Painless Application Development with 3D Slicer and Python

Basic Python and Index tricks

Demian Wassermann
LMI 09/2008

Python Basics

Python Basics

- Object oriented language

- Interpreted language
- Everything is an object
- References are the main way of passing parameters
- Duck typing

Python Basics

- Object oriented language
- Some functional capabilities

- Lambda expression
- Lists by comprehension
- Partial evaluation not directly supported

Python Basics

- Object oriented language
- Some functional capabilities
- Everything can be defined or redefined in execution time

Python basic Datatypes

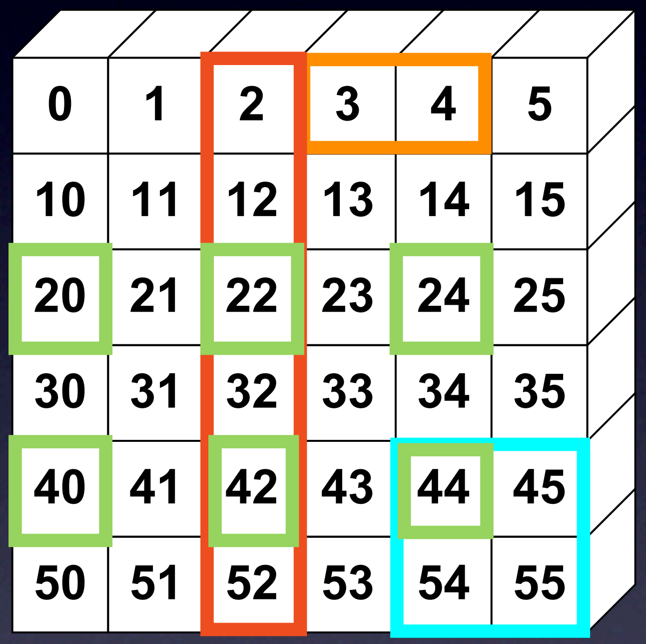
- Literals (i.e. integers, floats, complex and charactes)
- Tuples: fixed combinations of objects `t=('a',2)`
- Lists: resizable combinations of objects `t=['a',2]`
- Dictionaries: key/value pairs `t['a']=2`
- Sets: non-iterable containers with a fast pertenence operation `t={'a',2}`
- Functions
- Classes
- Modules

Numpy basic Datatypes

- array datatype
  - Multidimensional array
  - Operations are done in an element by element basis
- matrix datatype
  - Bidimensional array of elements
  - matrix semantics

Numpy: slicing

```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,22,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



> Slicing does not create copies of the array's contents

[Jones,Oliphant]

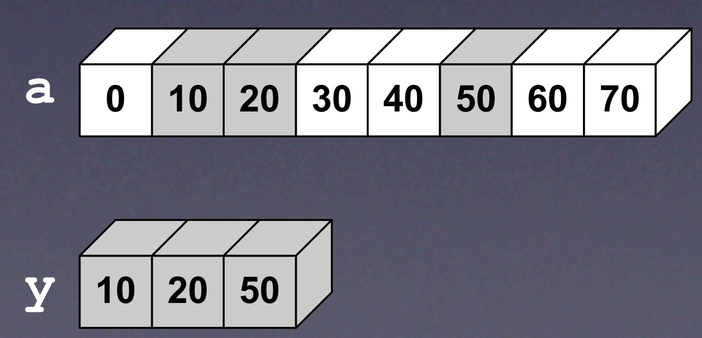
Numpy: fancy indexing

INDEXING BY POSITION

```
>>> a = arange(0,80,10)


# fancy indexing
>>> y = a[[1, 2, -3]]
>>> print y
[10 20 50]

# using take
>>> y = take(a,[1,2,-3])
>>> print y
[10 20 50]
```

INDEXING WITH BOOLEANS

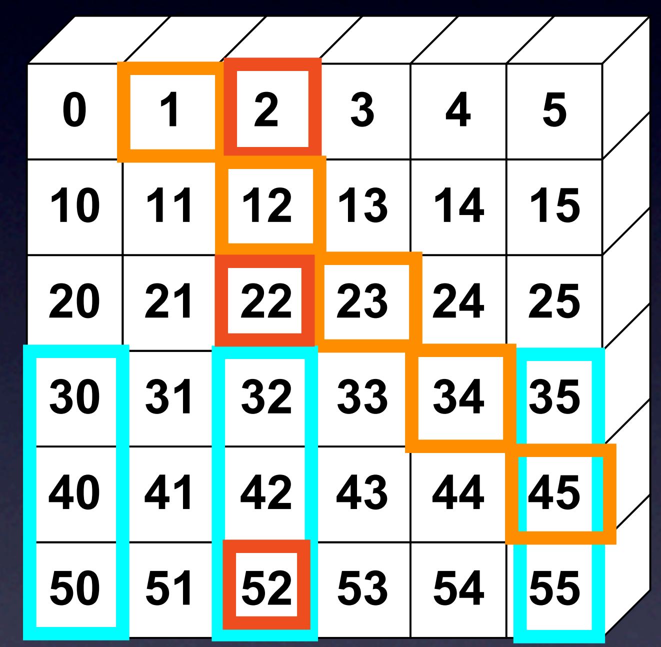
```
>>> mask = array([0,1,1,0,0,1,0,0],
...                dtype=bool)

# fancy indexing
>>> y = a[mask]
>>> print y
[10,20,50]

# using compress
>>> y = compress(mask, a)
>>> print y
[10,20,50]
```



[Jones,Oliphant]

Numpy: fancier indexing

```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]
array([ 1, 12, 23, 34, 45])

>>> a[3:,[0, 2, 5]]
array([[30, 32, 35],
       [40, 42, 45]])
       [50, 52, 55]])

>>> mask = array([1,0,1,0,0,1],
                 dtype=bool)
>>> a[mask,2]
array([2,22,52])
```

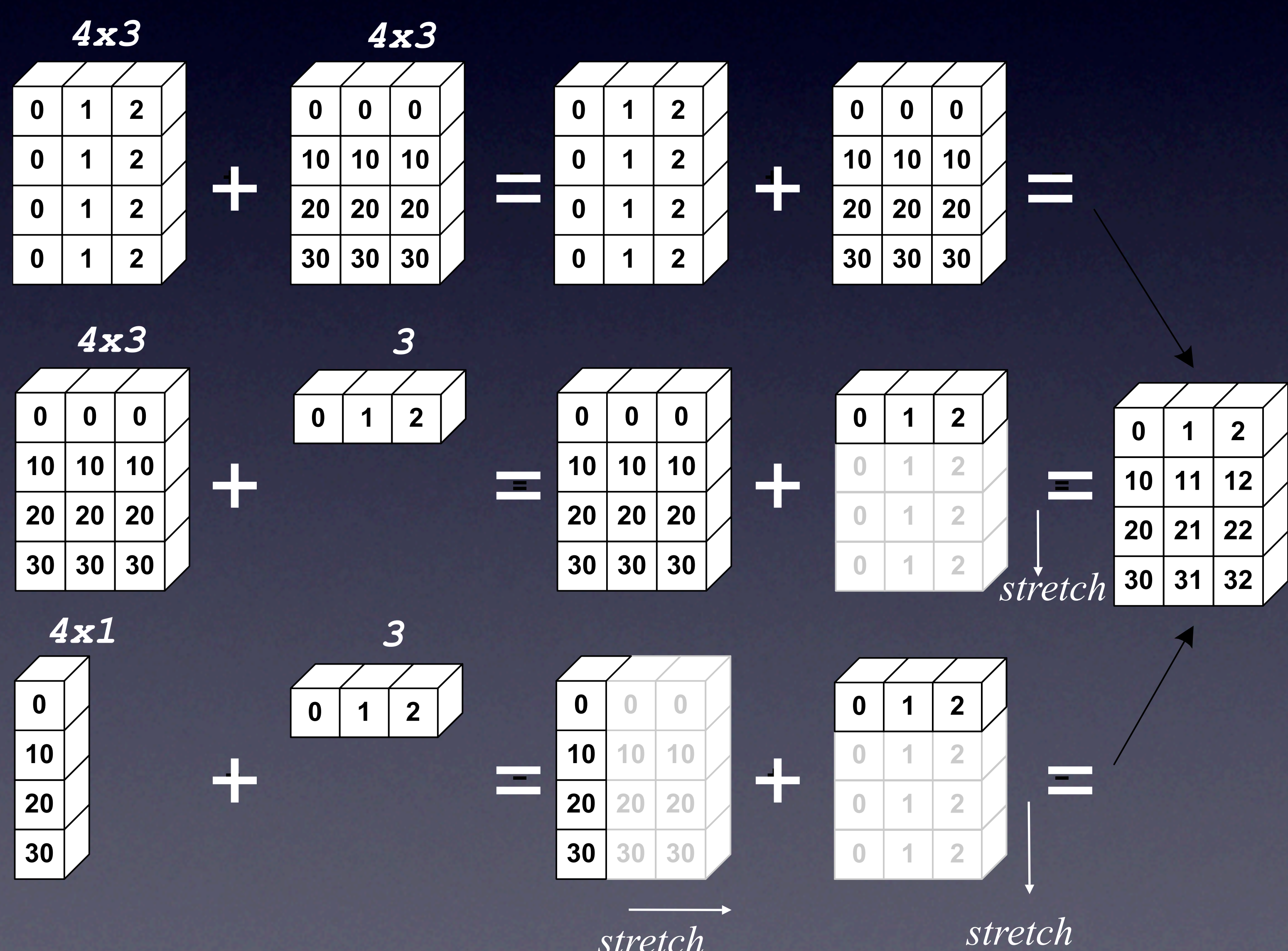


Unlike slicing, fancy indexing creates copies instead of views into original arrays.

[Jones,Oliphant]

Numpy broadcasting

Semantic of binary operations between arrays

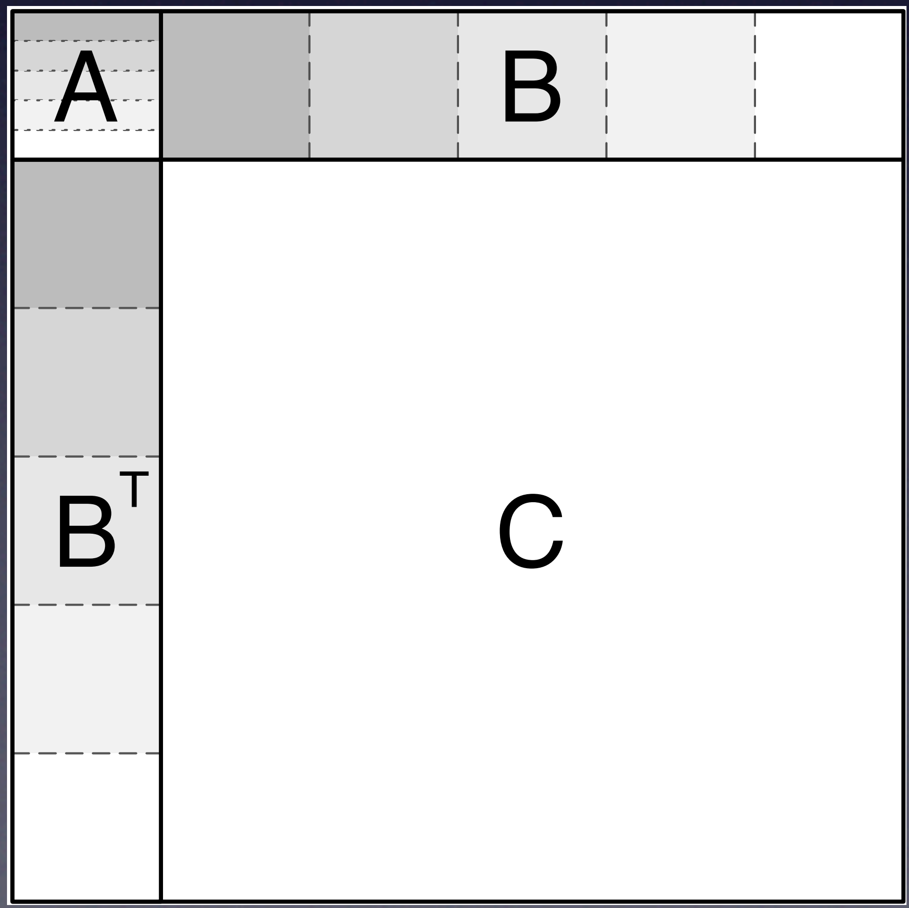


[Oliphant]

Full example: NCuts Nÿstrom

Get the most significative eigenvectors of a normalized matrix W, using only the submatrices A and B

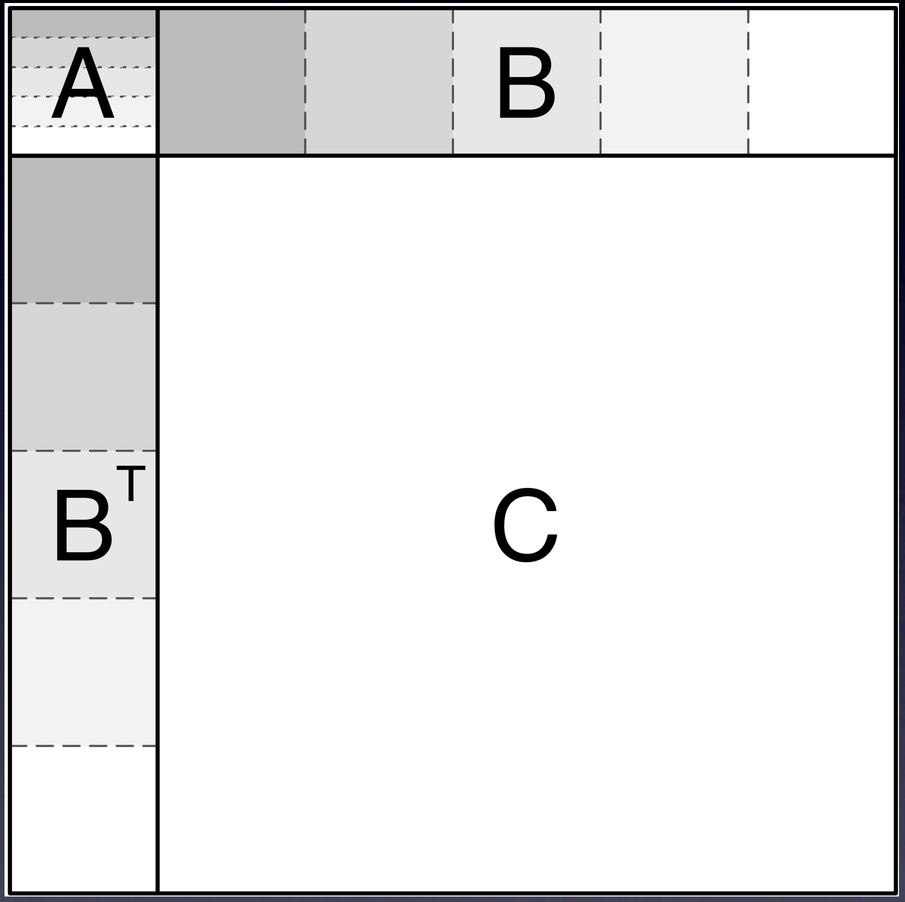
$$\mathcal{W} = \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}.$$



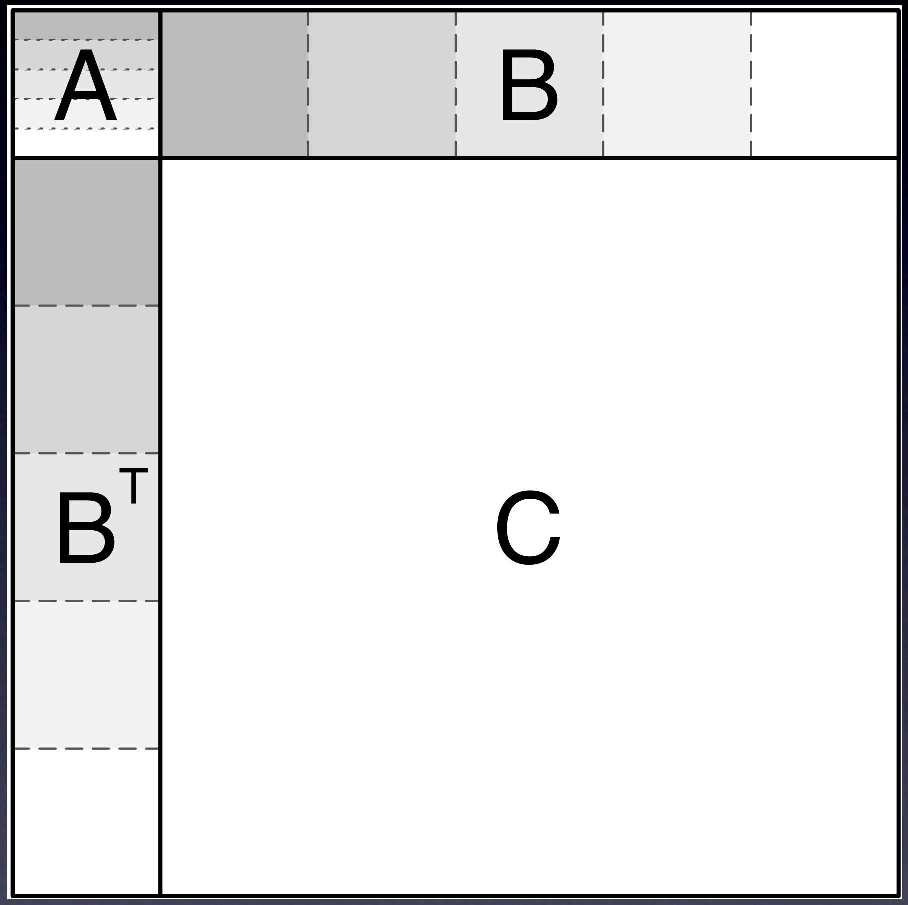
- Subsampling W to get A
- Normalizing of the partial parts
- Eigenvalue decomposition of A
- Estimation of the Eigenvectors of W

Full example: NCuts Nÿstrom

Subsampling D to get A and B



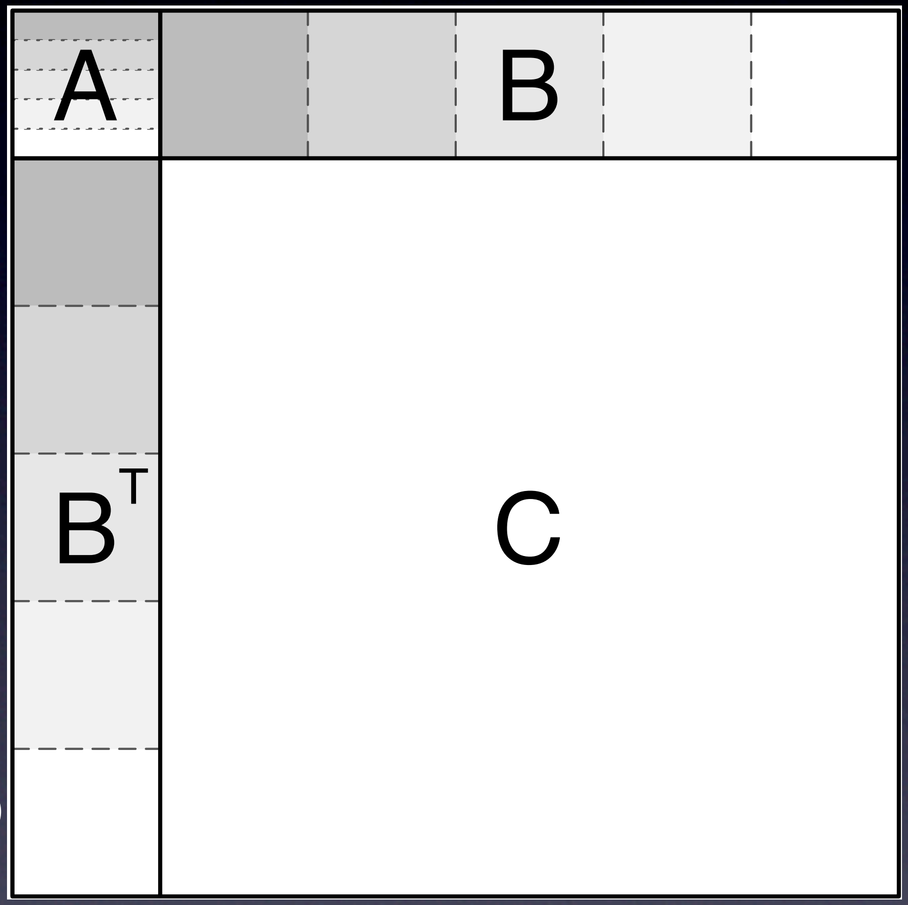
Full example: NCuts Nÿstrom



Subsampling D to get A and B

```
#W is the initial matrix
#ratio is the relative size of A with respect to W

import numpy
from numpy import random
```

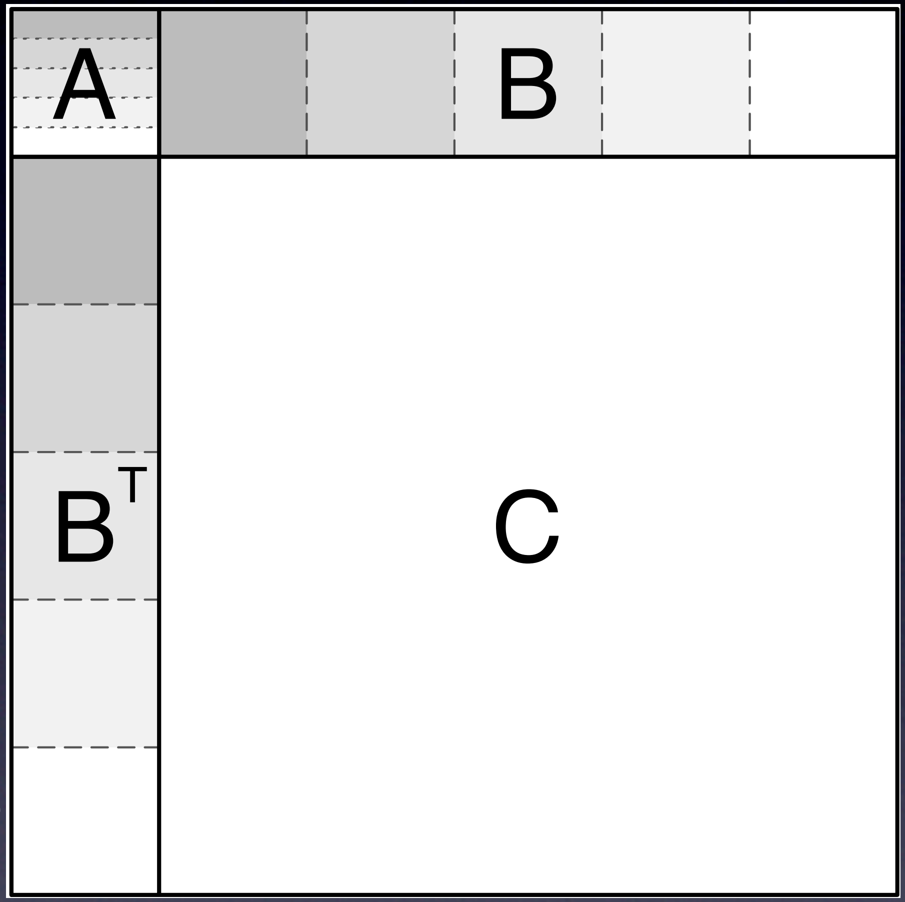
Full example: NCuts Nÿstrom



Subsampling D to get A and B

```
#W is the initial matrix
#ratio is the relative size of A with respect to W

import numpy
from numpy import random

shuffled_indexes = numpy.arange( W.shape[0], dtype=int )
random.shuffle( shuffled_indexes )
```

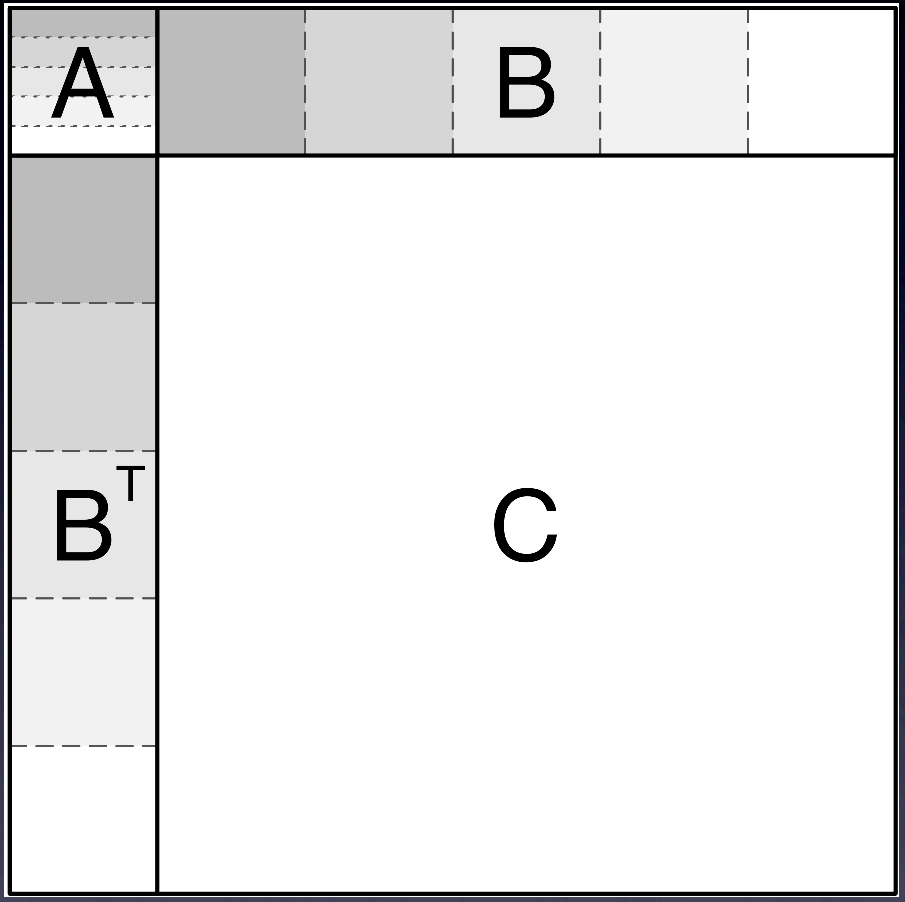
Full example: NCuts Nÿstrom



Subsampling D to get A and B

```
#W is the initial matrix
#ratio is the relative size of A with respect to W

import numpy
from numpy import random

shuffled_indexes = numpy.arange( W.shape[0], dtype=int )
random.shuffle( shuffled_indexes )

Na = int(numpy.round( elementQty*ratio ))
Nb = elementQty-Na
a_indexes = shuffled_indexes[:Na]
b_indexes = shuffled_indexes[Na:]
```

Full example: NCuts Nÿstrom

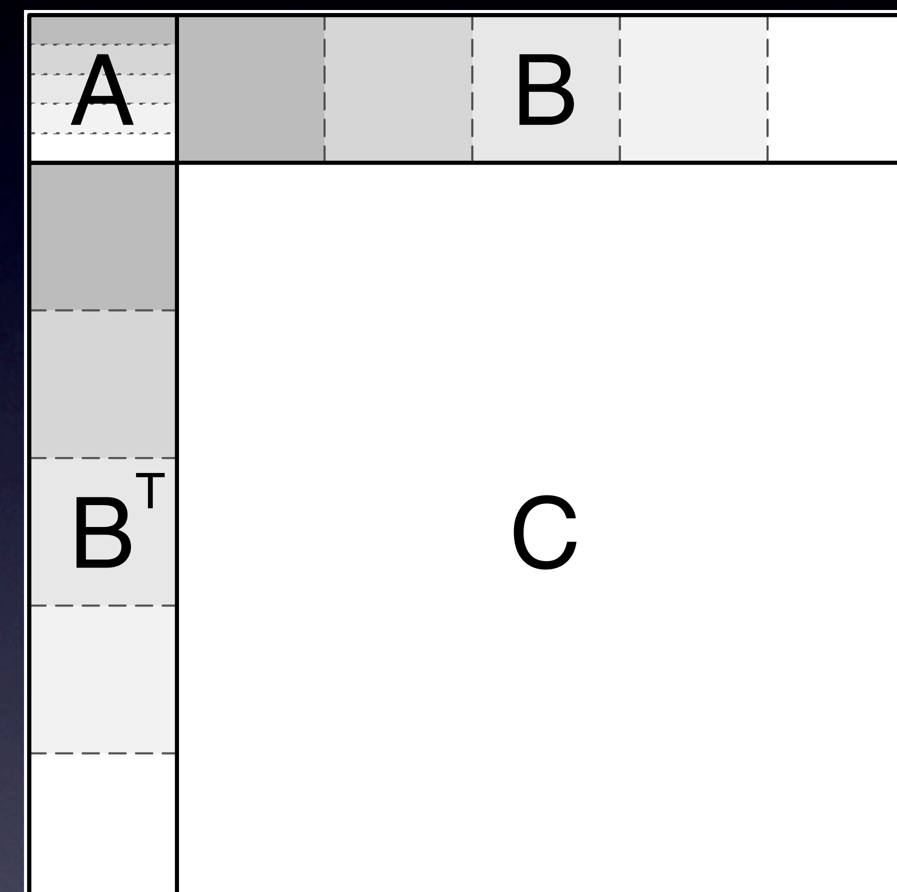


Subsampling D to get A and B

```
#W is the initial matrix
#ratio is the relative size of A with respect to W

import numpy
from numpy import random

shuffled_indexes = numpy.arange( W.shape[0], dtype=int )
random.shuffle( shuffled_indexes )

Na = int(numpy.round( elementQty*ratio ))
Nb = elementQty-Na
a_indexes = shuffled_indexes[:Na]
b_indexes = shuffled_indexes[Na:]

A = numpy.asmatrix(D[a_indexes,:][:,a_indexes])
B = numpy.asmatrix(D[a_indexes,:][:,b_indexes])
```

Full example: NCuts Nÿstrom

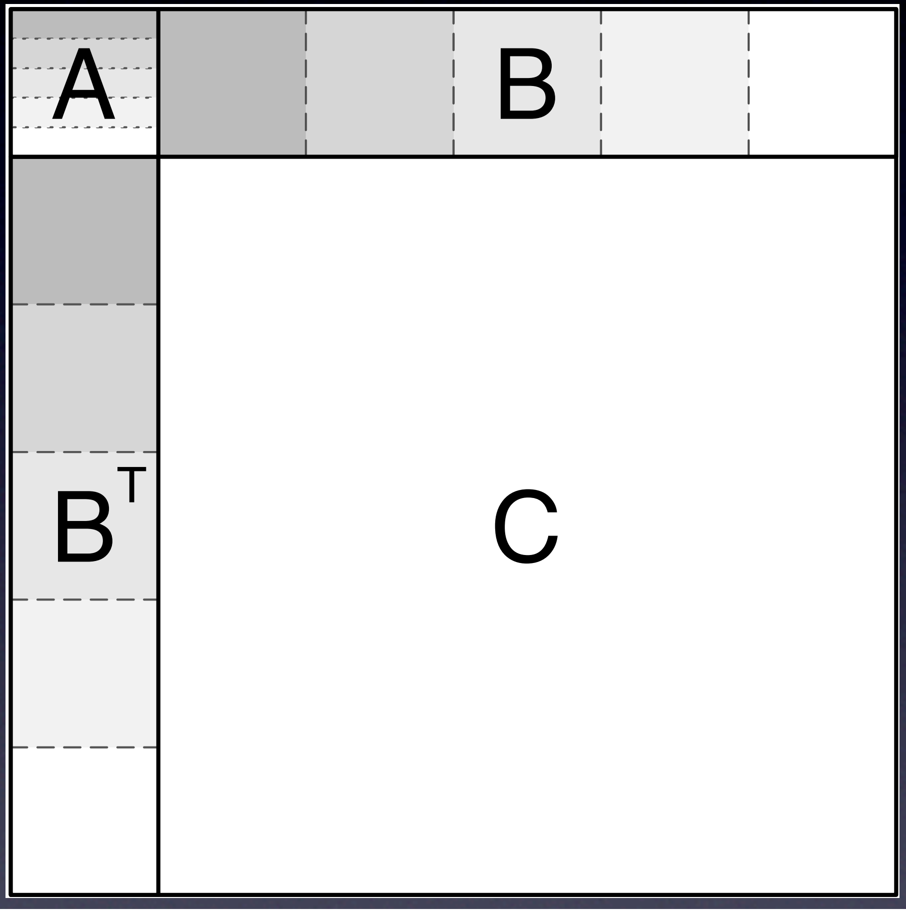
Normalizing A and B



$$\mathcal{W} = \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}.$$

$$\hat{\mathbf{d}} = \begin{bmatrix} \mathbf{a}_r + \mathbf{b}_r \\ \mathbf{b}_c + \mathbf{B}^T\mathbf{A}^{-1}\mathbf{b}_r \end{bmatrix}$$

Full example: NCuts Nÿstrom



Normalizing A and B

```
a_r = A.sum(1)
b_r = B.sum(1)
b_c = B.T.sum(1)
```

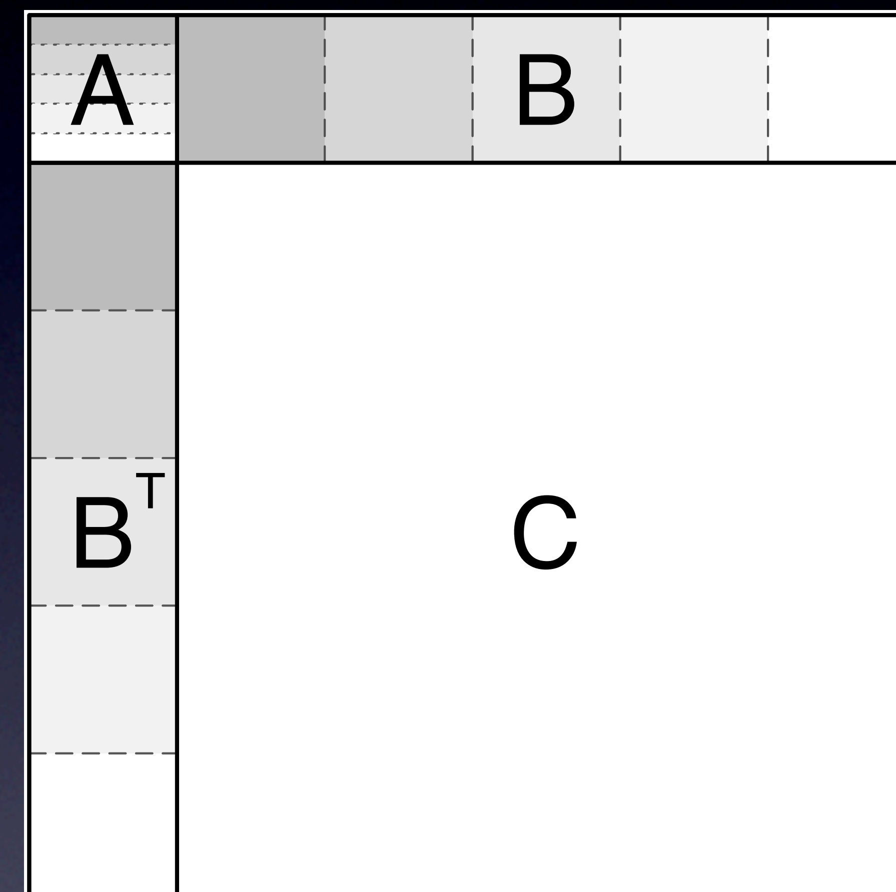
$$\mathcal{W} = \mathbf{D}^{-\frac{1}{2}} \mathbf{W} \mathbf{D}^{-\frac{1}{2}}.$$

$$\hat{\mathbf{d}} = \begin{bmatrix} \mathbf{a}_r + \mathbf{b}_r \\ \mathbf{b}_c + \mathbf{B}^T \mathbf{A}^{-1} \mathbf{b}_r \end{bmatrix}$$

Full example: NCuts Nÿstrom

Normalizing A and B

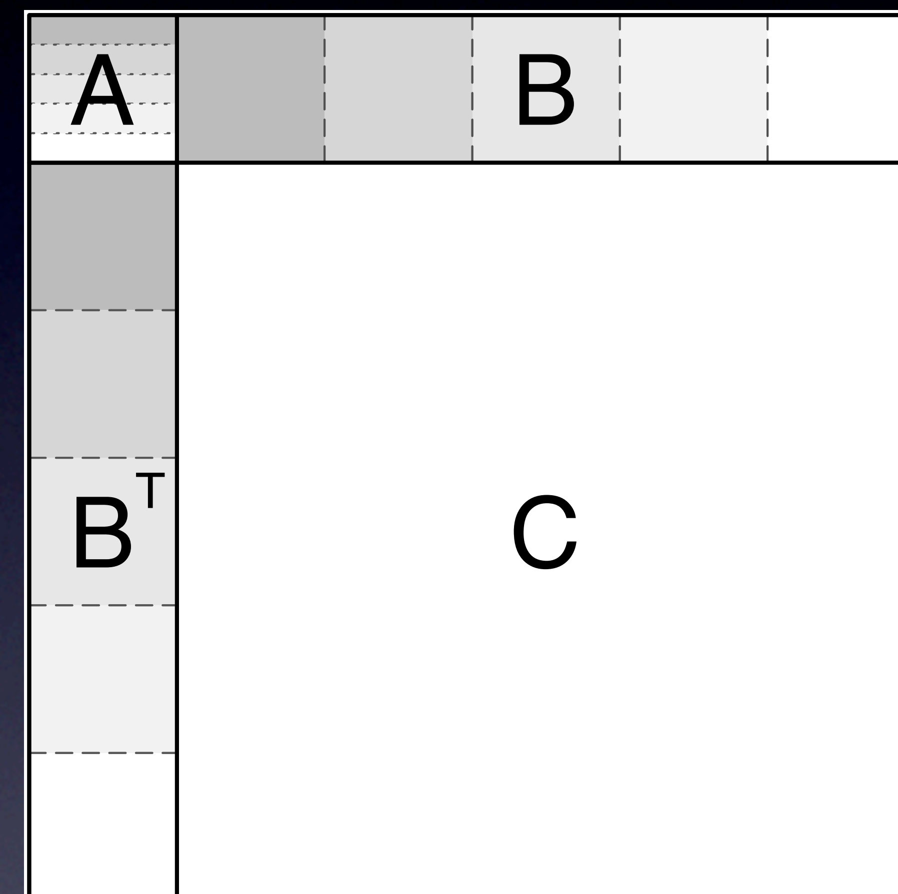
```
a_r = A.sum(1)
b_r = B.sum(1)
b_c = B.T.sum(1)

d = numpy.vstack((\
      a_r + b_r,\
      b_c + (B.T * linalg.inv(A)) * b_r\
      ))

d_inv_sqr = 1./numpy.sqrt(d)
```



$$\mathcal{W} = \mathbf{D}^{-\frac{1}{2}} \mathbf{W} \mathbf{D}^{-\frac{1}{2}}.$$

$$\hat{\mathbf{d}} = \left[\begin{array}{c} \mathbf{a}_r + \mathbf{b}_r \\ \mathbf{b}_c + \mathbf{B}^T \mathbf{A}^{-1} \mathbf{b}_r \end{array} \right]$$

Full example: NCuts Nÿstrom



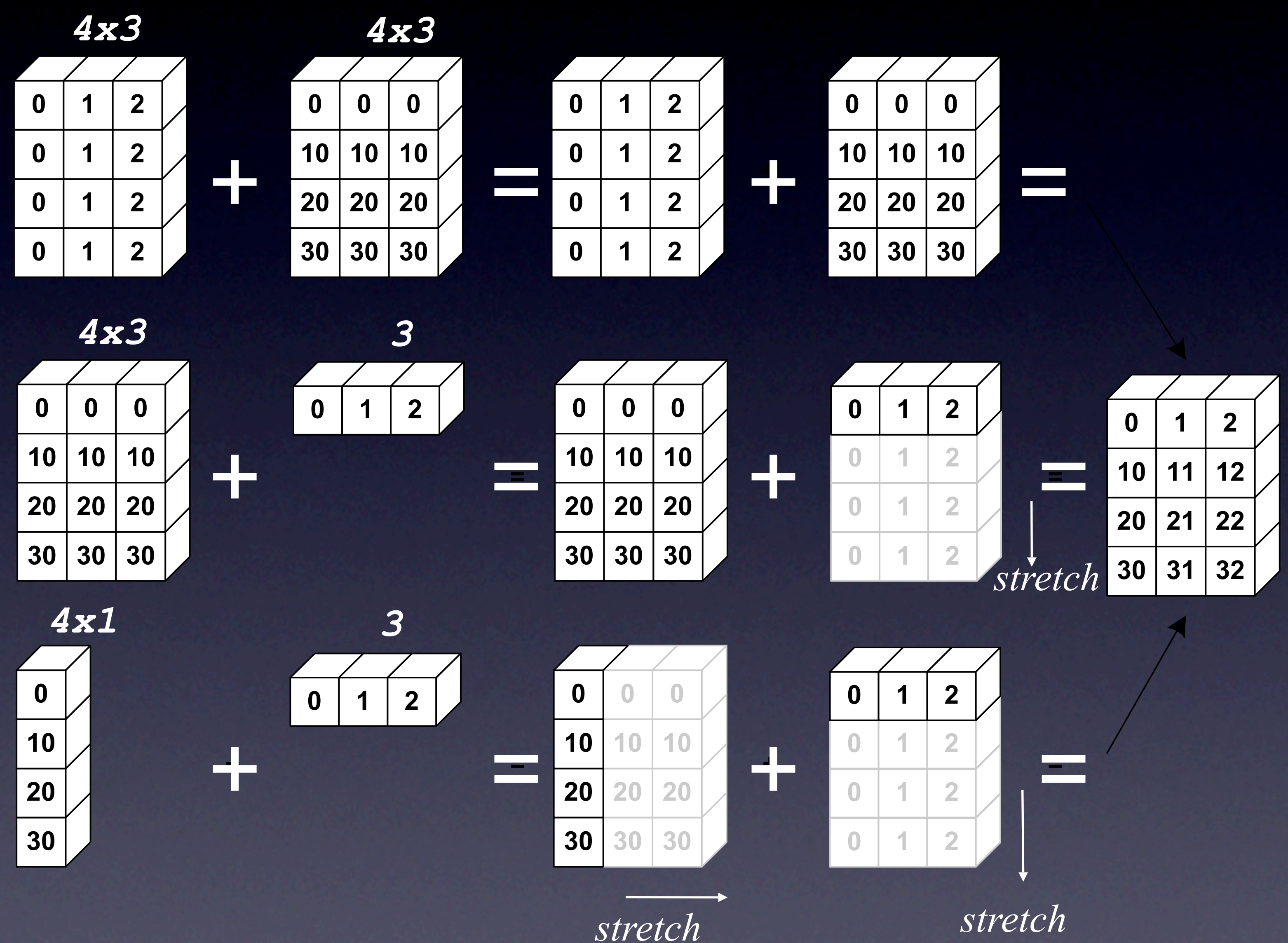
Normalizing A and B

```
 a_r = A.sum(1)
 b_r = B.sum(1)
 b_c = B.T.sum(1)

 d = numpy.vstack((\
      a_r + b_r,\
      b_c + (B.T * linalg.inv(A)) * b_r\
      ))

d_inv_sqr = 1./numpy.sqrt(d)

An = \
 numpy.multiply(\
    numpy.multiply( d_inv_sqrt[:Na], A ),\
 d_inv_sqrt[:Na].T)
Bn = \
   numpy.multiply(\
      numpy.multiply( d_inv_sqrt[Na:].T, B ),\
   d_inv_sqrt[:Na])
```

$$\mathcal{W} = \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}.$$

$$\hat{\mathbf{d}} = \begin{bmatrix} \mathbf{a}_r + \mathbf{b}_r \\ \mathbf{b}_c + \mathbf{B}^T\mathbf{A}^{-1}\mathbf{b}_r \end{bmatrix}$$

Numpy broadcasting



Equivalent syntax for the normalization

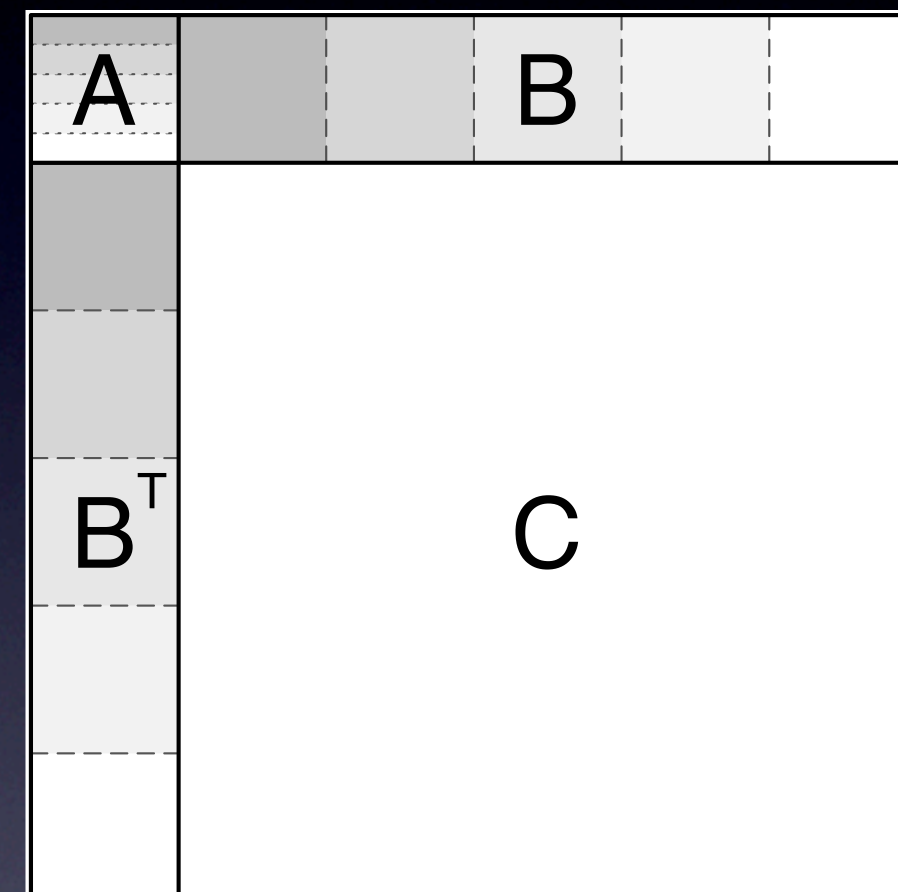
```
d_inv_sqrt = numpy.asarray(d_inv_sqrt)
An = d_inv_sqrt[:Na]*numpy.asarray(A)*d_inv_sqrt[:Na].T
#Warning here the d_inv_sqr and A are arrays not matrices
```

[Oliphant]

Full example: NCuts Nÿstrom

Eigenvalue decomposition of A

```
Delta,U = linalg.eig(An)

Delta_inv = numpy.asmatrix( numpy.diag(1./Delta) )

Ubar = numpy.vstack((\
      U,\
      Bn.T*U*Delta_inv\
      ))

return Delta, Ubar[ numpy.argsort( shuffled_indexes ),:]
```



$$\bar{\mathbf{U}} = \begin{bmatrix} \mathbf{U} \\ \mathcal{B}^T \mathbf{U} \mathbf{\Lambda}^{-1} \end{bmatrix}$$

Time and energy to take a look at one more module?