numpy自定义数据类型

引言

```
import sys
sys.getsizeof(2)
```

输出为：28

一个在C中占4个字节的整型在python中占据了28个字节！看起来python在处理一个整形，实际上是在对一个对象进行操作。由于python一切皆对象，连int类型和float类型背后都是一个对象，因此直接用python本身作计算密集的工作是得不偿失的。

让python快起来的关键就是适当地引入静态，提前告诉python数据结构和类型。在这一层面上，基于C和fortran的numpy，可以直接对内存进行操作，也提供了很多方便的接口，因此理解numpy是用python作科学计算的基础和关键。任何其他加速python代码的方式，如cython，numba，本质上都是在python里实现静态，也都和numpy有着密不可分的联系。

```
x = np.zeros(1000,dtype=np.int32)
sys.getsizeof(x)
```

输出为：4104

这个内存占用已经和C比较接近了。

np.dtype对象

静态就是在程序运行前确定如何解释一块连续的内存。在`np.array`中可以指定`dtype`参数，比如`x = np.zeros(1000,dtype=np.int32)`，告诉python，数组x是一个由1000个`np.int32`类型的数据构成的数据对象，这类似于C中的数组。当然我们也可以指定一些有不同种元素构成的数据对象，比如创建由1个整形，两个单精度浮点数，一个字符串组成的对象，类似于C中的结构体。这个需求可以通过`np.dtype`对象来实现。

np.dtype对象描述了一块固定大小的内存块是如何被解释的，采用np.dtype创建一个结构体时，需要指定结构体中每一段的名称，数据类型以及数目，每一段都用一个元组包裹起来，称之为字段（field）。比如现在想要创建一个由16个字符，一个2x2 float32构成的结构体：

```
dt = np.dtype([('name',np.unicode_,16),('grades',np.float32,(2,2))])
```

在创建结构体时，可以按照dt指定的方式进行初始化：

```
x = np.array([('shen',[[1,2],[3,4]]),('yang',[[5,6],[7,8]])],dtype=dt)
```

可以像一般的数组一样引用x中的元素，获取某一元素的某一个字段时，可以像字典一样获取该字段的元素：

```
print(x.shape)
print('------')
print(x[0])
print('------')
print(x[0]['name'])
```

输出为：

```
(2,)
------
('shen', [[1., 2.], [3., 4.]])
------
shen
```

生成的一个结构体占据的字节数是16+2*32 = 80：

```
sys.getsizeof(np.array([('shen',[[1,2],[3,4]]) for i in
range(100)],dtype=dt)
```

输出为：8104，剩下的104个字节大概是numpy对象额外占据的一些信息。

现在想要获取这个结构化数组的某一字段时，还必须要像字典一样用中括号去获取元素。我们也可以像获取对象的属性一样用点来获取元素，np.recarray方法实现了这样的需求(record array)，若x已经是一个结构化数组了，像上面的x，可以用.view将它转换为recarray：

```
x = x.view(np.recarray)
display(x.name)
display(x.grades)
```

输出为：

```
array(['shen', 'yang'], dtype='<U16')
array([[[1., 2.],
        [3., 4.]],

       [[5., 6.],
        [7., 8.]]], dtype=float32)
```

新建一个recarray时，需要指定这个recarray的shape，dtype：

```
x = np.recarray(shape=(2,),dtype=dt)
```

此时x中是garbage数据，可以用buf参数赋给recarray一个符合要求的数组：

```
np.recarray((2,),dtype=dt,buf=np.array([('shen',[[1,2],[3,4]]),('yang',
[[5,6],[7,8]])],dtype=dt))
```

一个比较典型的例子，有1000个2维的点，每个点的坐标的数据类型都是`float32`，现在想通过.x和.y来获取它的坐标：

```
dtype_point = np.dtype([('x',np.float32),('y',np.float32)])
point_arrays = np.random.random((1000,2)).astype(np.float32)
point_recarrays = np.recarray(shape=
(1000,),dtype=dtype_point,buf=point_arrays)
```

此时可以直接用`point_recarrays[0].x`获取元素.

说明

这种结构化数组的好处就是内存分布是完全确定的，因此它是直接被numba所支持的，不需要任何其他设置：

```
import numba as nb

@nb.jit(nopython=True)
def func_jit(point_recarrays):
    _sum = 0
```

```
    for point in point_recarrays:
        _sum += np.abs(point.x) + np.abs(point.y)
    return _sum

def func_nojit(point_recarrays):
    _sum = 0
    for point in point_recarrays:
        _sum += np.abs(point.x) + np.abs(point.y)
    return _sum
```

比较一下运行时间：

```
%%timeit
func_jit(point_recarrays)
```

```
1.92 µs ± 130 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

```
%%timeit
func_nojit(point_recarrays)
```

```
11.4 ms ± 1.47 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

当然这种自定义的dtype没法直接用numpy提供的函数，numpy提供了通过写C文件来扩展自定义类型函数的方法，但是我暂时还没学会..https://numpy.org/doc/stable/user/c-info.ufunc-tutorial.html