**Ell: 序列空间上运算的实现及其应用**

宋丛威[[1]](#footnote-1)

（北京雁栖湖应用数学研究院，北京 怀柔 101408）

摘要：Ell是笔者开发的用于实现序列空间上运算的数值计算库。它也可以处理任何序列模型，如图像可以理解成二维序列。Ell类直接继承numpy的数组类。和数组不同的是，它要模拟定义在整数上的序列，必须为每个对象存储指标的上下界。根据上下界，程序可以自动调整大小，实现两个任意大小的同维序列的初等运算。最后复杂的图像处理实验验证了程序的可靠性和实用性。

关键字：Ell，序列空间，numpy，图像处理，小波分析

1. **概述**

全体序列构成线性空间，是数学重要的研究对象。最常见的序列是数列，等价于定义在自然数（或整数）上的函数[1]。数列和幂级数是对应的[2]。当数列有限时，对应于多项式（或Laurent多项式[3]）。见下式。

计算机显然只能存储有有限数列。因此可以用Python第三方库numpy提供的array[4]来表示数列。但是，这么做是不够的，数组没有提供指标的上下界。两个数列运算时，即使有相同的维数，但是它们的指标没有对齐，也是不能直接调用array的方法。开发Ell就是为了解决这个难题。

之所以将这个库称为Ell，是因为序列空间常用Latex代码\ell表示，显示为*ℓ*。在Python程序设计的语境中，我们也把Ell的各种子类称为Ell类，类的对象称为ell，就像numpy中表示向量和矩阵的对象称为array。

除了实现序列运算外，设计Ell的另一个更为重要的动机是图像处理。笔者在设计Ell时，正在教授小波分析课程，希望亲手实现小波分析算法[5]，程序尽可能适应数学公式的写法，而不是直接使用像PyWavelets[6]封装良好的第三方库。

1. **设计思路**

设计思路其实也很简单，就是继承numpy.ndarray, 并设置min\_index, max\_index属性记录数组指标的上下界，如序列指标的上下界分别为0和3。不过也需要处理很多复杂的细节。

直接继承numpy.ndarray的基类是BaseEll，主要实现基本操作，其余Ell类是其子类。主要子类有Ell1d, Ell2d, Ellnd，分别表示一维，二维，高维序列。Ellnd已经可以表示所有ell了，只是目前绝大多数运算只关于一维和二维序列，因此特别独立出两个子类，优化各自的数据结构和运算。读者在构造自己的Ell类时，可继承BaseEll或者其中某个子类。笔者未来扩展新的Ell类也会如此。子类分得过多可能会带来类型转换的麻烦，不过程序基本能自动进行合理的类型转换。此外，我们只建议操作min\_index, 而max\_index可根据ell的shape属性推断出来。

在做初等运算的时候，要对两个ell进行指标的对齐。因此必须重载所有array基本的代数运算，在调用父方法之前，对齐指标，即给出包含所有上下界的最小范围，参与运算的序列也要通过填充0拉伸到到相应范围。这些处理并不简单，但却是非常关键的。以加法运算为例。

def fit(f):

def \_f(obj, other):

if isinstance(other, BaseEll):

mi, ma = common\_index(obj.index\_pair, other.index\_pair)

obj = obj.resize(mi, ma)

other = other.resize(mi, ma)

return f(obj, other)

return \_f

class BaseEll

@fit

def \_\_iadd\_\_(self, other):

return \_iadd(self, other)

def common\_index(\*index\_pairs):

# common index, the minimal range containing all ranges

mi, ma = tuple(zip(\*index\_pairs))

return np.min(mi, axis=0), np.max(ma, axis=0)

指标对齐，最关键的两步：计算公共指标，然后根据公共指标进行resize操作。resize的细节比较复杂，不宜展示。对于一维ell，主要程序用伪代码描述。

# 计算拉伸后的指标和自身原有指标的差距

m = min\_index - self.min\_index

M = max\_index - self.max\_index

if m < 0:

# 在self前面/左侧补0

elif m > 0:

# 对self切片

if M > 0:

# 在self后面/右侧补0

elif M < 0:

# 对self切片

# 获得新ell并返回(具有公共指标)

1. **应用实例**

下面演示Ell基本用法。Ell的API尽可能和array一致，因此非常容易上手。我们来看几个简单实例。

运行前，请通过命令import ell import \*导入模块。

**3.1 算法构造**

Ell是numpy.ndarray的子类，继承了几乎所有数组运算。由于Ell的设计，运算时不要求两个ell对齐，但应该有相同的维度。下面做一个序列减法：

a = Ell1d([1,2,3,4]) # min\_index=0

b = Ell1d([2,3,4,5,6,7], min\_index=-3)

c = Ell1d([-2.0, -3.0, -4.0, -4.0, -4.0, -4.0, 4.0], min\_index=-3)

assert a-b==c

只要没有报错，程序就是正确的。其余初等运算类同。

只有一个运算例外，Ell重载@用来实现卷积。@在numpy中是运来计算矩阵乘积的，而序列的卷积等价于其对应的Toeplitz型矩阵的乘积。可以说这也是Ell的一个特色。对于一维ell，卷积核数组卷积是一样的，对于高维ell（高维之间，高维和一维之间），要区分整体卷积和按轴卷积。下面我们用非常“数学化”的命令实现Sobel边缘检测。运行结果见图1。

# 构造 Sobel 矩阵

s1 = Ell1d([-1,-2,-1], min\_index=-1)

s2 = Ell1d([1,0,-1] , min\_index=-1)

s = s1.tensor(s2) # 等价于Ell2d([[-1,0,1],[-2,0,2], [-1,0,1]], min\_index=-1)

im = ImageGray.open('lenna.jpg') # 或者 im = ImageRGB.open('lenna.jpg')

s\_im = im @ s

# Ell 允许一维和高维ell沿矩阵轴进行卷积，速度会更快

# 等价于s\_im = im.conv1d(s1, axis=0).conv1d(s2, axis=1)

# 另一个方向的Sobel边缘检测是 im.conv1d(s1, axis=1).conv1d(s2, axis=0)

s\_im.imshow()



图1 Sobel边缘检测试验

**3.2 复杂实例**

图像是二维序列。为了处理彩色图像，需要增加一维存储三个channel值。因此底层的array是三维的，但这个序列依然是二维的，第三维是不需要指标。这样的系列被称为多值序列。基类为BaseMultiEll，子类有MultiEll1d, MultiEll2d, MultiEllnd, 意义与单值序列相似，其中MultiEll2d是RGB图像序列类ImageRGB的父类.

很幸运，numpy的API使得多值序列类大多数方法可以直接继承BaseEll无需重载。Sobel边缘检测可以应用于ImageRGB对象，而无需修改代码，结果见图2。多值ell与同维单值ell可以像向量和数量那样运算，可将单值ell理解成重复值的多值ell。



图2 Sobel边缘检测试验（RGB彩图）

在上一个例子中，我们用卷积运算完成一个简单的图像处理任务，本例要实现小波分析中著名的Mallat算法[7,8]。

运行GitHub仓库中examples/mallat.py文件，产生图2。结果说明算法运行非常成功。此外还提供了著名的pyramid算法，图像压缩，图像融合[9]应用实例等。

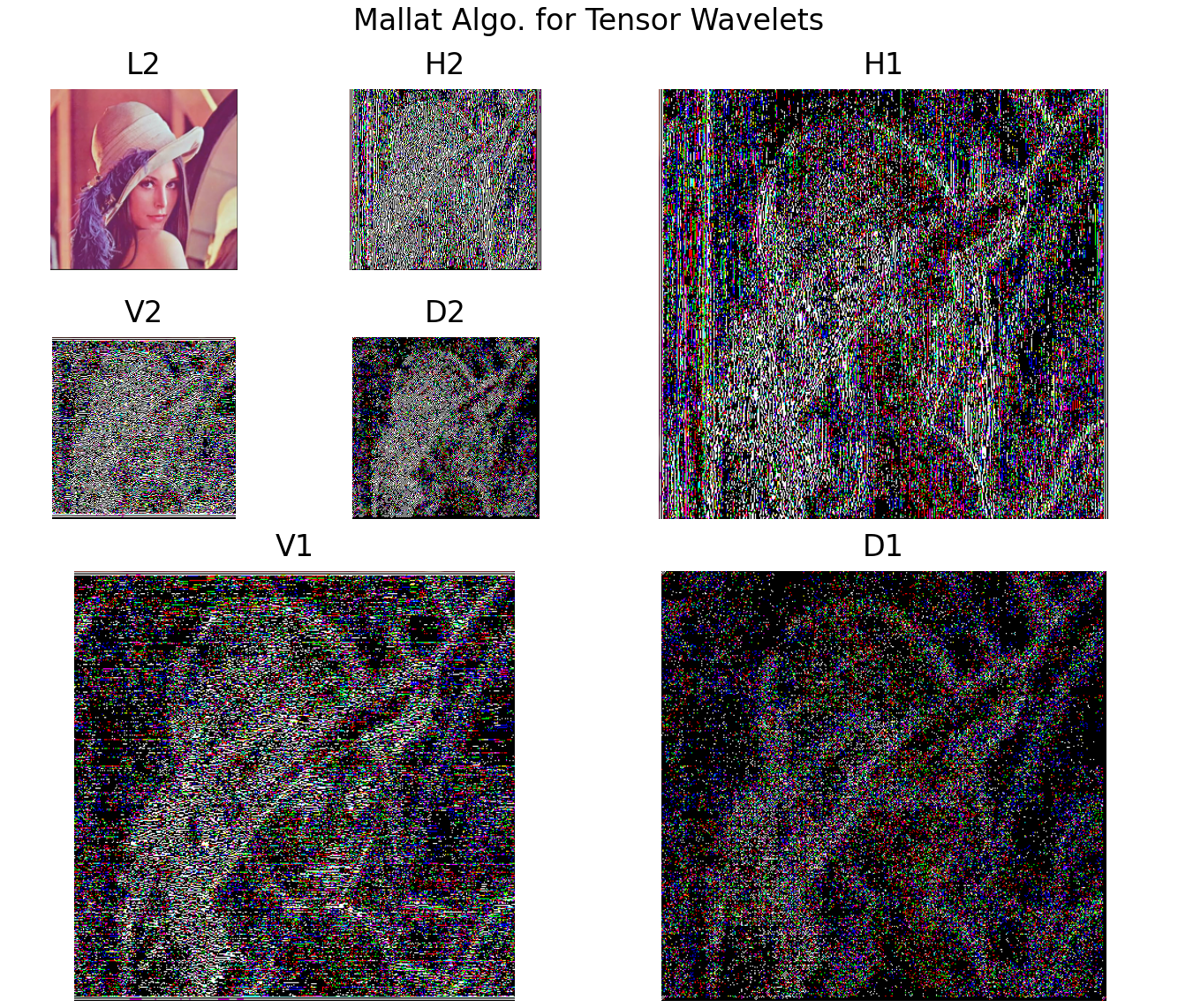


图3 图像层级分解的Mallat 算法

1. **结语**

Ell的实现得益于numpy的强大。许多功能可以自然迁移到复杂的Ell类上。本文简要展示了它的用法，和一个较为复杂的小波分析案例。在某些方面，它比PyWavelets等图像处理库易用。它的设计原则是尽可能符合数学上的习惯。

Ell还处于开发中。未来会建立表示音频的Ell类。这没有较大的技术难点。音频是一维的，如果是双声道，那就是多值一维序列。视频的Ell类自然是（多值）三维的。比较有难点的是定义稀疏序列。需要一个和数值数组对应的数组记录指标。可能还会考虑混合形式，比如二维序列，第一维稀疏，第二维不稀疏。这又将是一个大工程。稀疏ell，和一般ell共享运算，只是存储形式不同，非常适合于存储具有稀疏性的小波系数。

另一个技术难点是索引。Ell不仅应该索引其指标范围内的数值，还可以索引范围以外的数值，也就是0。索引指标本身对应于序列本身的指标，而不是像numpy的数组那样从0索引。Ell虽然是有限维的，但是表示的是含有限非零值的无限维数组。如果要表示无限非零值的数组，需要借助函数，该函数能产生任意位置的数值。

图像处理出现大量的卷积运算自动改变图像大小。可以通过resize方法裁剪大小。但是这可能丢掉信息。最好的办法是建立周期Ell类。这些类（代表的序列）在卷积时是不会改变大小的。但这样的卷积需要专门定义。

完整源码已经上传至GitHub，<https://github.com/Freakwill/ell>，包含大量有趣的实例。（见文件夹examples）笔者希望有感兴趣的读者参与到Ell的开发中。

**参考文献**

[1] Sequence space. <https://en.wikipedia.org/wiki/Sequence_space>[OL], 2021.

[2] Joseph D. [Sequences and series in Banach spaces](https://archive.org/details/sequencesseriesi0000dies)[M]. Springer-Verlag, 1984.

[3] Laurent polynomial. <https://en.wikipedia.org/wiki/Laurent_polynomial>[OL], 2021.

[4] Numpy. [https://numpy.org/[OL](https://numpy.org/%5bOL)], 2021.

[5] Daubechies I. Ten Lectures on Wavelets[M]. Philadelphia, the Society for Industrial and Applied Mathematics, 1992.

[6] PyWavelets. <https://pywavelets.readthedocs.io/en/latest/> [OL], 2021.

[7] Sobel I. History and Definition of Sobel Operator[OL]. 2014. <https://www.researchgate.net/publication/239398674_An_Isotropic_3x3_Image_Gradient_Operator>

[8] 邸继征. 小波分析原理[M]. 北京: 科学出版社, 2010.

[9] 孙延奎. 小波变换与图像、图形处理技术 (第二版)[M]. 北京: 清华大学出版社, 2018.

1. 作者简介**：**宋丛威（1986—），男，博士，助理研究员，研究方向为小波分析、调和分析、机器学习

   收稿日期**：**2021-7-xx [↑](#footnote-ref-1)