'nearest' (`a a a a | a b c d | d d d d`)
            The input is extended by replicating the last pixel.

        'mirror' (`d c b | a b c d | c b a`)
            The input is extended by reflecting about the center of the last
            pixel.

        'wrap' (`a b c d | a b c d | a b c d`)
            The input is extended by wrapping around to the opposite edge.
    cval : scalar, optional
        Value to fill past edges of input if `mode` is 'constant'. Default
        is 0.0.

    Examples
    --------
    >>> from scipy import ndimage, misc
    >>> import matplotlib.pyplot as plt
    >>> fig = plt.figure()
    >>> plt.gray()  # show the filtered result in grayscale
    >>> ax1 = fig.add_subplot(121)  # left side
    >>> ax2 = fig.add_subplot(122)  # right side
    >>> ascent = misc.ascent()
    >>> result = ndimage.laplace(ascent)
    >>> ax1.imshow(ascent)
    >>> ax2.imshow(result)
    >>> plt.show()
```

See https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.laplace.html

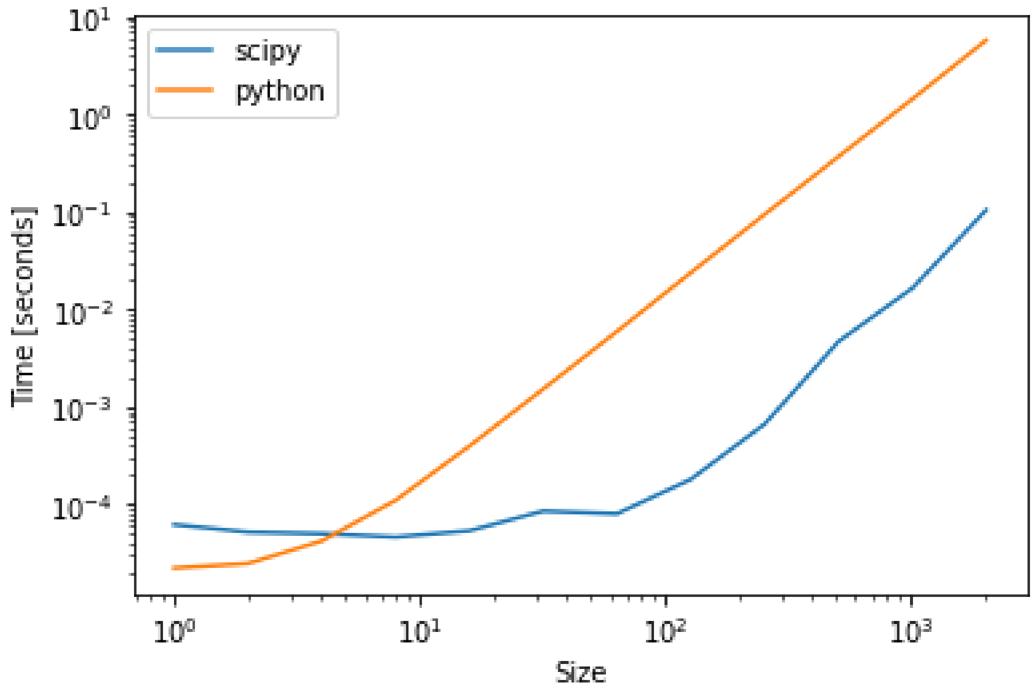
[18]:
```
arr = np.random.rand(10,10,10)
ndimage.laplace(arr).shape
```

[18]:
```
(10, 10, 10)
```

1.2.2 Performance

[19]:
```python
from timeit import timeit
from pandas import DataFrame

times = DataFrame(index=[2 ** i for i in range(12)])

for size in times.index:
    x = np.random.rand(size, size)
    for name,fnc in [
        ("scipy", ndimage.laplace),
        ("python", laplace),
    ]:
        times.at[size, name] = timeit(lambda: fnc(x), number=1)

times.plot(logx=True, logy=True, ylabel="Time [seconds]", xlabel="Size")
```

[19]:
```
<AxesSubplot:xlabel='Size', ylabel='Time [seconds]'>
```



1.2.3 Exercise 1

Implement other modes of `laplace`: i.e. reflect, constant, nearest, mirror.

See https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.laplace.html

```
[20]: def laplace(arr):
          assert len(arr.shape)==2
          out = np.zeros_like(arr)
          X,Y = arr.shape
          for x in range(X):
              for y in range(Y):
                  out[x,y] = arr[x-1,y] + arr[(x+1)%X,y] + arr[x,y-1] +␣
       ↪arr[x,(y+1)%Y] - 4*arr[x,y]
          return out
```

```
[21]: def laplace_modes(arr, mode = "wrap"):
          shape = arr.shape
          out = np.zeros_like(arr)
          ext = np.pad(arr, ((1,1),)*len(shape), mode=mode)
          X,Y = ext.shape
          for x in range(1,X-1):
              for y in range(1,Y-1):
                  out[x-1,y-1] = ext[x-1,y] + ext[x+1,y] + ext[x,y-1] + ext[x,y+1] -␣
       ↪4*ext[x,y]
          return out
```

```
[22]: mode="wrap"
      arr = np.random.rand(10,10)
      np.allclose(laplace_modes(arr, mode=mode), ndimage.laplace(arr, mode=mode))
```

```
[22]: False
```

1.2.4 Exercise 2

Make `laplace` valid for a n-dimensional array

1.2.5 Exercise 3

Write `laplace` using numpy.roll

```
[24]: help(np.roll)
```

```
Help on function roll in module numpy:

roll(a, shift, axis=None)
    Roll array elements along a given axis.

    Elements that roll beyond the last position are re-introduced at
    the first.

    Parameters
```

```
----------
a : array_like
    Input array.
shift : int or tuple of ints
    The number of places by which elements are shifted.  If a tuple,
    then `axis` must be a tuple of the same size, and each of the
    given axes is shifted by the corresponding number.  If an int
    while `axis` is a tuple of ints, then the same value is used for
    all given axes.
axis : int or tuple of ints, optional
    Axis or axes along which elements are shifted.  By default, the
    array is flattened before shifting, after which the original
    shape is restored.

Returns
-------
res : ndarray
    Output array, with the same shape as `a`.

See Also
--------
rollaxis : Roll the specified axis backwards, until it lies in a
           given position.

Notes
-----
.. versionadded:: 1.12.0

Supports rolling over multiple dimensions simultaneously.

Examples
--------
>>> x = np.arange(10)
>>> np.roll(x, 2)
array([8, 9, 0, 1, 2, 3, 4, 5, 6, 7])
>>> np.roll(x, -2)
array([2, 3, 4, 5, 6, 7, 8, 9, 0, 1])

>>> x2 = np.reshape(x, (2,5))
>>> x2
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> np.roll(x2, 1)
array([[9, 0, 1, 2, 3],
       [4, 5, 6, 7, 8]])
>>> np.roll(x2, -1)
array([[1, 2, 3, 4, 5],
       [6, 7, 8, 9, 0]])
```

```
>>> np.roll(x2, 1, axis=0)
array([[5, 6, 7, 8, 9],
       [0, 1, 2, 3, 4]])
>>> np.roll(x2, -1, axis=0)
array([[5, 6, 7, 8, 9],
       [0, 1, 2, 3, 4]])
>>> np.roll(x2, 1, axis=1)
array([[4, 0, 1, 2, 3],
       [9, 5, 6, 7, 8]])
>>> np.roll(x2, -1, axis=1)
array([[1, 2, 3, 4, 0],
       [6, 7, 8, 9, 5]])
```

[25]:
```python
def laplace_modes(arr, mode = "wrap"):
    shape = arr.shape
    out = np.zeros_like(arr)
    ext = np.pad(arr, ((1,1),)*len(shape), mode=mode)
    X,Y = ext.shape
    for x in range(1,X-1):
        for y in range(1,Y-1):
            out[x-1,y-1] = ext[x-1,y] + ext[x+1,y] + ext[x,y-1] + ext[x,y+1] -␣
↪4*ext[x,y]
    return out
```

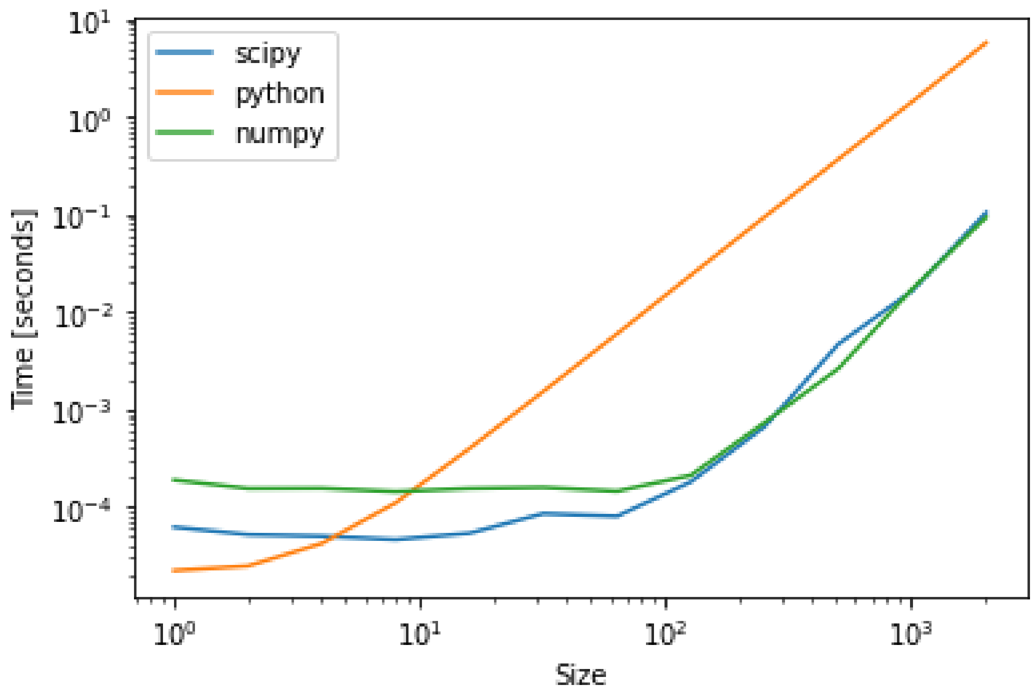
[26]:
```python
def numpy_laplace(arr, mode = "wrap"):

    shape = arr.shape
    n = len(shape)
    ext = np.pad(arr, ((1,1),)*len(shape), mode=mode)

    out = -2*n*arr

    inner = (slice(1,-1),)*len(shape)

    for i in range(n):
        out += np.roll(ext, 1, axis=i)[inner]
        out += np.roll(ext, -1, axis=i)[inner]

    return out
```

[27]:
```python
arr = np.random.rand(10,10,10)
np.allclose(numpy_laplace(arr), ndimage.laplace(arr, mode="wrap"))
```

[27]:
```
True
```

[28]:
```
for size in times.index:
    x = np.random.rand(size, size)
    times.at[size, "numpy"] = timeit(lambda: numpy_laplace(x), number=4)/4

times.plot(logx=True, logy=True, ylabel="Time [seconds]", xlabel="Size")
```

[28]:
```
<AxesSubplot:xlabel='Size', ylabel='Time [seconds]'>
```



[29]:
```
import cProfile

size=2000
x = np.random.rand(size, size)
cProfile.run('numpy_laplace(x)', sort="tottime")
```

```
         131 function calls (124 primitive calls) in 0.078 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.034    0.034    0.078    0.078 <ipython-
input-26-bf7b3186e9a1>:1(numpy_laplace)
        4    0.033    0.008    0.033    0.008 numeric.py:1113(roll)
        1    0.010    0.010    0.010    0.010 arraypad.py:86(_pad_simple)
        1    0.000    0.000    0.078    0.078 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 {built-in method numpy.empty}
     12/5    0.000    0.000    0.044    0.009 {built-in method
numpy.core._multiarray_umath.implement_array_function}
        2    0.000    0.000    0.000    0.000 arraypad.py:384(_set_wrap_both)
        4    0.000    0.000    0.000    0.000
numeric.py:1308(normalize_axis_tuple)
        1    0.000    0.000    0.010    0.010 arraypad.py:532(pad)
        1    0.000    0.000    0.078    0.078 {built-in method builtins.exec}
        8    0.000    0.000    0.000    0.000 {built-in method numpy.array}
        1    0.000    0.000    0.000    0.000
stride_tricks.py:114(_broadcast_to)
        4    0.000    0.000    0.033    0.008 <__array_function__
internals>:2(roll)
        1    0.000    0.000    0.000    0.000 {method 'reduce' of 'numpy.ufunc'
objects}
        1    0.000    0.000    0.000    0.000 arraypad.py:457(_as_pairs)
        4    0.000    0.000    0.000    0.000 <__array_function__
internals>:2(empty_like)
        4    0.000    0.000    0.000    0.000 numeric.py:1358(<listcomp>)
        2    0.000    0.000    0.000    0.000 arraypad.py:58(_view_roi)
        4    0.000    0.000    0.000    0.000 {built-in method
numpy.core._multiarray_umath.normalize_axis_index}
        1    0.000    0.000    0.010    0.010 <__array_function__
internals>:2(pad)
        4    0.000    0.000    0.000    0.000 _asarray.py:86(asanyarray)
        1    0.000    0.000    0.000    0.000 fromnumeric.py:52(_wrapfunc)
        8    0.000    0.000    0.000    0.000 arraypad.py:33(_slice_at_axis)
        1    0.000    0.000    0.000    0.000 _methods.py:41(_amin)
        2    0.000    0.000    0.000    0.000 _asarray.py:14(asarray)
        1    0.000    0.000    0.000    0.000 {method 'astype' of
'numpy.ndarray' objects}
        1    0.000    0.000    0.000    0.000 <__array_function__
internals>:2(broadcast_to)
        4    0.000    0.000    0.000    0.000 {built-in method builtins.min}
        1    0.000    0.000    0.000    0.000 function_base.py:244(iterable)
        1    0.000    0.000    0.000    0.000 {method 'min' of 'numpy.ndarray'
objects}
        4    0.000    0.000    0.000    0.000 numeric.py:1194(<dictcomp>)
        1    0.000    0.000    0.000    0.000 <__array_function__
internals>:2(round_)
        1    0.000    0.000    0.000    0.000 fromnumeric.py:3168(around)
        3    0.000    0.000    0.000    0.000 arraypad.py:120(<genexpr>)
        1    0.000    0.000    0.000    0.000 stride_tricks.py:141(broadcast_to)
        4    0.000    0.000    0.000    0.000 numeric.py:1109(_roll_dispatcher)
        1    0.000    0.000    0.000    0.000 fromnumeric.py:3628(round_)
        4    0.000    0.000    0.000    0.000 {built-in method _operator.index}
        1    0.000    0.000    0.000    0.000 {method 'round' of 'numpy.ndarray'
objects}
```

```
        3    0.000    0.000    0.000    0.000 {built-in method builtins.len}
        4    0.000    0.000    0.000    0.000 multiarray.py:75(empty_like)
        1    0.000    0.000    0.000    0.000 <__array_function__
internals>:2(around)
        4    0.000    0.000    0.000    0.000 {method 'items' of 'dict' objects}
        3    0.000    0.000    0.000    0.000 arraypad.py:109(<genexpr>)
        1    0.000    0.000    0.000    0.000 {method 'tolist' of
'numpy.ndarray' objects}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.any}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.getattr}
        1    0.000    0.000    0.000    0.000
stride_tricks.py:24(_maybe_view_as_subclass)
        3    0.000    0.000    0.000    0.000 stride_tricks.py:119(<genexpr>)
        1    0.000    0.000    0.000    0.000 {method 'disable' of
'_lsprof.Profiler' objects}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.iter}
        2    0.000    0.000    0.000    0.000
fromnumeric.py:3164(_around_dispatcher)
        1    0.000    0.000    0.000    0.000 arraypad.py:524(_pad_dispatcher)
        1    0.000    0.000    0.000    0.000
stride_tricks.py:137(_broadcast_to_dispatcher)
        1    0.000    0.000    0.000    0.000 {built-in method
builtins.callable}
```

1.2.6 Decorators (extra)

[30]:
```python
from time import time

def timer(fnc):
    "Decorator"

    def decorated(*args, **kwargs):
        start = time()
        out = fnc(*args, **kwargs)
        stop = time()
        print("Elapsed time", stop-start, "s")
        return out

    return decorated
```

[31]:
```python
tprint = timer(print)
tprint
```

[31]:
```
<function __main__.timer.<locals>.decorated(*args, **kwargs)>
```

1.2.7 Dynamic arguments (extra)

[49]:
```
def fnc(*args, **kwargs):
    print("args", args)
    print("kwargs", kwargs)
```

[50]:
```
fnc(1,2,3,c=2,b=3)
```

```
args (1, 2, 3)
kwargs {'c': 2, 'b': 3}
```

[47]:
```
def fnc(a,b,c):
    print(a,b,c)

fnc(1,2,3)
fnc(1,c=2,b=3)
```

```
1 2 3
1 3 2
```

1.2.8 Numba

Documentation: https://numba.readthedocs.io/en/stable/index.html

[32]:
```
from numba import jit
```

[33]:
```
def laplace(arr):
    assert len(arr.shape)==2
    out = np.zeros_like(arr)
    X,Y = arr.shape
    for x in range(X):
        for y in range(Y):
            out[x,y] = arr[x-1,y] + arr[(x+1)%X,y] + arr[x,y-1] +␣
 ↪arr[x,(y+1)%Y] - 4*arr[x,y]
    return out
```

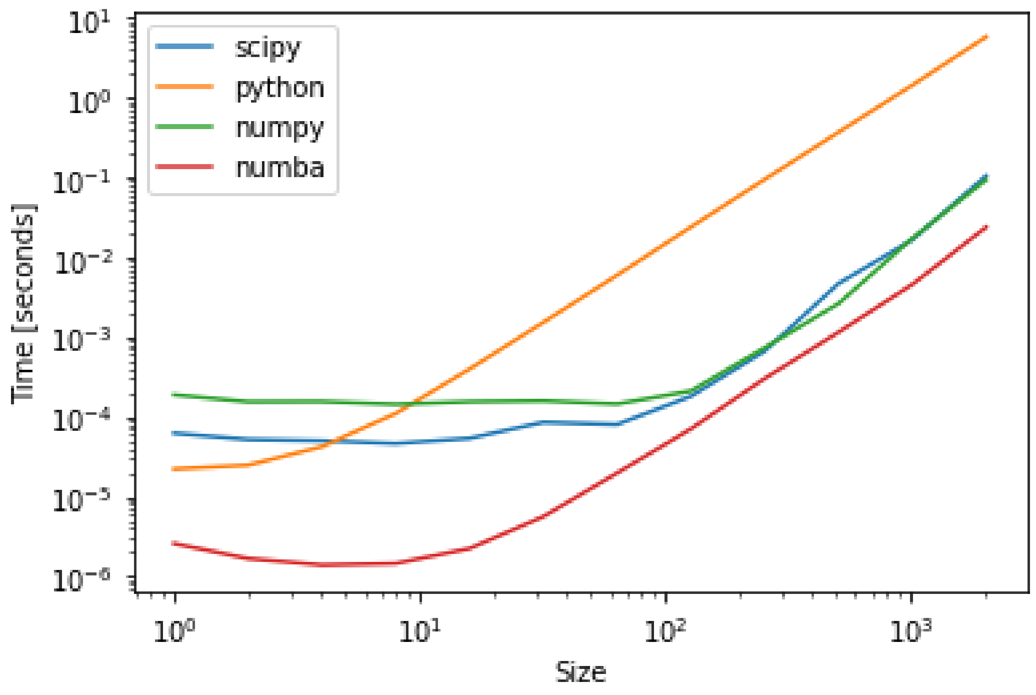
[34]:
```
numba_laplace = jit(laplace)
```

[35]:
```
arr = np.random.rand(10,10)
np.allclose(numba_laplace(arr), ndimage.laplace(arr, mode="wrap"))
```

[35]:
```
True
```

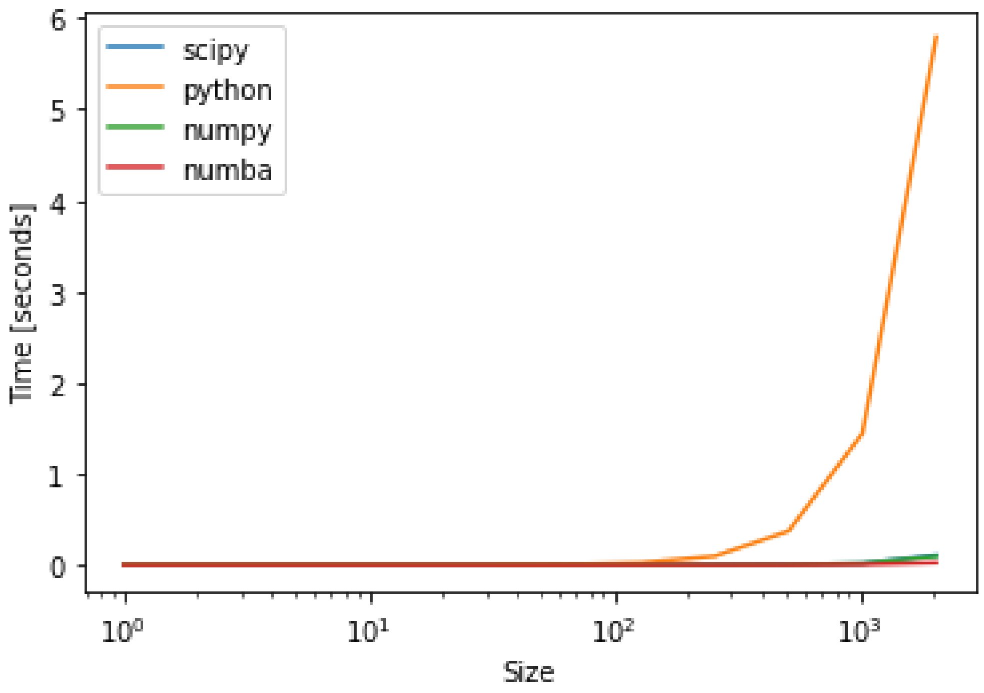
[36]:
```
for size in times.index:
    x = np.random.rand(size, size)
    times.at[size, "numba"] = timeit(lambda: numba_laplace(x), number=4)/4
times.plot(logx=True, logy=True, ylabel="Time [seconds]", xlabel="Size")
```

```
[36]: <AxesSubplot:xlabel='Size', ylabel='Time [seconds]'>
```



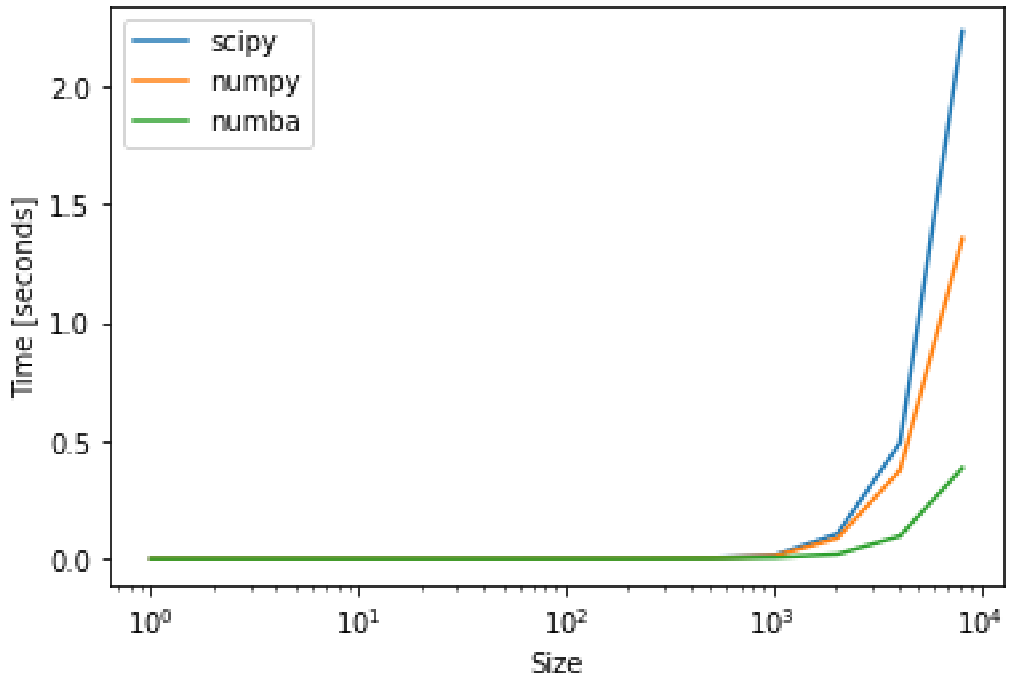
```
[37]: times.plot(logx=True, ylabel="Time [seconds]", xlabel="Size")
```

```
[37]: <AxesSubplot:xlabel='Size', ylabel='Time [seconds]'>
```



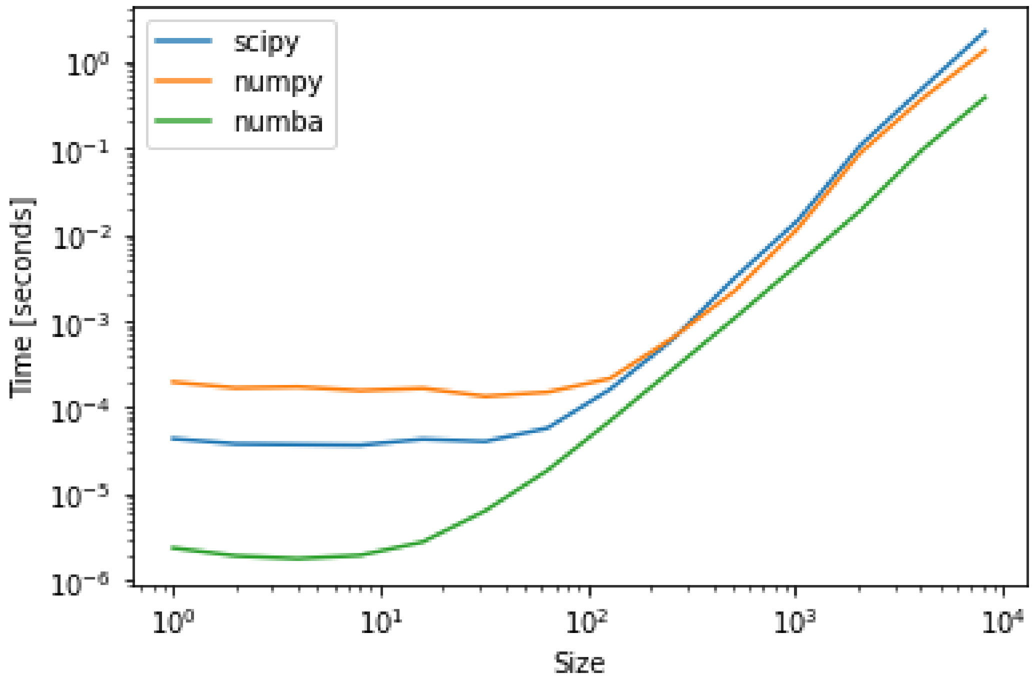
```
[38]: times = DataFrame(index=[2 ** i for i in range(14)])

      for size in times.index:
          x = np.random.rand(size, size)
          for name,fnc in [
              ("scipy", ndimage.laplace),
              ("numpy", numpy_laplace),
              ("numba", numba_laplace),
          ]:
              times.at[size, name] = timeit(lambda: fnc(x), number=4)/4

      times.plot(logx=True, ylabel="Time [seconds]", xlabel="Size")
```

```
[38]: <AxesSubplot:xlabel='Size', ylabel='Time [seconds]'>
```



```
[39]: times.plot(logx=True, logy=True, ylabel="Time [seconds]", xlabel="Size")
```

```
[39]: <AxesSubplot:xlabel='Size', ylabel='Time [seconds]'>
```



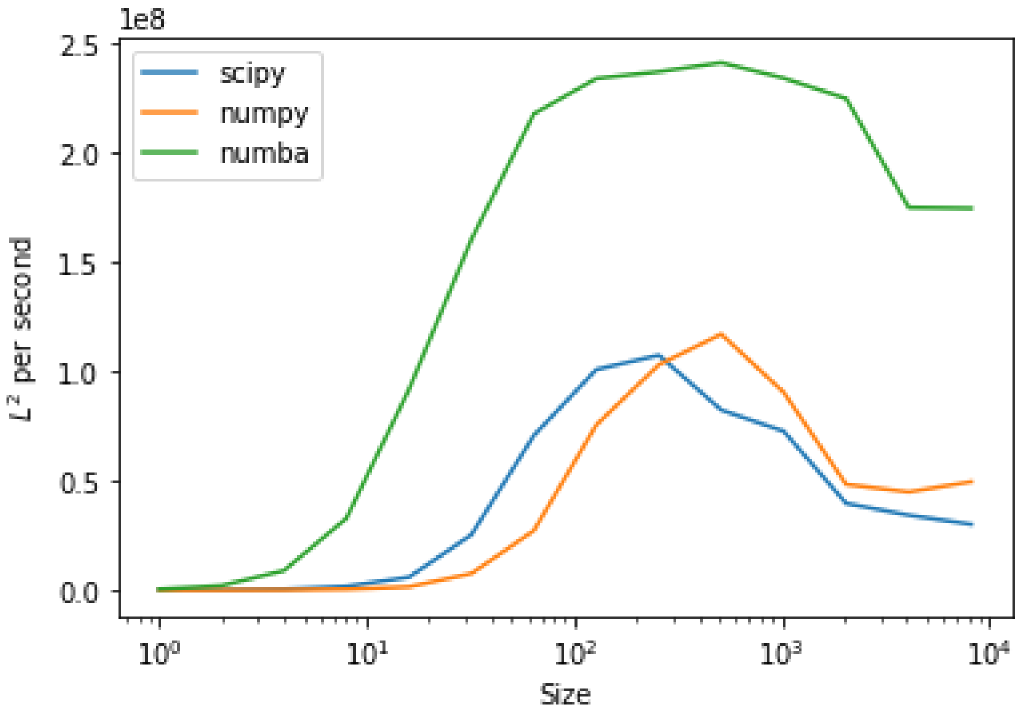
$$t = c + bx^c$$

$c =$ overheads

$b =$ performance

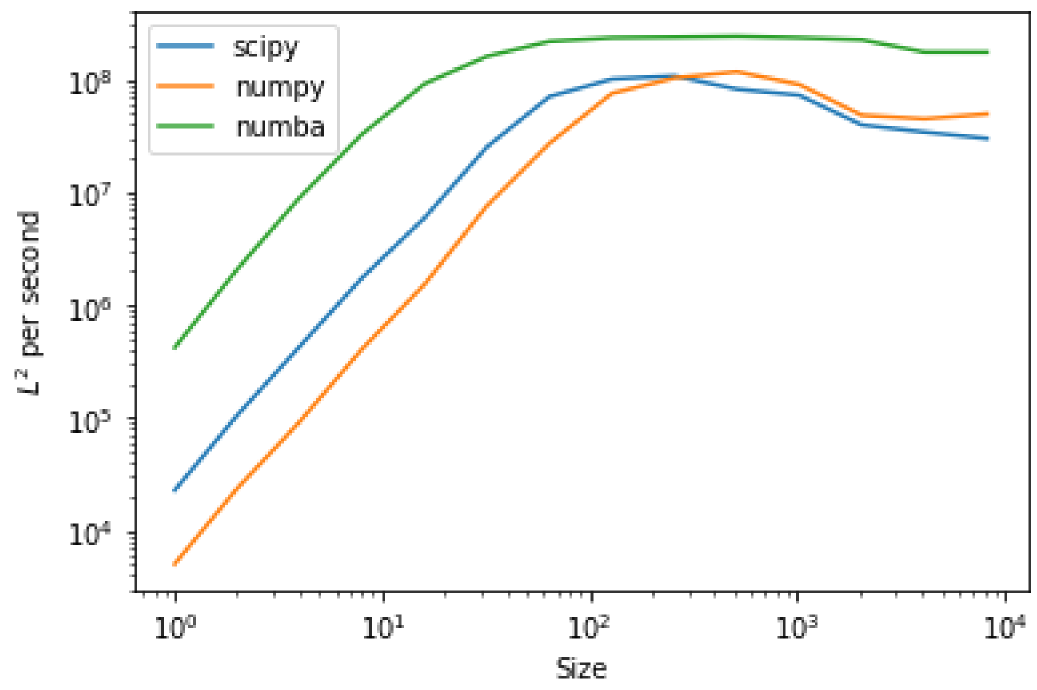
```
[40]: perf = times.apply(lambda x: np.array(times.index)**2 / x)
      perf.plot(logx=True, ylabel="$L^2$ per second", xlabel="Size")
```

```
[40]: <AxesSubplot:xlabel='Size', ylabel='$L^2$ per second'>
```



```
[41]: perf.plot(logx=True, logy=True, ylabel="$L^2$ per second", xlabel="Size")
```

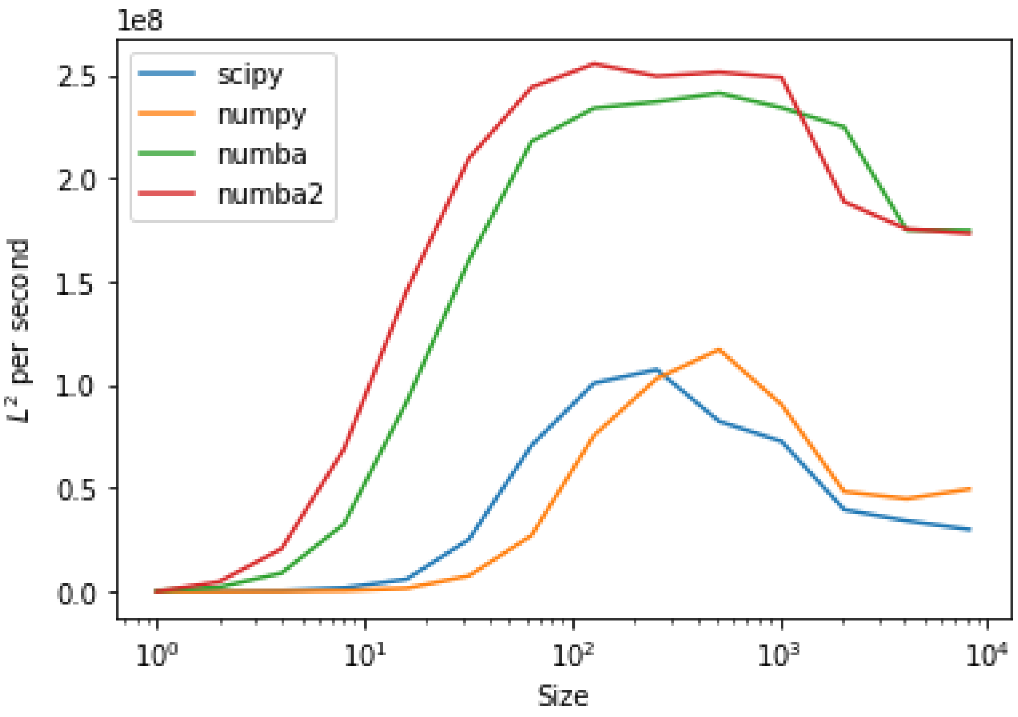
```
[41]: <AxesSubplot:xlabel='Size', ylabel='$L^2$ per second'>
```



```
[51]: numba2_laplace = jit(laplace, nopython=True)
```

```
[52]: for size in times.index:
          x = np.random.rand(size, size)
          times.at[size, "numba2"] = timeit(lambda: numba2_laplace(x), number=20)/20

      perf = times.apply(lambda x: np.array(times.index)**2 / x)
      perf.plot(logx=True, ylabel="$L^2$ per second", xlabel="Size")
```

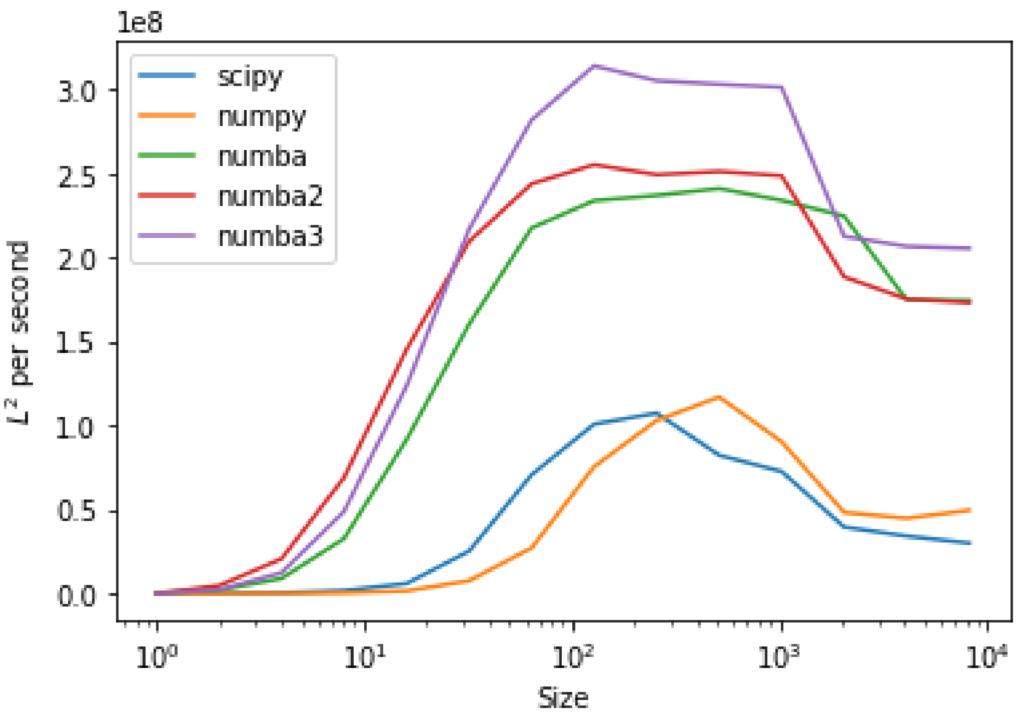
```
[52]: <AxesSubplot:xlabel='Size', ylabel='$L^2$ per second'>
```



```
[53]: numba3_laplace = jit(laplace, nopython=True, fastmath=True)
```

```
[54]: for size in times.index:
          x = np.random.rand(size, size)
          times.at[size, "numba3"] = timeit(lambda: numba3_laplace(x), number=4)/4

      perf = times.apply(lambda x: np.array(times.index)**2 / x)
      perf.plot(logx=True, ylabel="$L^2$ per second", xlabel="Size")
```

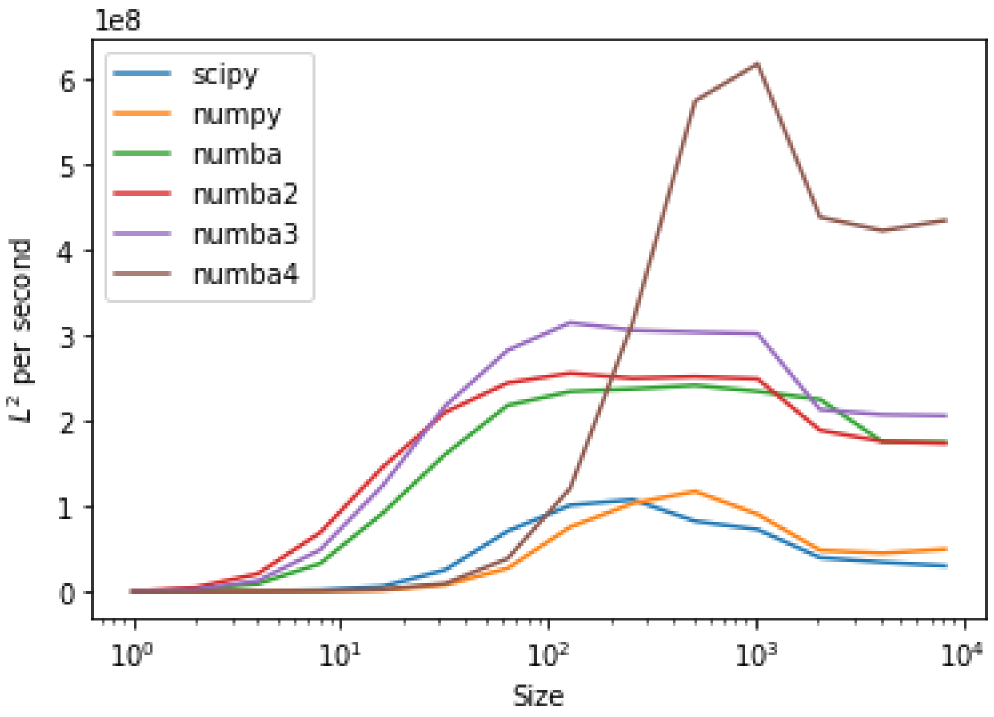
```
[54]: <AxesSubplot:xlabel='Size', ylabel='$L^2$ per second'>
```



```
[57]: from numba import prange

      @jit(nopython=True, fastmath=True, parallel=True)
      def numba4_laplace(arr):
          assert len(arr.shape)==2
          out = np.zeros_like(arr)
          X,Y = arr.shape
          for x in prange(X):
              for y in prange(Y):
                  out[x,y] = arr[x-1,y] + arr[(x+1)%X,y] + arr[x,y-1] +␣
       ↪arr[x,(y+1)%Y] - 4*arr[x,y]
          return out
```

```
[60]: for size in times.index:
          x = np.random.rand(size, size)
          times.at[size, "numba4"] = timeit(lambda: numba4_laplace(x), number=4)/4

      perf = times.apply(lambda x: np.array(times.index)**2 / x)
      perf.plot(logx=True, ylabel="$L^2$ per second", xlabel="Size")
```

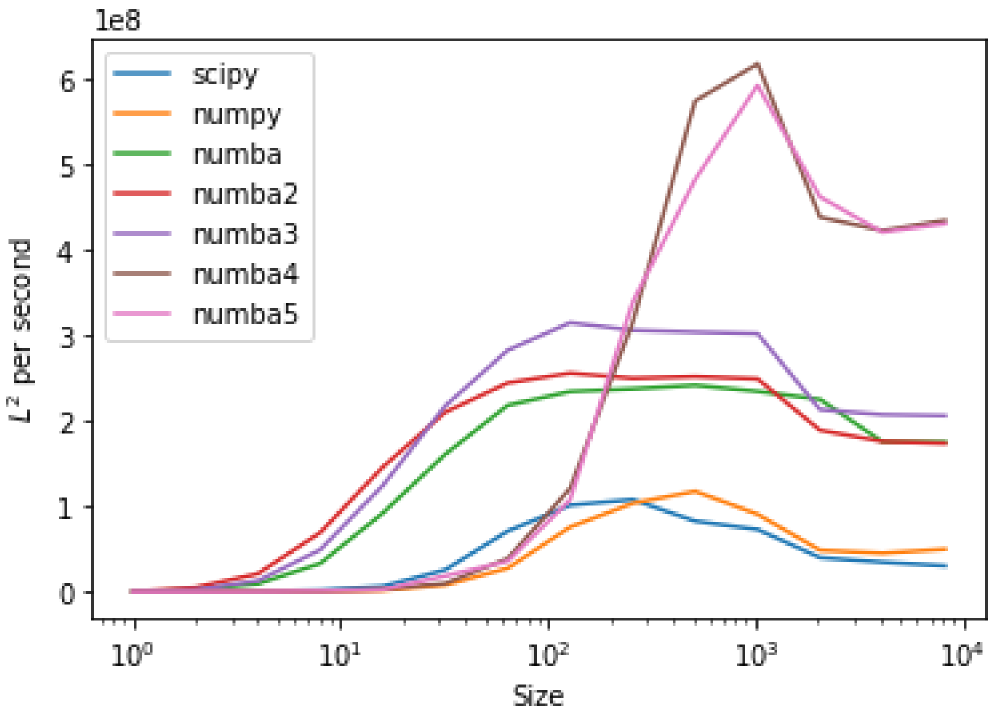
```
[60]: <AxesSubplot:xlabel='Size', ylabel='$L^2$ per second'>
```



[61]:
```
from numba import prange

@jit(nopython=True, fastmath=True, parallel=True)
def numba5_laplace(arr):
    assert len(arr.shape)==2
    out = np.zeros_like(arr)
    X,Y = arr.shape
    for x in prange(X):
        for y in range(Y):
            out[x,y] = arr[x-1,y] + arr[(x+1)%X,y] + arr[x,y-1] +␣
 ↪arr[x,(y+1)%Y] - 4*arr[x,y]
    return out
```

[62]:
```
for size in times.index:
    x = np.random.rand(size, size)
    times.at[size, "numba5"] = timeit(lambda: numba5_laplace(x), number=4)/4

perf = times.apply(lambda x: np.array(times.index)**2 / x)
perf.plot(logx=True, ylabel="$L^2$ per second", xlabel="Size")
```

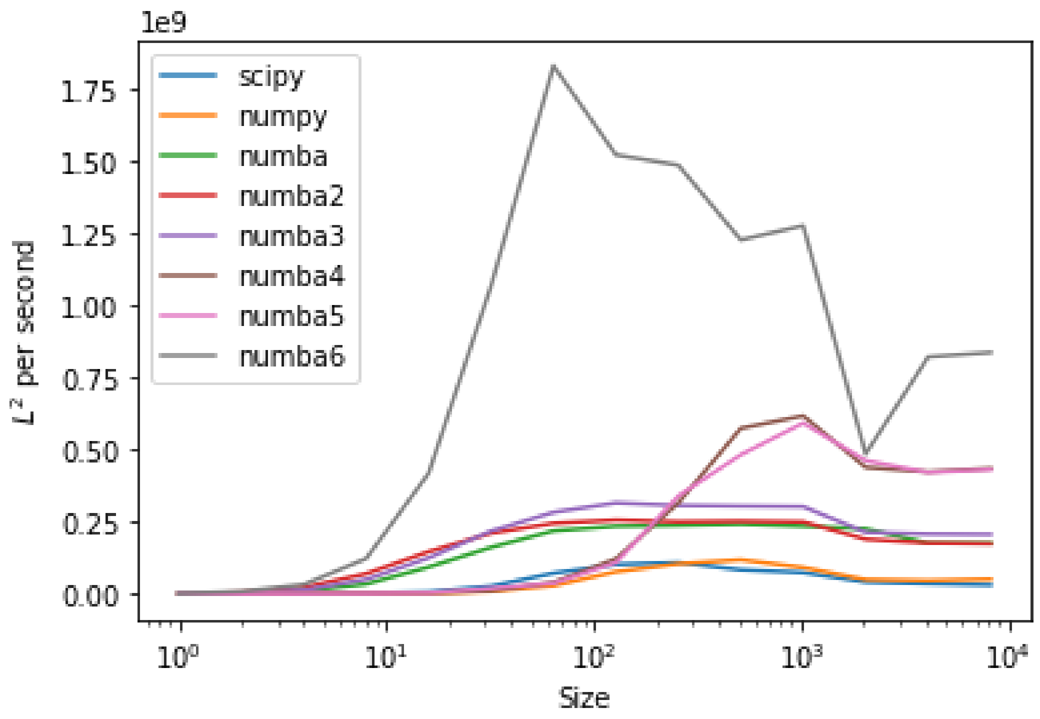
[62]:
```
<AxesSubplot:xlabel='Size', ylabel='$L^2$ per second'>
```



[67]:
```
@jit(nopython=True, fastmath=True)
def numba6_laplace(arr, out):
    """Laplace operator in NumPy for 2D images. Numba accelerated."""
    X,Y = out.shape
    for x in range(1,X-1):
        for y in range(1,Y-1):
            out[x,y] = arr[x-1,y] + arr[x+1,y] + arr[x,y-1] + arr[x,y+1] -␣
 ↪4*arr[x,y]
```

[65]:
```
for size in times.index:
    x = np.random.rand(size, size)
    arr = np.pad(x, ((1,1),)*len(x.shape), mode="wrap")
    out = np.zeros_like(arr)
    times.at[size, "numba6"] = timeit(lambda: numba6_laplace(arr, out),␣
 ↪number=20)/20

perf = times.apply(lambda x: np.array(times.index)**2 / x)
perf.plot(logx=True, ylabel="$L^2$ per second", xlabel="Size")
```

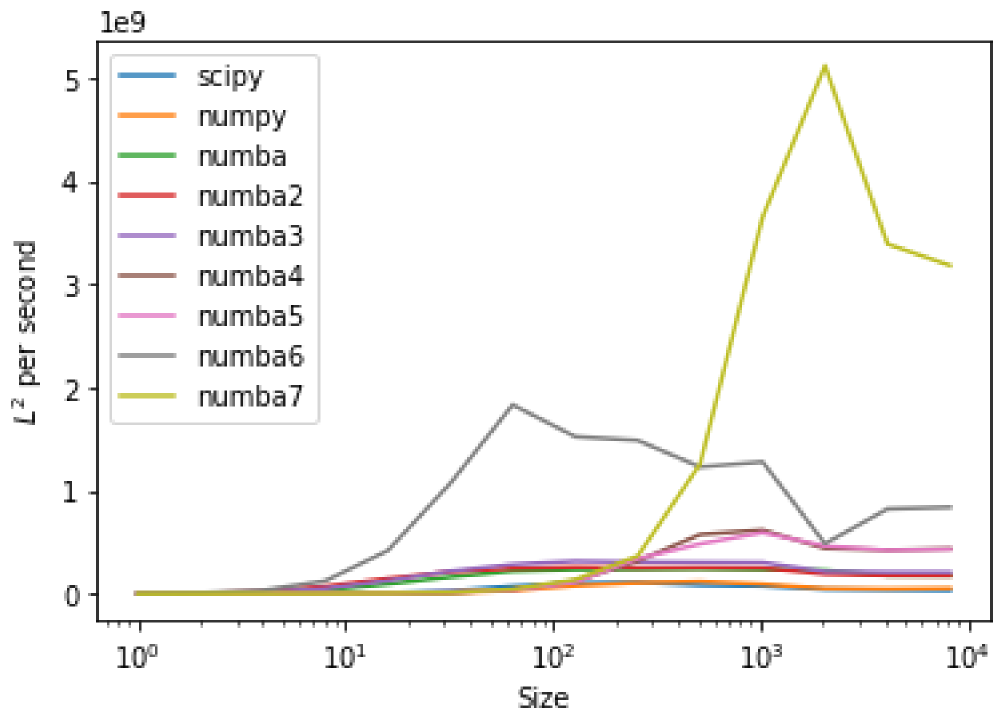
[65]:
```
<AxesSubplot:xlabel='Size', ylabel='$L^2$ per second'>
```



[68]:
```
@jit(nopython=True, fastmath=True, parallel=True)
def numba7_laplace(arr, out):
    """Laplace operator in NumPy for 2D images. Numba accelerated."""
    X,Y = out.shape
    for x in prange(1,X-1):
        for y in prange(1,Y-1):
            out[x,y] = arr[x-1,y] + arr[x+1,y] + arr[x,y-1] + arr[x,y+1] -␣
 ↪4*arr[x,y]
```

[69]:
```
for size in times.index:
    x = np.random.rand(size, size)
    arr = np.pad(x, ((1,1),)*len(x.shape), mode="wrap")
    out = np.zeros_like(arr)
    times.at[size, "numba7"] = timeit(lambda: numba7_laplace(arr, out),␣
 ↪number=20)/20

perf = times.apply(lambda x: np.array(times.index)**2 / x)
perf.plot(logx=True, ylabel="$L^2$ per second", xlabel="Size")
```

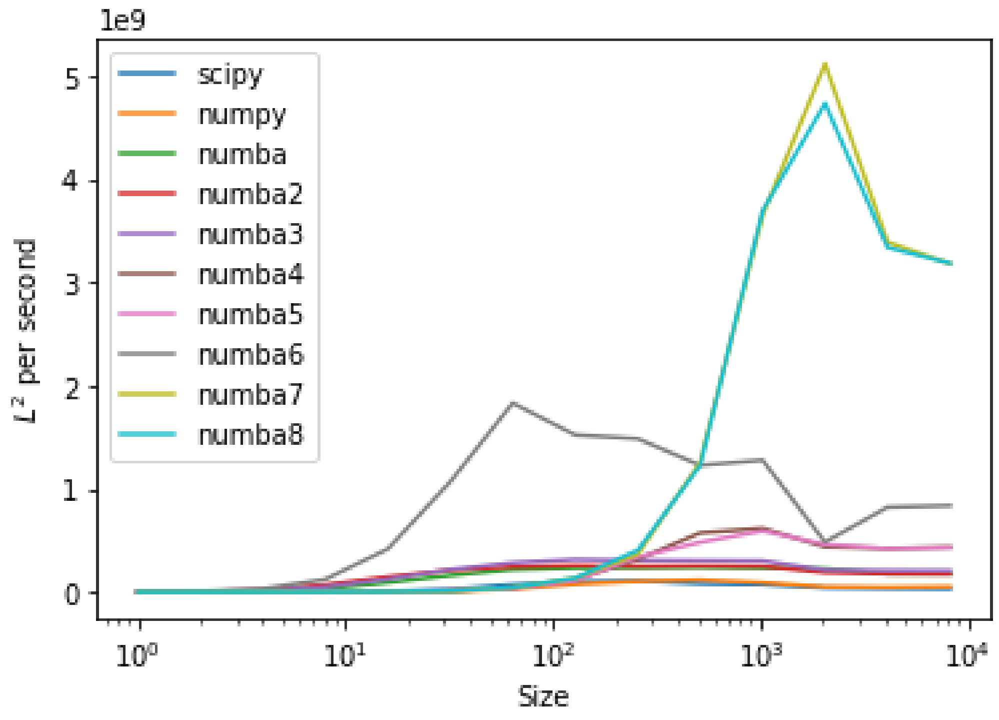
[69]:
```
<AxesSubplot:xlabel='Size', ylabel='$L^2$ per second'>
```



```
[70]: @jit(nopython=True, fastmath=True, parallel=True)
      def numba8_laplace(arr, out):
          """Laplace operator in NumPy for 2D images. Numba accelerated."""
          X,Y = out.shape
          for x in prange(1,X-1):
              for y in range(1,Y-1):
                  out[x,y] = arr[x-1,y] + arr[x+1,y] + arr[x,y-1] + arr[x,y+1] -␣
       ↪4*arr[x,y]
```

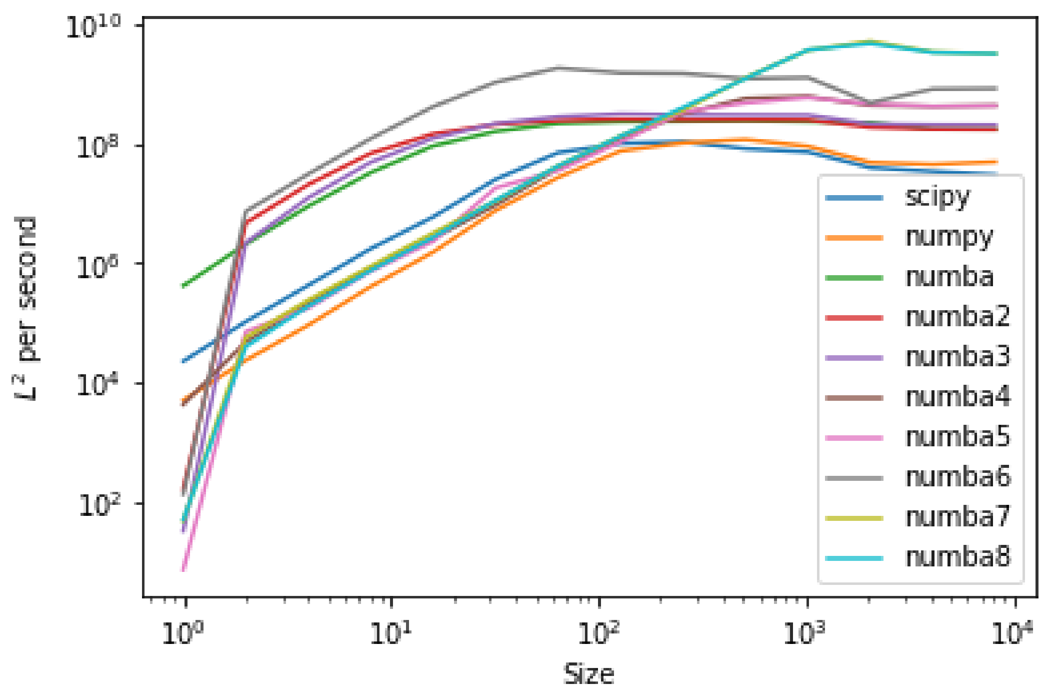
```
[71]: for size in times.index:
          x = np.random.rand(size, size)
          arr = np.pad(x, ((1,1),)*len(x.shape), mode="wrap")
          out = np.zeros_like(arr)
          times.at[size, "numba8"] = timeit(lambda: numba8_laplace(arr, out),␣
       ↪number=20)/20

      perf = times.apply(lambda x: np.array(times.index)**2 / x)
      perf.plot(logx=True, ylabel="$L^2$ per second", xlabel="Size")
```

```
[71]: <AxesSubplot:xlabel='Size', ylabel='$L^2$ per second'>
```



```
[72]: perf.plot(logx=True, logy=True, ylabel="$L^2$ per second", xlabel="Size")
```

```
[72]: <AxesSubplot:xlabel='Size', ylabel='$L^2$ per second'>
```



1.2.9 cppyy

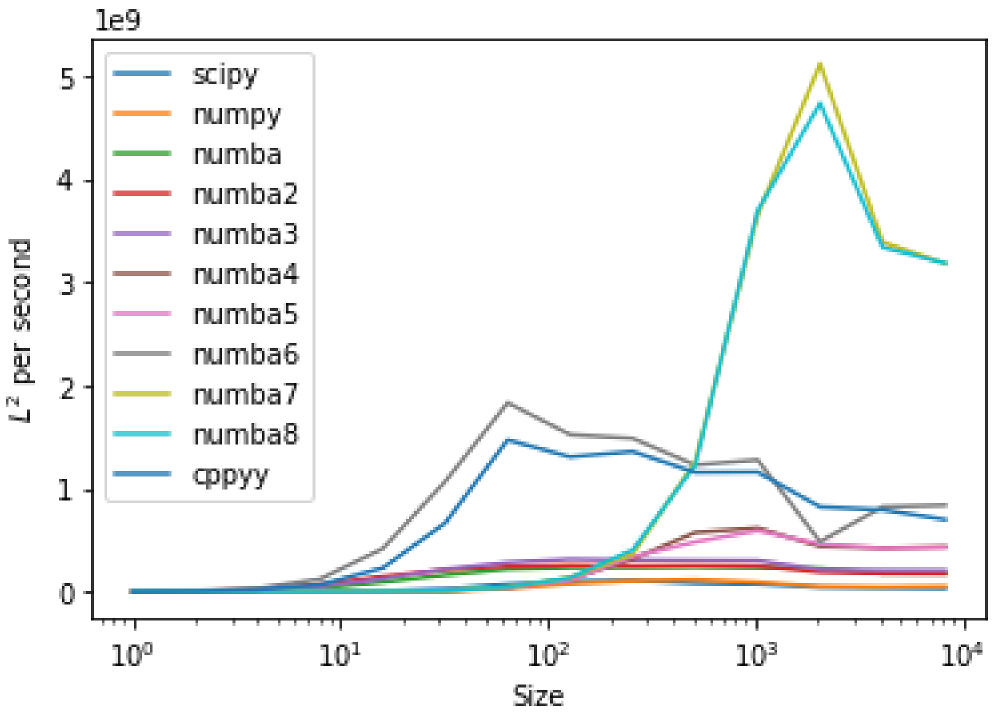
Documentation: https://cppyy.readthedocs.io/en/latest/

[80]:
```
import cppyy

cppyy.cppdef("""
void laplace(double *arr, double *out, int X, int Y) {

  for(int x=1; x<X-1; x++)
    for(int y=1; y<Y-1; y++)
      out[x*Y+y] = arr[(x-1)*Y+y] + arr[(x+1)*Y+y] + arr[x*Y+y-1] +␣
↪arr[x*Y+y+1] - 4*arr[x*Y+y];

}
""")
```

[80]:
```
True
```

[86]:
```
for size in times.index:
    x = np.random.rand(size, size)
    arr = np.pad(x, ((1,1),)*len(x.shape), mode="wrap")
    out = np.zeros_like(arr)
    X, Y = arr.shape
    times.at[size, "cppyy"] = timeit(lambda: cppyy.gbl.laplace(arr, out, X, Y),␣
↪number=20)/20

perf = times.apply(lambda x: np.array(times.index)**2 / x)
perf.plot(logx=True, ylabel="$L^2$ per second", xlabel="Size")
```

[86]:
```
<AxesSubplot:xlabel='Size', ylabel='$L^2$ per second'>
```

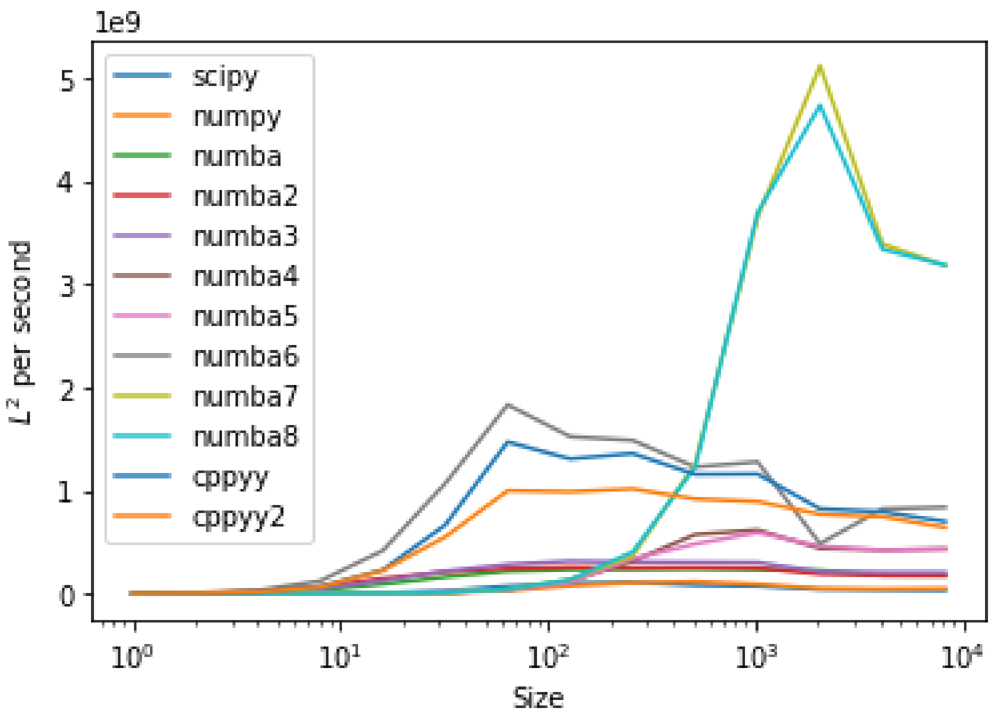


```
[87]: cppyy.c_include("cppyy/laplace.h")
      cppyy.load_library("cppyy/liblaplace.so")
```

```
[87]: True
```

```
[91]: for size in times.index:
          x = np.random.rand(size, size)
          arr = np.pad(x, ((1,1),)*len(x.shape), mode="wrap")
          out = np.zeros_like(arr)
          X, Y = arr.shape
          times.at[size, "cppyy2"] = timeit(lambda: cppyy.gbl.laplace2(arr, out, X,␣
       ↪Y), number=20)/20

      perf = times.apply(lambda x: np.array(times.index)**2 / x)
      perf.plot(logx=True, ylabel="$L^2$ per second", xlabel="Size")
```

```
[91]: <AxesSubplot:xlabel='Size', ylabel='$L^2$ per second'>
```



[104]:
```
cppyy.c_include("cppyy/laplace2.h")
cppyy.load_library("cppyy/liblaplace2.so")
```

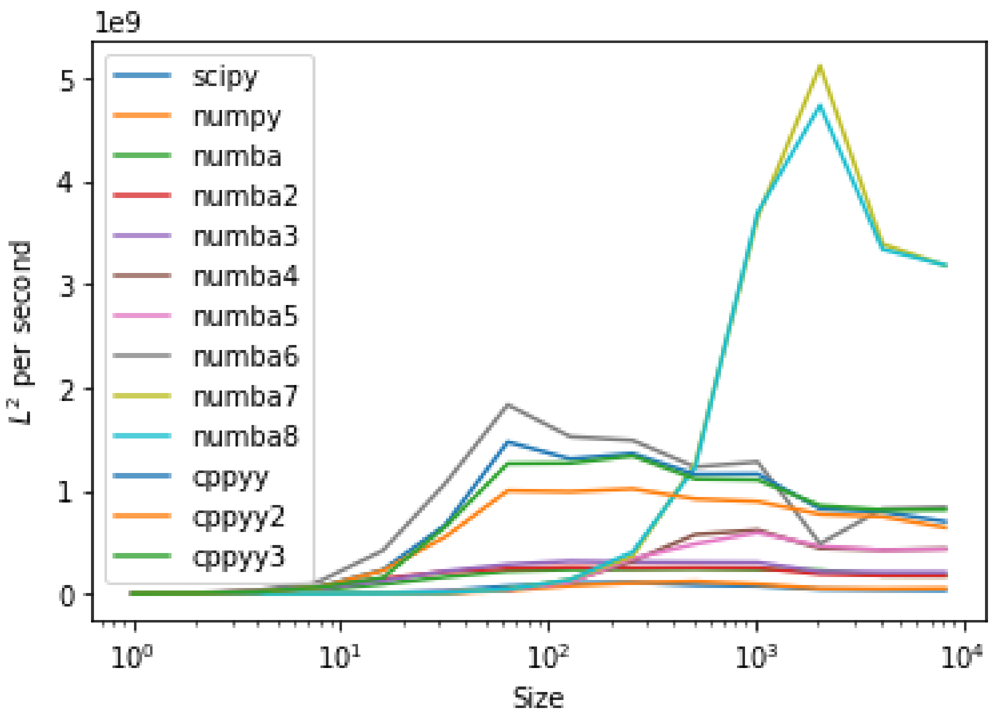
[104]:
```
True
```

[105]:
```
cppyy.gbl.laplace3
```

[105]:
```
<cppyy.CPPOverload at 0x2ba050694f00>
```

[106]:
```
for size in times.index:
    x = np.random.rand(size, size)
    arr = np.pad(x, ((1,1),)*len(x.shape), mode="wrap")
    out = np.zeros_like(arr)
    X, Y = arr.shape
    times.at[size, "cppyy3"] = timeit(lambda: cppyy.gbl.laplace3(arr, out, X,␣
 ↪Y), number=20)/20

perf = times.apply(lambda x: np.array(times.index)**2 / x)
perf.plot(logx=True, ylabel="$L^2$ per second", xlabel="Size")
```

[106]:
```
<AxesSubplot:xlabel='Size', ylabel='$L^2$ per second'>
```



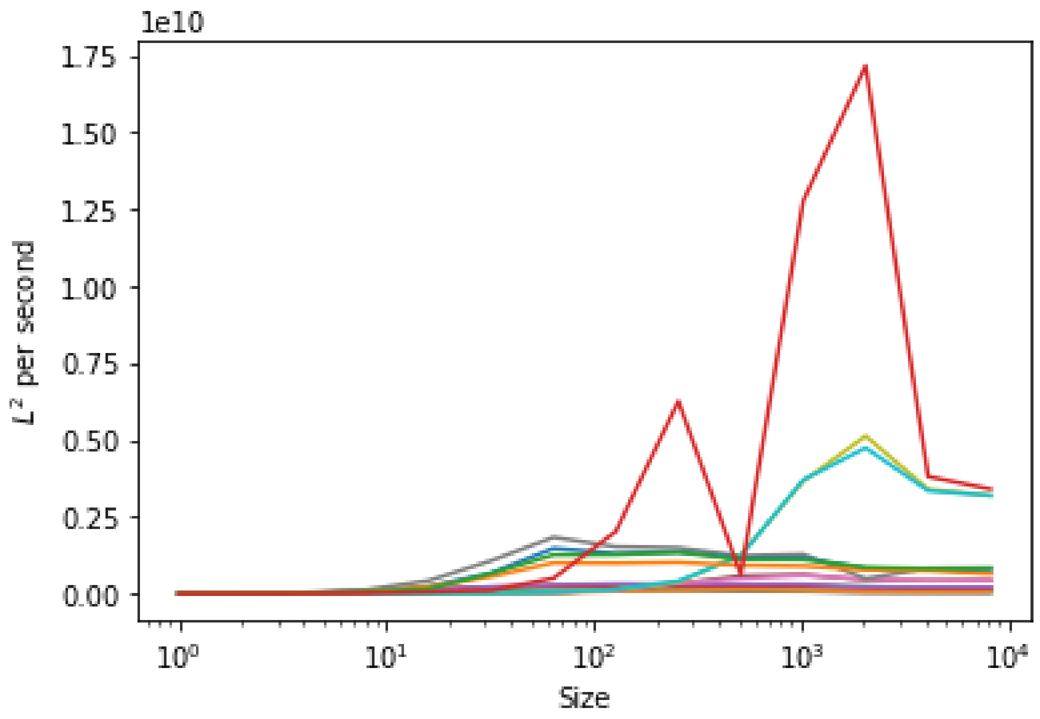
[107]:
```
cppyy.c_include("cppyy/laplace3.h")
cppyy.load_library("cppyy/liblaplace3.so")
```

[107]:
```
True
```

[120]:
```
import os
os.environ["OMP_NUM_THREADS"] = "20"
```

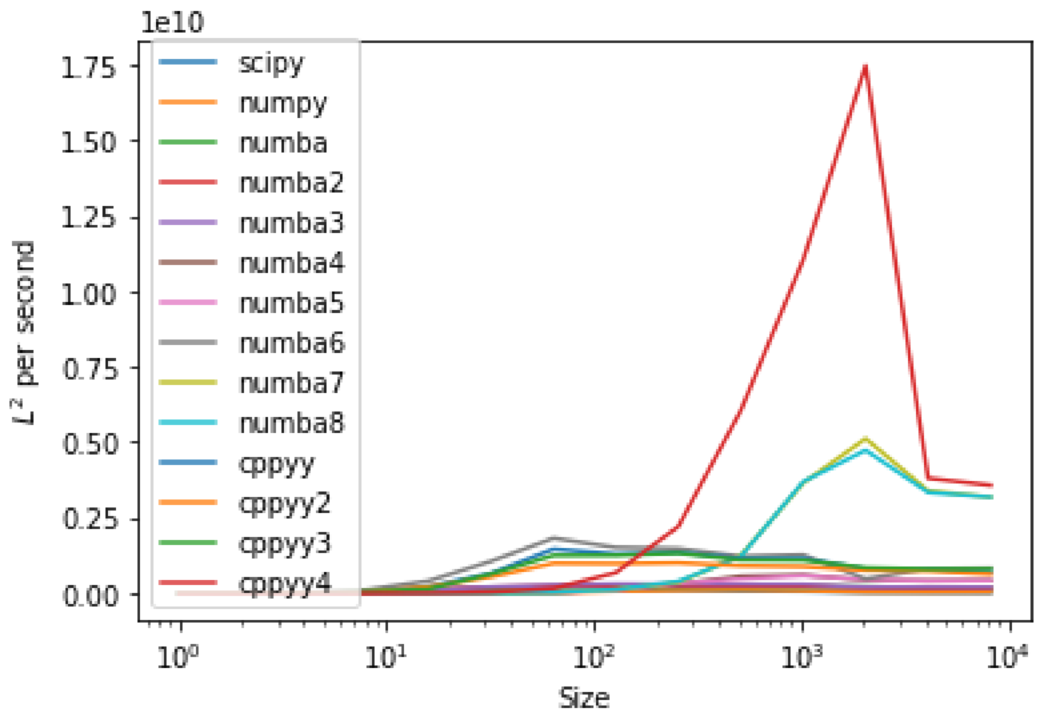
[121]:
```
import sys

for size in times.index:
    x = np.random.rand(size, size)
    arr = np.pad(x, ((1,1),)*len(x.shape), mode="wrap")
    out = np.zeros_like(arr)
    X, Y = arr.shape
    times.at[size, "cppyy4"] = timeit(lambda: cppyy.gbl.laplace4(arr, out, X,␣
 ↪Y), number=20)/20

perf = times.apply(lambda x: np.array(times.index)**2 / x)
perf.plot(logx=True, ylabel="$L^2$ per second", xlabel="Size", legend=False)
```

[121]:
```
<AxesSubplot:xlabel='Size', ylabel='$L^2$ per second'>
```



```
[112]: perf.plot(logx=True, ylabel="$L^2$ per second", xlabel="Size", legend=True)
       plt.legend(loc='lower left')
```

```
[112]: <matplotlib.legend.Legend at 0x2ba0535db9a0>
```



```
[ ]:
```

1.2.10 OpenBLAS

URL: https://www.openblas.net/

`module show OpenBLAS`

```
[101]: #cppyy.include("cblas.h")
       cppyy.include("/nvme/h/buildsets/eb210126_cyclone/software/OpenBLAS/0.3.
        ↪12-GCC-10.2.0/include/cblas.h")
```

```
[101]: True
```

```
[96]: #cppyy.include("libopenblas")
      cppyy.load_library("/nvme/h/buildsets/eb210126_cyclone/software/OpenBLAS/0.3.
       ↪12-GCC-10.2.0/lib/libopenblas.so")
```

```
[96]: True
```

```
[122]: cppyy.gbl.cblas_dsum
```

```
[122]: <cppyy.CPPOverload at 0x2ba050702100>
```

[127]:
```
arr = np.random.rand(10)
arr.sum()
```

[127]:
```
5.998074835967339
```

[128]:
```
cppyy.gbl.cblas_dsum(10, arr, 1)
```

[128]:
```
5.998074835967338
```

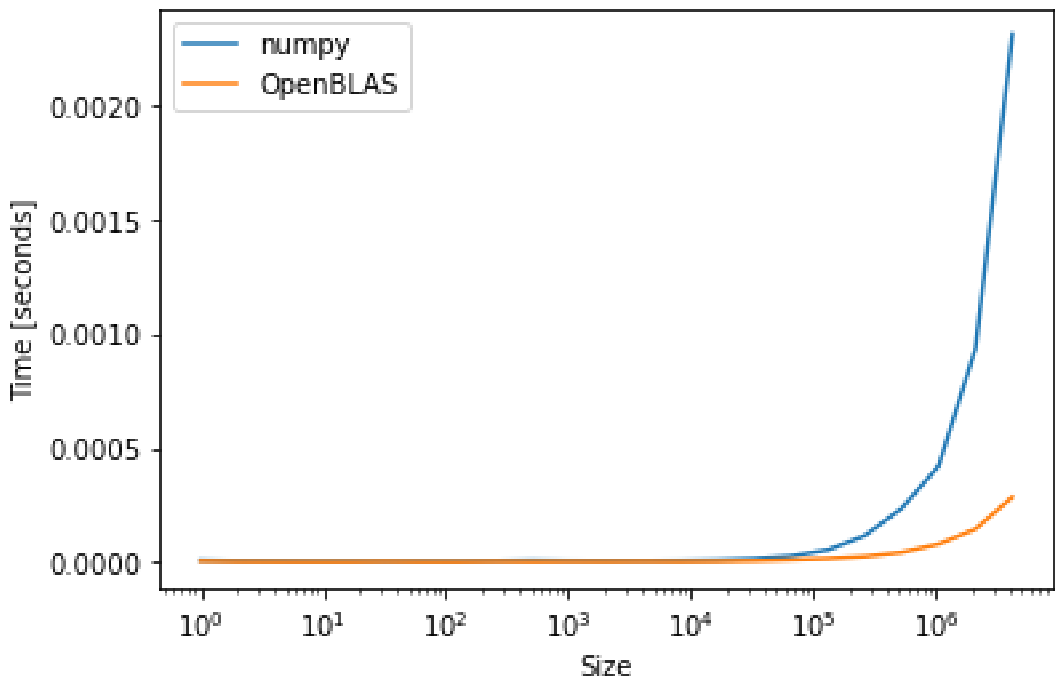
[137]:
```
def openblas_sum(arr):
    arr = arr.reshape((-1))
    size = arr.shape[0]
    out = cppyy.gbl.cblas_dsum(size, arr, 1)
    return out
```

[131]:
```
openblas_sum(arr)
```

[131]:
```
5.998074835967338
```

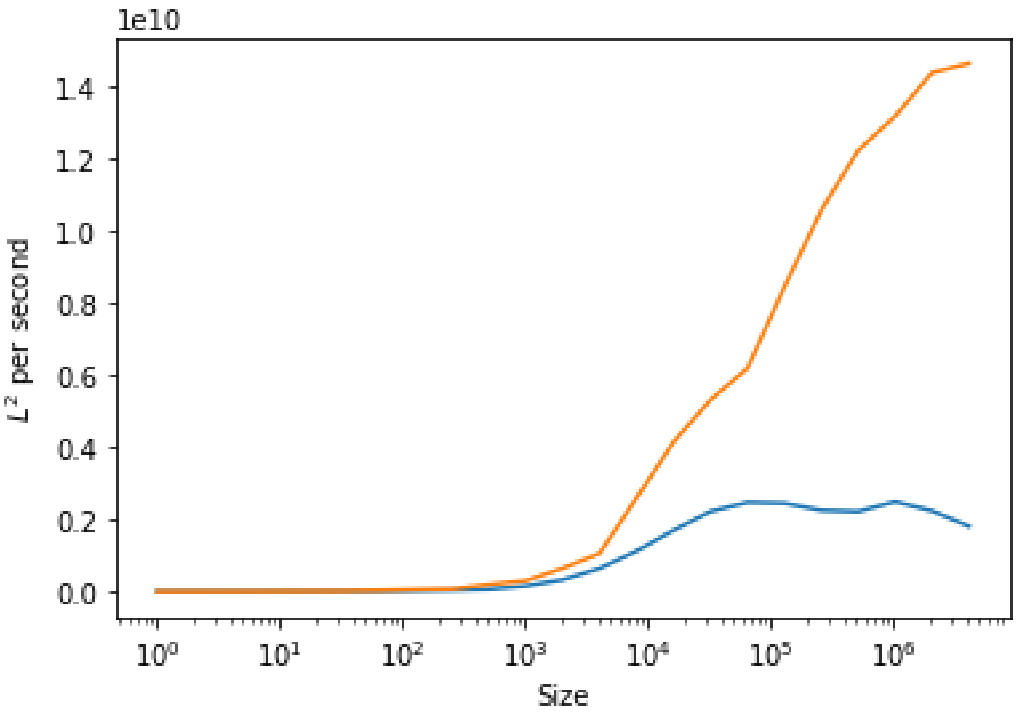
[143]:
```
times = DataFrame(index=[2 ** i for i in range(23)])

for size in times.index:
    x = np.random.rand(size)
    for name,fnc in [
        ("numpy", np.sum),
        ("OpenBLAS", openblas_sum),
    ]:
        times.at[size, name] = timeit(lambda: fnc(x), number=4)/4

times.plot(logx=True, ylabel="Time [seconds]", xlabel="Size")
```

[143]:
```
<AxesSubplot:xlabel='Size', ylabel='Time [seconds]'>
```



```
[144]: perf = times.apply(lambda x: np.array(times.index) / x)
       perf.plot(logx=True, ylabel="$L^2$ per second", xlabel="Size", legend=False)
```

```
[144]: <AxesSubplot:xlabel='Size', ylabel='$L^2$ per second'>
```



1.2.11 mpi4py

Documentation: https://mpi4py.readthedocs.io/en/stable/

```
pip install --force-reinstall --no-cache-dir  mpi4py
```