A Numexpr example

Numexpr is a library for the fast execution of array transformation. One can define complex elementwise operations on array and Numexpr will generate efficient code to execute the operations.

```
import numexpr as ne
import numpy as np
```

Numexpr provides fast multithreaded operations on array elements. Let's test it on some large arrays.

```
a = np.random.rand(1000000)
b = np.random.rand(1000000)
```

Componentwise addition is easy.

```
ne.evaluate("a + 1")
ne.evaluate("a + b")
```

```
array([1.3195833 , 0.92546223, 1.68758307, ..., 1.19557921, 1.19559017,
       0.24145174])
```

We can evaluate complex expressions.

```
ne.evaluate('a*b-4.1*a > 2.5*b')
```

```
array([False, False, False, ..., False, False, False])
```

Let's compare the performance with Numpy.

```
%%timeit
a * b - 4.1 * a > 2.5 * b
```

```
7.89 ms ± 91.7 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
%%timeit
ne.evaluate('a*b-4.1*a > 2.5*b')
```

```
995 µs ± 40.5 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Numexpr is a factor 10 faster compared to Numpy, a nice improvement with very little effort. Let us compare some more complex operations.

```
%%timeit
ne.evaluate("sin(a) + arcsinh(a/b)")
```

```
5.4 ms ± 100 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

We can compare it with Numpy.

```
%%timeit
np.sin(a) + np.arcsinh(a / b)
```

```
39.5 ms ± 664 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

By Timo Betcke