https://gitlab.com/gersonjferreira/computational-quantum-mechanics

Computational Quantum Mechanics

Prof. Gerson J. Ferreira

INFIS/UFU 2020/1

Eigenstates of 2D systems

INSTITUTO DE FÍSICA
Universidade Federal de Uberlândia

2D systems

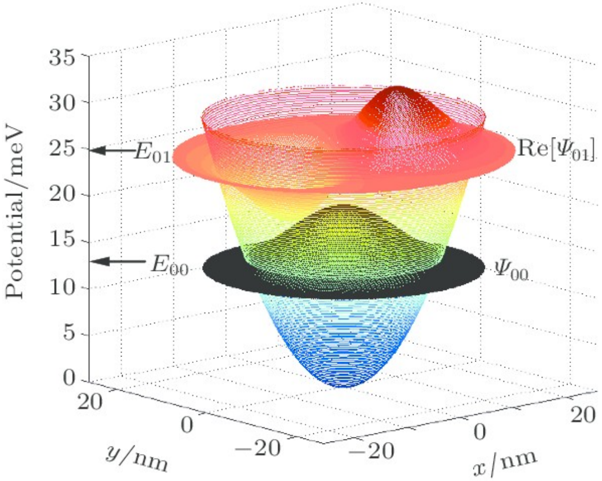
Now we want to consider a 2D Hamiltonian and solve: $H\psi(x,y) = E\psi(x,y)$

$$H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

2D systems

Now we want to consider a 2D Hamiltonian and solve: $H\psi(x,y) = E\psi(x,y)$

$$H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$



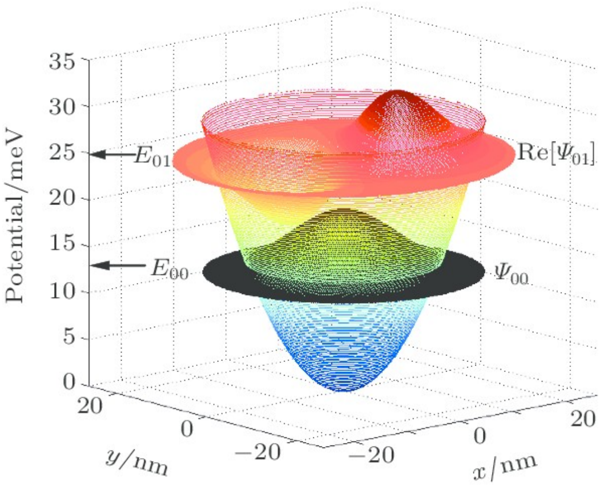
[Edvinsson, R. Soc. Open Sci. 5, 180387 (2018)]

2D systems

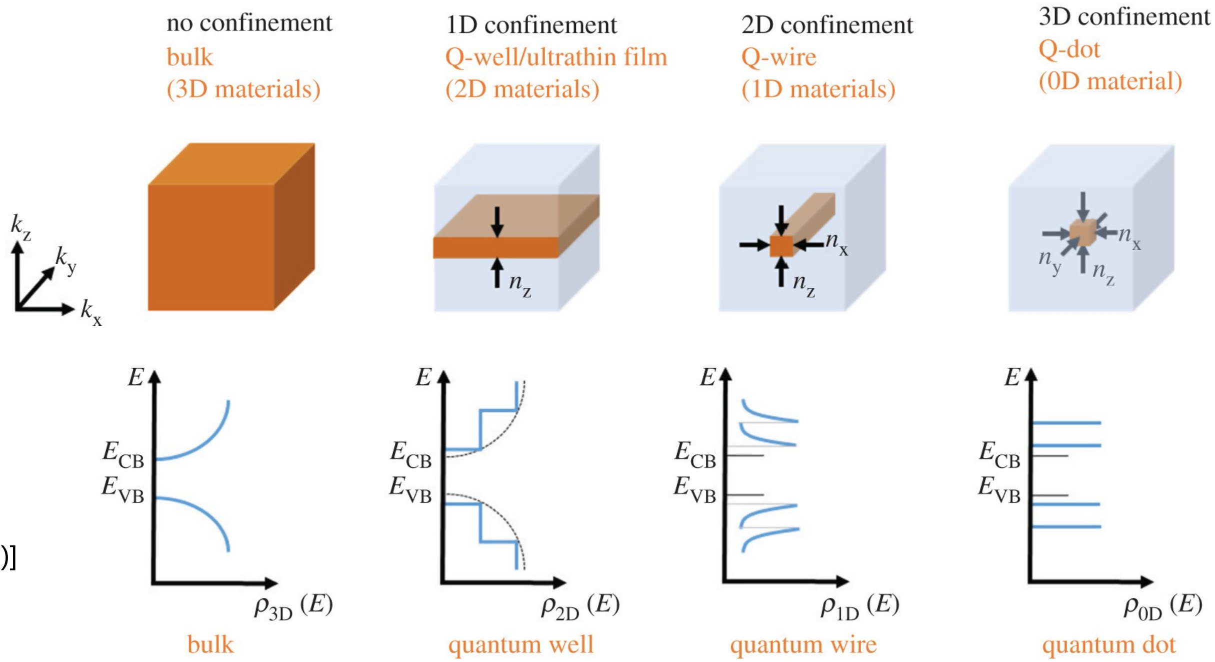
Now we want to consider a 2D Hamiltonian and solve: $H\psi(x,y) = E\psi(x,y)$

$$H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

[Song et al. Chinese Phys. B 21, 057302 (2012)]



[Edvinsson, R. Soc. Open Sci. 5, 180387 (2018)]



Matrix representation for finite differences

$$H\psi(x,y) = E\psi(x,y) \qquad H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

Matrix representation for finite differences

$$H\psi(x,y) = E\psi(x,y) \qquad H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

Before, for 1D systems we had ψ(x)

Matrix representation for finite differences

$$H\psi(x,y) = E\psi(x,y) \qquad H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

Before, for 1D systems we had ψ(x)

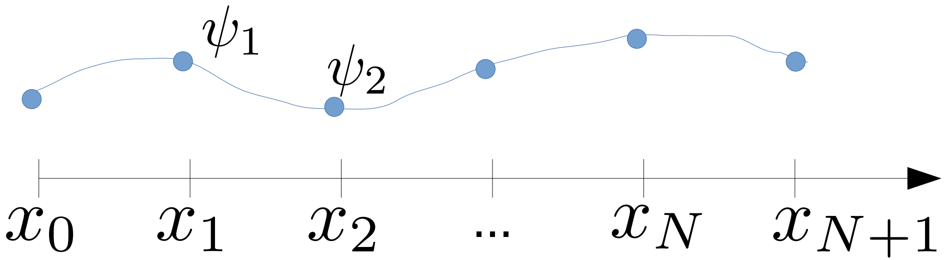
it was natural to write ψ(x) as a vector on the discrete coordinates:

Matrix representation for finite differences

$$H\psi(x,y) = E\psi(x,y) \qquad H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

Before, for 1D systems we had ψ(x)

it was natural to write ψ(x) as a vector on the discrete coordinates:



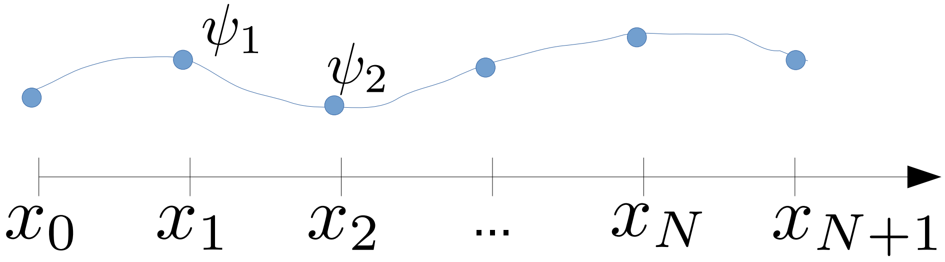
Matrix representation for finite differences

$$H\psi(x,y) = E\psi(x,y) \qquad H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

Before, for 1D systems we had ψ(x)

it was natural to write ψ(x) as a vector on the discrete coordinates:

$$\psi(x) \to \vec{\psi} = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \cdots \\ \psi_{N-1} \end{pmatrix}$$



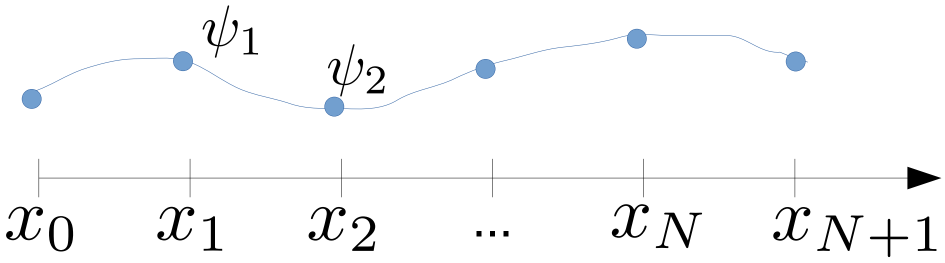
Matrix representation for finite differences

$$H\psi(x,y) = E\psi(x,y) \qquad H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

Before, for 1D systems we had ψ(x)

it was natural to write ψ(x) as a vector on the discrete coordinates:

$$\psi(x) \to \vec{\psi} = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \cdots \\ \psi_{N-1} \end{pmatrix}$$



where

$$\psi_n = \psi(x_n)$$

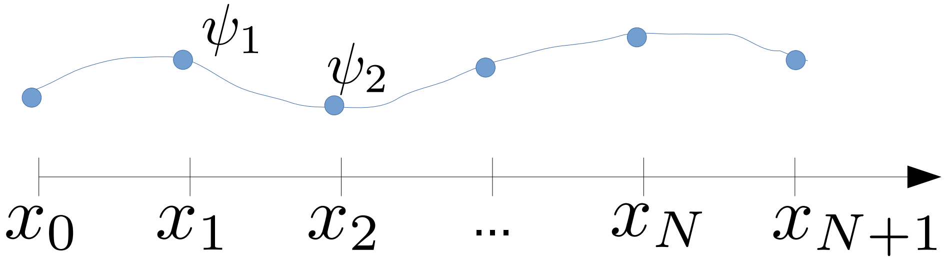
Matrix representation for finite differences

$$H\psi(x,y) = E\psi(x,y) \qquad H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

Before, for 1D systems we had ψ(x)

it was natural to write ψ(x) as a vector on the discrete coordinates:

$$\psi(x) \rightarrow \vec{\psi} = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \cdots \\ \psi_{N-1} \end{pmatrix}$$



where

$$\psi_n = \psi(x_n)$$

$$\psi''_n = \frac{\psi_{n-1} - 2\psi_n + \psi_{n+1}}{\Delta x^2}$$

Matrix representation for finite differences

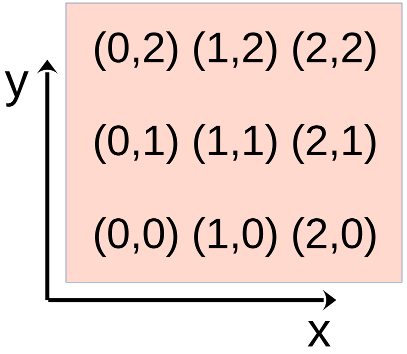
$$H\psi(x,y) = E\psi(x,y) \qquad H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

But now we have ψ(x, y): how do we organize it as a vector?

Matrix representation for finite differences

$$H\psi(x,y) = E\psi(x,y) \qquad H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

But now we have ψ(x, y): how do we organize it as a vector?



(0,2) (1,2) (2,2)

(0,1) (1,1) (2,1)

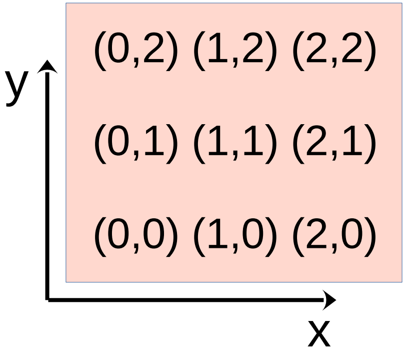
(0,0) (1,0) (2,0)

Matrix representation for finite differences

$$H\psi(x,y) = E\psi(x,y) \qquad H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

But now we have ψ(x, y): how do we organize it as a vector?

Label each point as $p_n = (x_n, y_n)$



(0,2) (1,2) (2,2)

(0,1) (1,1) (2,1)

(0,0) (1,0) (2,0)

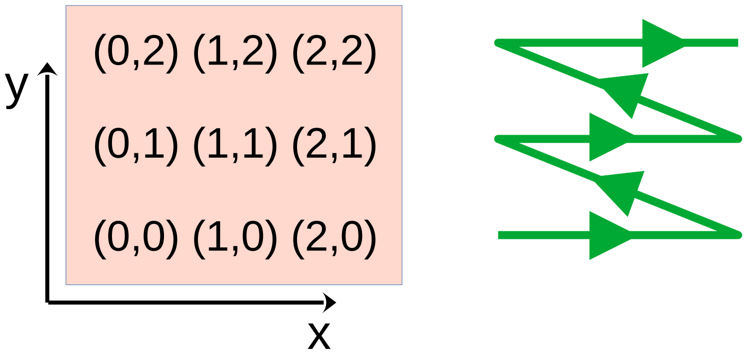
Matrix representation for finite differences

$$H\psi(x,y) = E\psi(x,y) \qquad H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

But now we have ψ(x, y): how do we organize it as a vector?

Label each point as $p_n = (x_n, y_n)$

Sort as To get →



(0,2) (1,2) (2,2)

(0,1) (1,1) (2,1)

(0,0) (1,0) (2,0)

Matrix representation for finite differences

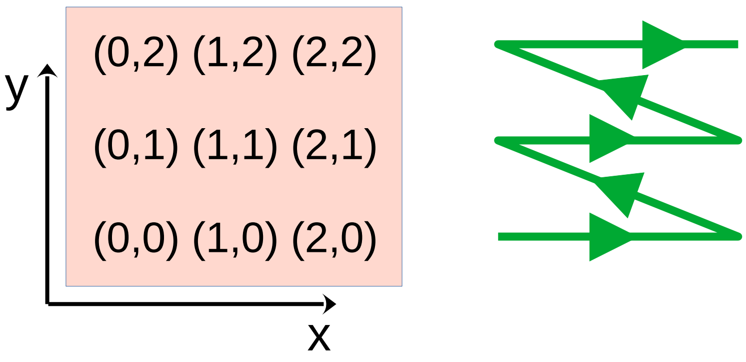
$$H\psi(x,y) = E\psi(x,y) \qquad H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

But now we have ψ(x, y): how do we organize it as a vector?

Label each point as $p_n = (x_n, y_n)$

Sort as

To get →



(0,2) (1,2) (2,2)

(0,1) (1,1) (2,1)

(0,0) (1,0) (2,0)

$p_0 = (0, 0)$
$p_1 = (1, 0)$
$p_2 = (2, 0)$
$p_3 = (0, 1)$
$p_4 = (1, 1)$
$p_5 = (2, 1)$
$p_6 = (0, 2)$
$p_7 = (1, 2)$
$p_8 = (2, 2)$

Matrix representation for finite differences

$$H\psi(x,y) = E\psi(x,y) \qquad H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

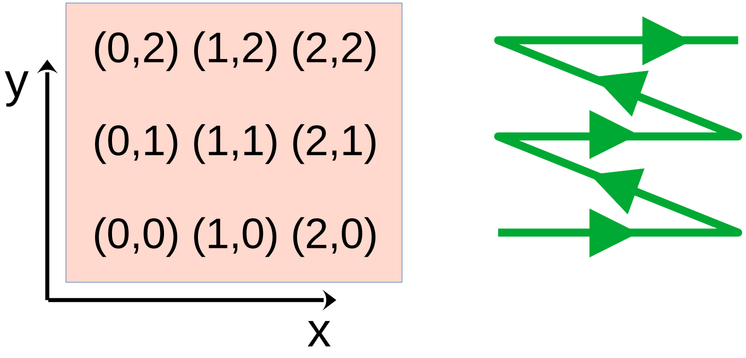
But now we have ψ(x, y): how do we organize it as a vector?

Label each point as $p_n = (x_n, y_n)$

Sort as

To get →

$p_0 = (0, 0)$
$p_1 = (1, 0)$
$p_2 = (2, 0)$
$p_3 = (0, 1)$
$p_4 = (1, 1)$
$p_5 = (2, 1)$
$p_6 = (0, 2)$
$p_7 = (1, 2)$
$p_8 = (2, 2)$



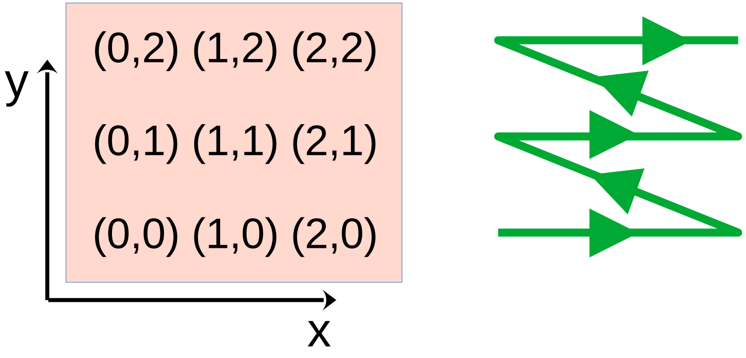
(0,2)	(1,2)	(2,2)
(0,1)	(1,1)	(2,1)
(0,0)	(1,0)	(2,0)

Matrix representation for finite differences

$$H\psi(x,y) = E\psi(x,y) \qquad H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

But now we have ψ(x, y): how do we organize it as a vector?

Label each point as $p_n = (x_n, y_n)$



(0,2) (1,2) (2,2)

(0,1) (1,1) (2,1)

(0,0) (1,0) (2,0)

Sort as

To get →

$p_0 = (0, 0)$
$p_1 = (1, 0)$
$p_2 = (2, 0)$
$p_3 = (0, 1)$
$p_4 = (1, 1)$
$p_5 = (2, 1)$
$p_6 = (0, 2)$
$p_7 = (1, 2)$
$p_8 = (2, 2)$

In Python… use:

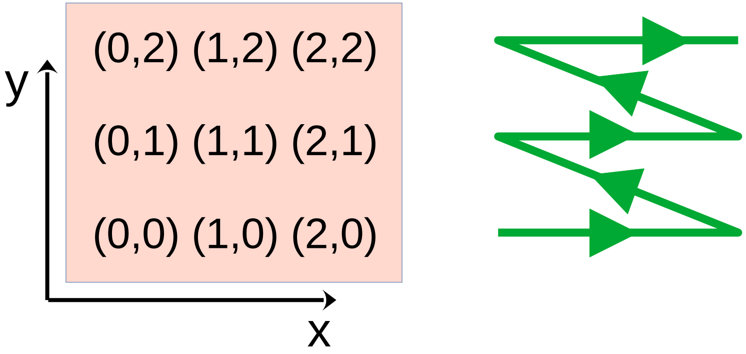
Matrix representation for finite differences

$$H\psi(x,y) = E\psi(x,y) \qquad H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

But now we have ψ(x, y): how do we organize it as a vector?

Label each point as $p_n = (x_n, y_n)$

Sort as



(0,2) (1,2) (2,2)

(0,1) (1,1) (2,1)

(0,0) (1,0) (2,0)

To get →

$p_0 = (0, 0)$
$p_1 = (1, 0)$
$p_2 = (2, 0)$
$p_3 = (0, 1)$
$p_4 = (1, 1)$
$p_5 = (2, 1)$
$p_6 = (0, 2)$
$p_7 = (1, 2)$
$p_8 = (2, 2)$

In Python… use:

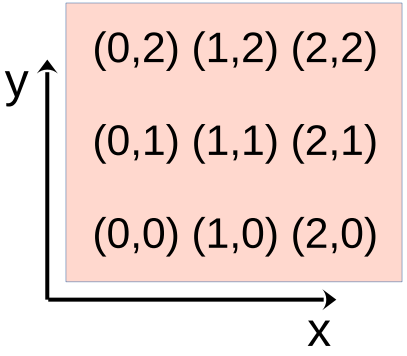
np.meshgrid(...)

Matrix representation for finite differences

$$H\psi(x,y) = E\psi(x,y) \qquad H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

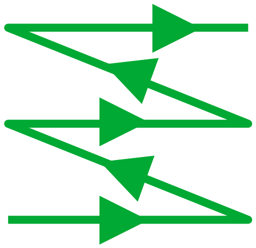
But now we have ψ(x, y): how do we organize it as a vector?

Label each point as $p_n = (x_n, y_n)$



(0,2)	(1,2)	(2,2)
(0,1)	(1,1)	(2,1)
(0,0)	(1,0)	(2,0)

Sort as



To get →

$p_0 = (0, 0)$
$p_1 = (1, 0)$
$p_2 = (2, 0)$
$p_3 = (0, 1)$
$p_4 = (1, 1)$
$p_5 = (2, 1)$
$p_6 = (0, 2)$
$p_7 = (1, 2)$
$p_8 = (2, 2)$

In Python… use:

np.meshgrid(...)

.flatten()

Matrix representation for finite differences

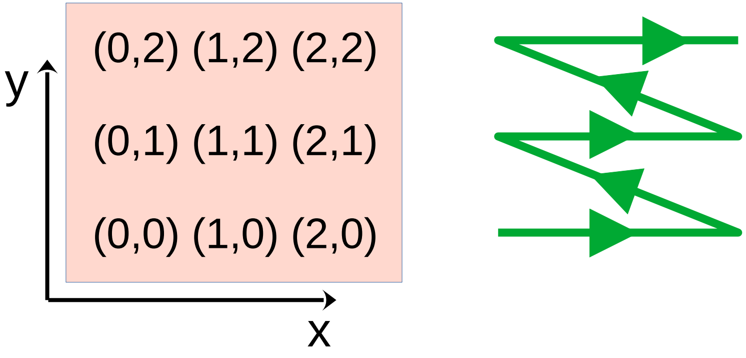
$$H\psi(x,y) = E\psi(x,y) \qquad H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x,y)$$

But now we have ψ(x, y): how do we organize it as a vector?

Label each point as $p_n = (x_n, y_n)$

Sort as

To get →



(0,2)	(1,2)	(2,2)
(0,1)	(1,1)	(2,1)
(0,0)	(1,0)	(2,0)

p_0 = (0, 0)
p_1 = (1, 0)
p_2 = (2, 0)
p_3 = (0, 1)
p_4 = (1, 1)
p_5 = (2, 1)
p_6 = (0, 2)
p_7 = (1, 2)
p_8 = (2, 2)

In Python… use:

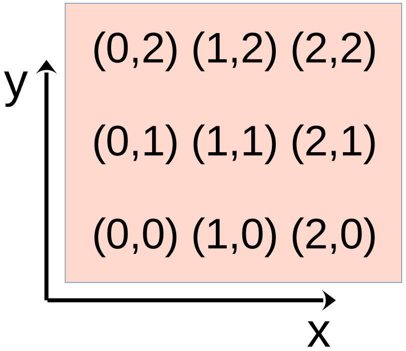
np.meshgrid(...)

.flatten()

np.kron(...)

How these python commands work? meshgrid and flatten

meshgrid: convert vectors x, y into coordinate matrices
flatten: collapses the array into one dimension



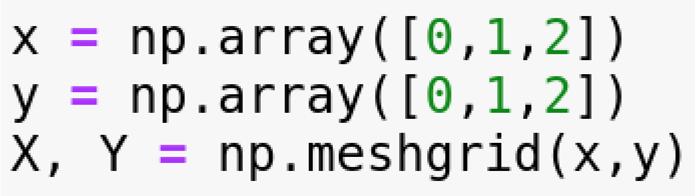
(0,2) (1,2) (2,2)

(0,1) (1,1) (2,1)

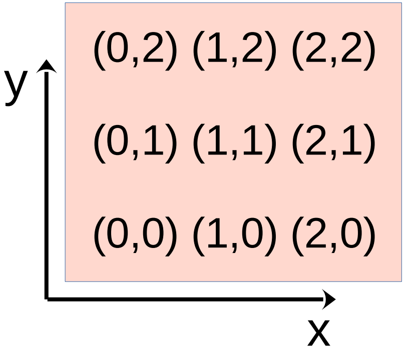
(0,0) (1,0) (2,0)

How these python commands work? meshgrid and flatten

meshgrid: convert vectors x, y into coordinate matrices
flatten: collapses the array into one dimension



```
x = np.array([0,1,2])
y = np.array([0,1,2])
X, Y = np.meshgrid(x,y)
```



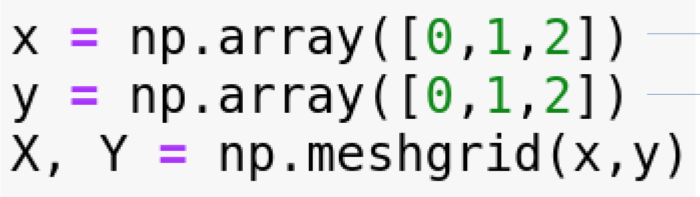
(0,2) (1,2) (2,2)

(0,1) (1,1) (2,1)

(0,0) (1,0) (2,0)

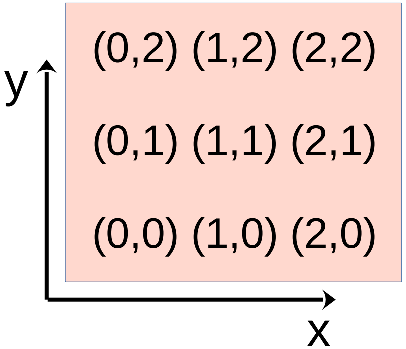
How these python commands work? meshgrid and flatten

meshgrid: convert vectors x, y into coordinate matrices
flatten: collapses the array into one dimension



```
x = np.array([0,1,2])
y = np.array([0,1,2])
X, Y = np.meshgrid(x,y)
```

vectors [0, 1, 2]



(0,2) (1,2) (2,2)

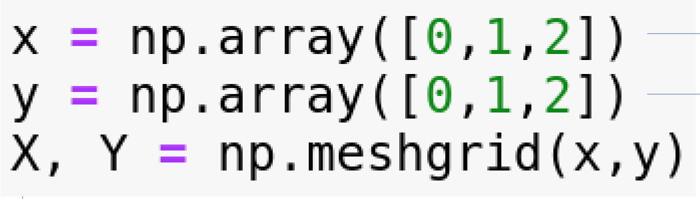
(0,1) (1,1) (2,1)

(0,0) (1,0) (2,0)

How these python commands work? meshgrid and flatten

meshgrid: convert vectors x, y into coordinate matrices

flatten: collapses the array into one dimension



```
x = np.array([0,1,2])
y = np.array([0,1,2])
X, Y = np.meshgrid(x,y)
```

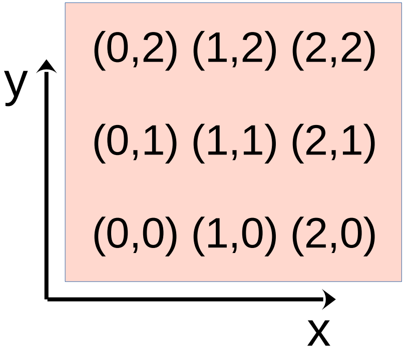
vectors [0, 1, 2]

Matrix:

x → |0,1,2|

x → |0,1,2|

x → |0,1,2|



(0,2) (1,2) (2,2)

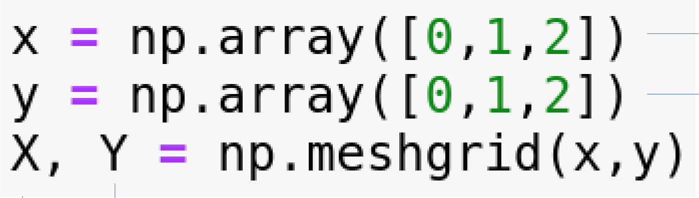
(0,1) (1,1) (2,1)

(0,0) (1,0) (2,0)

How these python commands work? meshgrid and flatten

meshgrid: convert vectors x, y into coordinate matrices

flatten: collapses the array into one dimension

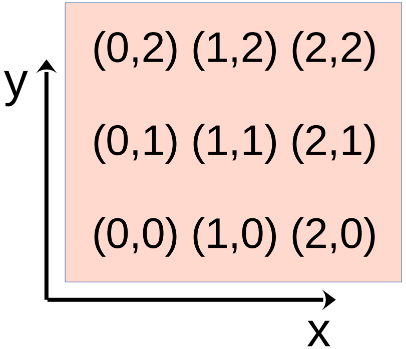


```
x = np.array([0,1,2])
y = np.array([0,1,2])
X, Y = np.meshgrid(x,y)
```

vectors [0, 1, 2]

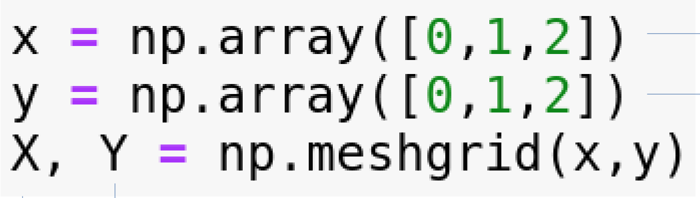
Matrix:
|0,0,0|
|1,1,1|
|2,2,2|
↑ ↑ ↑
y y y

Matrix:
x → |0,1,2|
x → |0,1,2|
x → |0,1,2|



How these python commands work? meshgrid and flatten

meshgrid: convert vectors x, y into coordinate matrices
flatten: collapses the array into one dimension



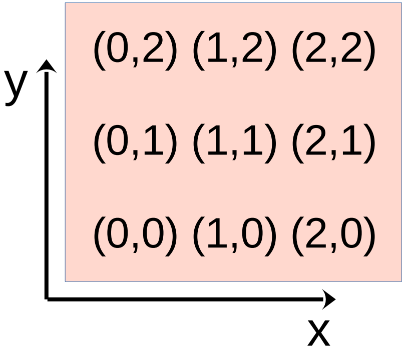
```
x = np.array([0,1,2])
y = np.array([0,1,2])
X, Y = np.meshgrid(x,y)
```

vectors [0, 1, 2]

Matrix:
x → |0,1,2|
x → |0,1,2|
x → |0,1,2|

Matrix:
|0,0,0|
|1,1,1|
|2,2,2|
↑ ↑ ↑
y y y

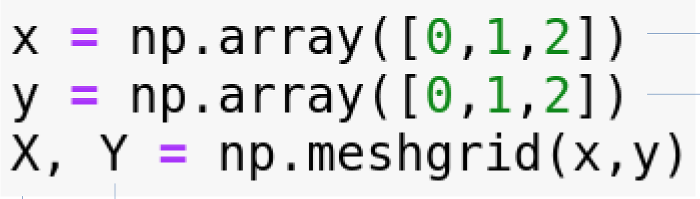
The matrix elements form the pairs $p_n = (x_n, y_n)$



INSTITUTO DE FÍSICA
Universidade Federal de Uberlândia

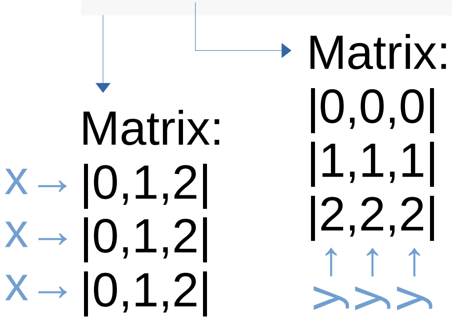
How these python commands work? meshgrid and flatten

meshgrid: convert vectors x, y into coordinate matrices
flatten: collapses the array into one dimension



```
x = np.array([0,1,2])
y = np.array([0,1,2])
X, Y = np.meshgrid(x,y)
```

vectors [0, 1, 2]

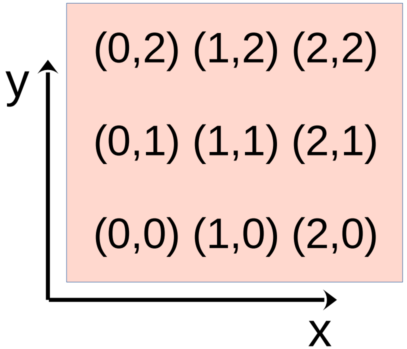


Matrix:
x → |0,1,2|
x → |0,1,2|
x → |0,1,2|

Matrix:
|0,0,0|
|1,1,1|
|2,2,2|
↑ ↑ ↑

The matrix elements form the pairs $p_n = (x_n, y_n)$

To linearize into vectors, use .flatten()



(0,2) (1,2) (2,2)

(0,1) (1,1) (2,1)

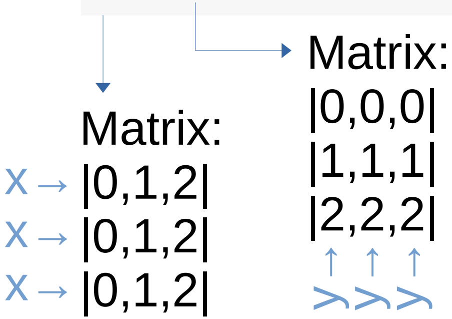
(0,0) (1,0) (2,0)

How these python commands work? meshgrid and flatten

meshgrid: convert vectors x, y into coordinate matrices
flatten: collapses the array into one dimension

```
x = np.array([0,1,2])
y = np.array([0,1,2])
X, Y = np.meshgrid(x,y)
```

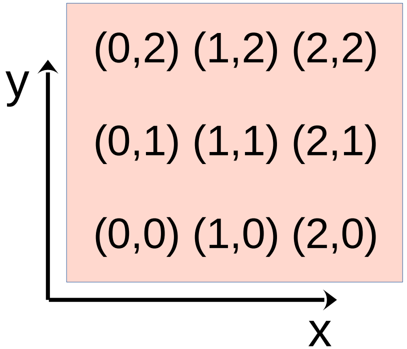
vectors [0, 1, 2]



The matrix elements form the pairs $p_n = (x_n, y_n)$

To linearize into vectors, use .flatten()

```
X = X.flatten()
Y = Y.flatten()
```



How these python commands work? meshgrid and flatten

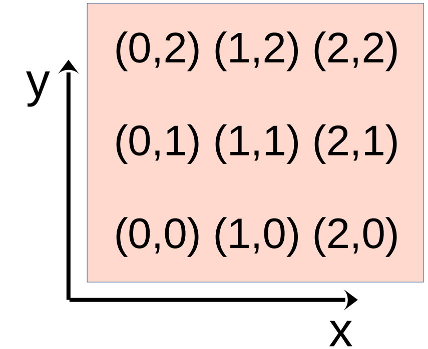
meshgrid: convert vectors x, y into coordinate matrices
flatten: collapses the array into one dimension

```
x = np.array([0,1,2])
y = np.array([0,1,2])
X, Y = np.meshgrid(x,y)
```

vectors [0, 1, 2]

Matrix:
x → |0,1,2|
x → |0,1,2|
x → |0,1,2|

Matrix:
|0,0,0|
|1,1,1|
|2,2,2|
↑ ↑ ↑
y y y



The matrix elements form the pairs $p_n = (x_n, y_n)$

To linearize into vectors, use .flatten()

```
X = X.flatten()
Y = Y.flatten()
```

0
1
2
0
1
2
0
1
2

How these python commands work? meshgrid and flatten

meshgrid: convert vectors x, y into coordinate matrices

flatten: collapses the array into one dimension

```
x = np.array([0,1,2])
y = np.array([0,1,2])
X, Y = np.meshgrid(x,y)
```

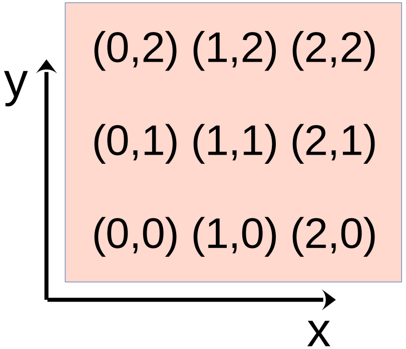
vectors [0, 1, 2]

Matrix:

x → |0,1,2|
x → |0,1,2|
x → |0,1,2|

Matrix:

|0,0,0|
|1,1,1|
|2,2,2|
↑ ↑ ↑
> > >



(0,2) (1,2) (2,2)

(0,1) (1,1) (2,1)

(0,0) (1,0) (2,0)

The matrix elements form the pairs $p_n = (x_n, y_n)$

To linearize into vectors, use .flatten()

```
X = X.flatten()
Y = Y.flatten()
```

0 0
1 0
2 0
0 1
1 1
2 1
0 2
1 2
2 2

How these python commands work? meshgrid and flatten

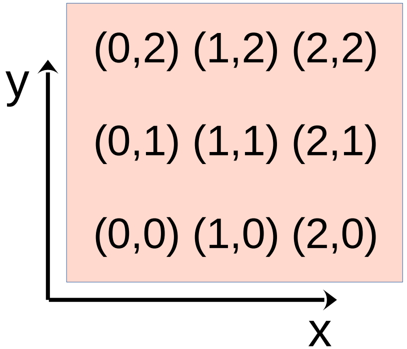
meshgrid: convert vectors x, y into coordinate matrices
flatten: collapses the array into one dimension

```
x = np.array([0,1,2])
y = np.array([0,1,2])
X, Y = np.meshgrid(x,y)
```

vectors [0, 1, 2]

Matrix:
x → |0,1,2|
x → |0,1,2|
x → |0,1,2|

Matrix:
|0,0,0|
|1,1,1|
|2,2,2|
↑ ↑ ↑
y y y



The matrix elements form the pairs $p_n = (x_n, y_n)$

To linearize into vectors, use .flatten()

```
X = X.flatten()
Y = Y.flatten()
```

X	Y	$p_n = (x_n, y_n)$
0	0	(0, 0)
1	0	(1, 0)
2	0	(2, 0)
0	1	(0, 1)
1	1	(1, 1)
2	1	(2, 1)
0	2	(0, 2)
1	2	(1, 2)
2	2	(2, 2)

INSTITUTO DE FÍSICA
Universidade Federal de Uberlândia

How these python commands work? meshgrid and flatten

meshgrid: convert vectors x, y into coordinate matrices

flatten: collapses the array into one dimension

```
x = np.array([0,1,2])
y = np.array([0,1,2])
X, Y = np.meshgrid(x,y)
```

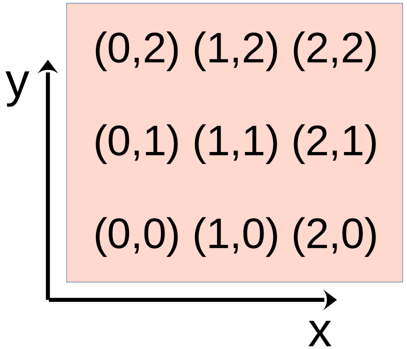
vectors [0, 1, 2]

Matrix:

x → |0,1,2|
x → |0,1,2|
x → |0,1,2|

Matrix:

|0,0,0|
|1,1,1|
|2,2,2|
↑ ↑ ↑
y y y



(0,2) (1,2) (2,2)
(0,1) (1,1) (2,1)
(0,0) (1,0) (2,0)

The matrix elements form the pairs $p_n = (x_n, y_n)$

To linearize into vectors, use .flatten()

```
X = X.flatten()
Y = Y.flatten()
```

		$p_n = (x_n, y_n)$
0	0	(0, 0)
1	0	(1, 0)
2	0	(2, 0)
0	1	(0, 1)
1	1	(1, 1)
2	1	(2, 1)
0	2	(0, 2)
1	2	(1, 2)
2	2	(2, 2)

Now check this:

```
V = 0.5*(X**2 + Y**2)
```

→ is this useful? ;-)

How these python commands work? meshgrid and flatten

meshgrid: convert vectors x, y into coordinate matrices
flatten: collapses the array into one dimension

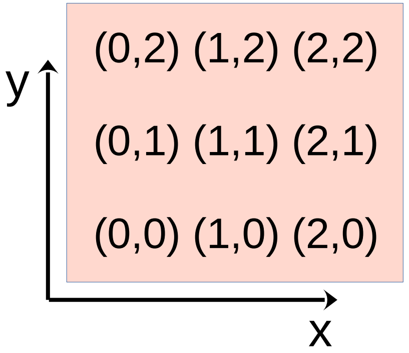
```
x = np.array([0,1,2])
y = np.array([0,1,2])
X, Y = np.meshgrid(x,y)
```

→ vectors [0, 1, 2]

You can also use kron! see next slides

Matrix:
x → |0,1,2|
x → |0,1,2|
x → |0,1,2|

Matrix:
|0,0,0|
|1,1,1|
|2,2,2|
↑ ↑ ↑
y y y



(0,2)	(1,2)	(2,2)
(0,1)	(1,1)	(2,1)
(0,0)	(1,0)	(2,0)

The matrix elements form the pairs $p_n = (x_n, y_n)$

To linearize into vectors, use .flatten()

```
X = X.flatten()
Y = Y.flatten()
```

X	Y	$p_n = (x_n, y_n)$
0	0	(0, 0)
1	0	(1, 0)
2	0	(2, 0)
0	1	(0, 1)
1	1	(1, 1)
2	1	(2, 1)
0	2	(0, 2)
1	2	(1, 2)
2	2	(2, 2)

Now check this:

```
V = 0.5*(X**2 + Y**2)
```

→ is this useful? ;-)

INSTITUTO DE FÍSICA
Universidade Federal de Uberlândia

How these python commands work? kron

kron: is the Kronecker product, or the direct product → ⊗

$p_n = (x_n, y_n)$

(0, 0)
(1, 0)
(2, 0)
(0, 1)
(1, 1)
(2, 1)
(0, 2)
(1, 2)
(2, 2)

How these python commands work? kron

kron: is the Kronecker product, or the direct product → ⊗

If A and B are matrices:

$p_n = (x_n, y_n)$

(0, 0)
(1, 0)
(2, 0)
(0, 1)
(1, 1)
(2, 1)
(0, 2)
(1, 2)
(2, 2)

How these python commands work? kron

kron: is the Kronecker product, or the direct product → ⊗

If A and B are matrices:

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{pmatrix} \rightarrow A \otimes B = \begin{pmatrix} a_{0,0}B & a_{0,1}B \\ a_{1,0}B & a_{1,1}B \end{pmatrix}$$

$p_n = (x_n, y_n)$

(0, 0)
(1, 0)
(2, 0)
(0, 1)
(1, 1)
(2, 1)
(0, 2)
(1, 2)
(2, 2)

INSTITUTO DE FÍSICA
Universidade Federal de Uberlândia

How these python commands work? kron

kron: is the Kronecker product, or the direct product → ⊗

If A and B are matrices:

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{pmatrix} \rightarrow A \otimes B = \begin{pmatrix} a_{0,0}B & a_{0,1}B \\ a_{1,0}B & a_{1,1}B \end{pmatrix}$$

Consider now the 1D matrix representations for these operators

$p_n = (x_n, y_n)$
(0, 0)
(1, 0)
(2, 0)
(0, 1)
(1, 1)
(2, 1)
(0, 2)
(1, 2)
(2, 2)

INSTITUTO DE FÍSICA
Universidade Federal de Uberlândia

How these python commands work? kron

kron: is the Kronecker product, or the direct product → ⊗

If A and B are matrices:

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{pmatrix} \rightarrow A \otimes B = \begin{pmatrix} a_{0,0}B & a_{0,1}B \\ a_{1,0}B & a_{1,1}B \end{pmatrix}$$

Consider now the 1D matrix representations for these operators (see previous classes)

$p_n = (x_n, y_n)$

(0, 0)
(1, 0)
(2, 0)
(0, 1)
(1, 1)
(2, 1)
(0, 2)
(1, 2)
(2, 2)

INSTITUTO DE FÍSICA
Universidade Federal de Uberlândia

How these python commands work? kron

kron: is the Kronecker product, or the direct product → ⊗

If A and B are matrices:

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{pmatrix} \rightarrow A \otimes B = \begin{pmatrix} a_{0,0}B & a_{0,1}B \\ a_{1,0}B & a_{1,1}B \end{pmatrix}$$

Consider now the 1D matrix representations for these operators (see previous classes)

$$D_x = \partial_x^2$$

$$I_x = \text{ identity } N_x \times N_x$$

$p_n = (x_n, y_n)$

(0, 0)
(1, 0)
(2, 0)
(0, 1)
(1, 1)
(2, 1)
(0, 2)
(1, 2)
(2, 2)

How these python commands work? kron

kron: is the Kronecker product, or the direct product → ⊗

If A and B are matrices:

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{pmatrix} \to A \otimes B = \begin{pmatrix} a_{0,0}B & a_{0,1}B \\ a_{1,0}B & a_{1,1}B \end{pmatrix}$$

Consider now the 1D matrix representations for these operators (see previous classes)

$$D_x = \partial_x^2$$

$$I_x = \text{ identity } N_x \times N_x$$

$$D_y = \partial_y^2$$

$$I_y = \text{ identity } N_y \times N_y$$

$p_n = (x_n, y_n)$
(0, 0)
(1, 0)
(2, 0)
(0, 1)
(1, 1)
(2, 1)
(0, 2)
(1, 2)
(2, 2)

INSTITUTO DE FÍSICA
Universidade Federal de Uberlândia

How these python commands work? kron

kron: is the Kronecker product, or the direct product → ⊗

If A and B are matrices:

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{pmatrix} \rightarrow A \otimes B = \begin{pmatrix} a_{0,0}B & a_{0,1}B \\ a_{1,0}B & a_{1,1}B \end{pmatrix}$$

Consider now the 1D matrix representations for these operators (see previous classes)

$$\left.\begin{aligned} D_x &= \partial_x^2 \\ I_x &= \text{ identity } N_x \times N_x \\ \\ D_y &= \partial_y^2 \\ I_y &= \text{ identity } N_y \times N_y \end{aligned}\right\}$$

The 2D version becomes:

$p_n = (x_n, y_n)$
(0, 0)
(1, 0)
(2, 0)
(0, 1)
(1, 1)
(2, 1)
(0, 2)
(1, 2)
(2, 2)

How these python commands work? kron

kron: is the Kronecker product, or the direct product → ⊗

If A and B are matrices:

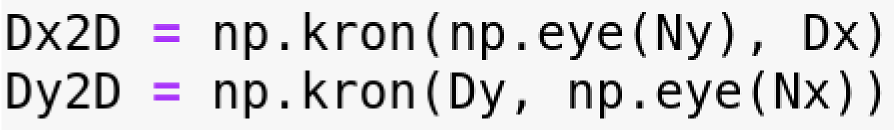
$$A = \begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{pmatrix} \rightarrow A \otimes B = \begin{pmatrix} a_{0,0}B & a_{0,1}B \\ a_{1,0}B & a_{1,1}B \end{pmatrix}$$

$p_n = (x_n, y_n)$
(0, 0)
(1, 0)
(2, 0)
(0, 1)
(1, 1)
(2, 1)
(0, 2)
(1, 2)
(2, 2)

Consider now the 1D matrix representations for these operators
(see previous classes)

$$\left.\begin{aligned} D_x &= \partial_x^2 \\ I_x &= \text{ identity } N_x \times N_x \\ D_y &= \partial_y^2 \\ I_y &= \text{ identity } N_y \times N_y \end{aligned}\right\}$$

The 2D version becomes:



```
Dx2D = np.kron(np.eye(Ny), Dx)
Dy2D = np.kron(Dy, np.eye(Nx))
```

How these python commands work? kron

kron: is the Kronecker product, or the direct product → ⊗

If A and B are matrices:

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{pmatrix} \rightarrow A \otimes B = \begin{pmatrix} a_{0,0}B & a_{0,1}B \\ a_{1,0}B & a_{1,1}B \end{pmatrix}$$

$p_n = (x_n, y_n)$
(0, 0)
(1, 0)
(2, 0)
(0, 1)
(1, 1)
(2, 1)
(0, 2)
(1, 2)
(2, 2)

Consider now the 1D matrix representations for these operators (see previous classes)

$$\left.\begin{aligned} D_x &= \partial_x^2 \\ I_x &= \text{ identity } N_x \times N_x \\ D_y &= \partial_y^2 \\ I_y &= \text{ identity } N_y \times N_y \end{aligned}\right\}$$

The 2D version becomes:

```
Dx2D = np.kron(np.eye(Ny), Dx)
Dy2D = np.kron(Dy, np.eye(Nx))
```

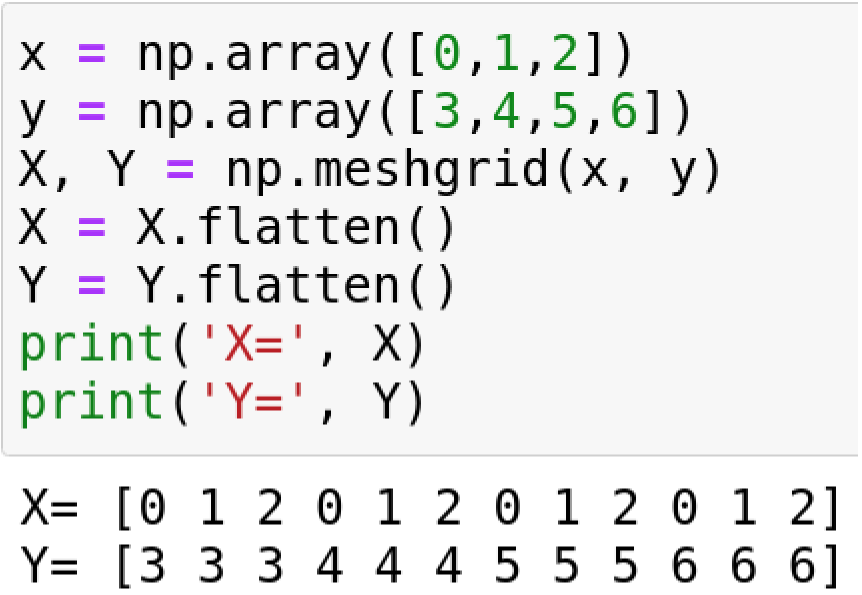
So we get:

```
H = -0.5*(Dx2D + Dy2D) + V
```

Using kron instead of meshgrid and flatten

Check that these codes return the same vectors:

Using meshgrid/flatten



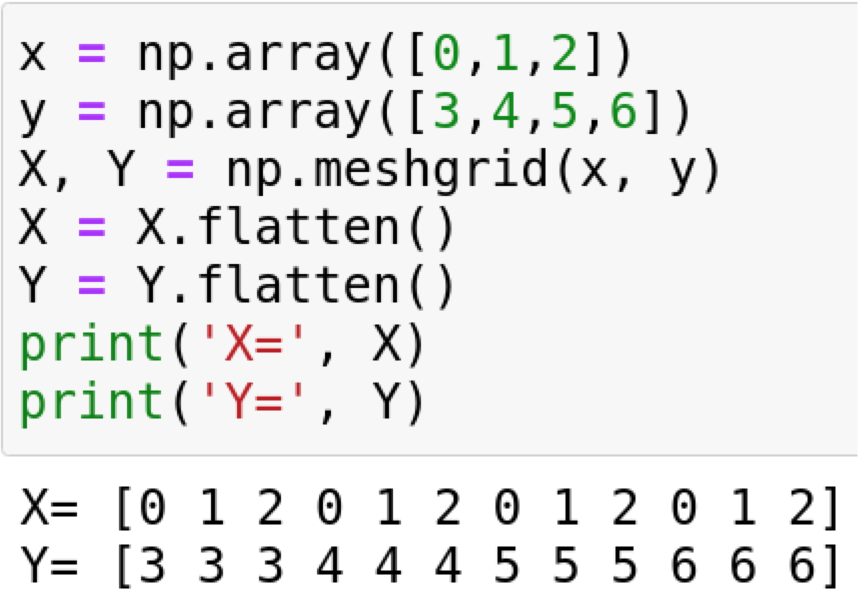
```
x = np.array([0,1,2])
y = np.array([3,4,5,6])
X, Y = np.meshgrid(x, y)
X = X.flatten()
Y = Y.flatten()
print('X=', X)
print('Y=', Y)
```

```
X= [0 1 2 0 1 2 0 1 2 0 1 2]
Y= [3 3 3 4 4 4 5 5 5 6 6 6]
```

Using kron instead of meshgrid and flatten

Check that these codes return the same vectors:

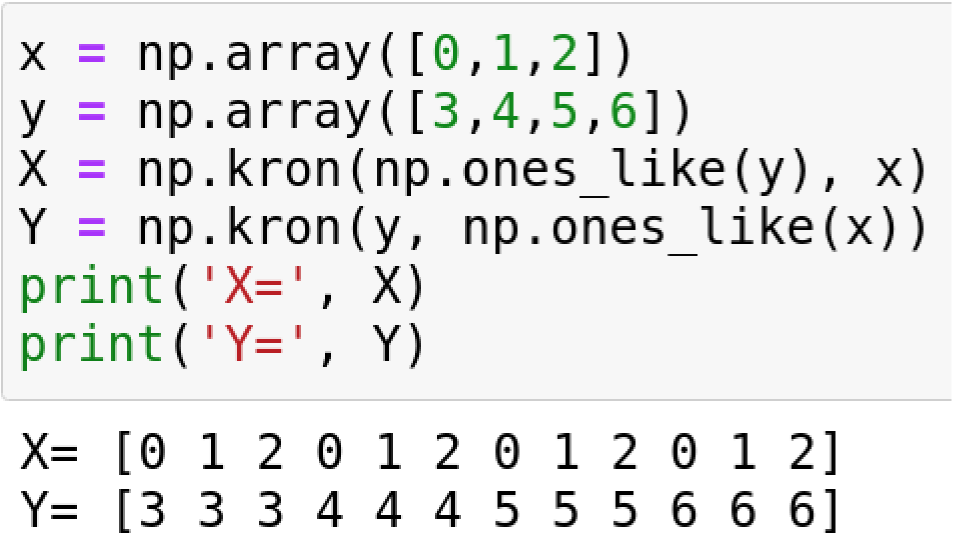
Using meshgrid/flatten



```
x = np.array([0,1,2])
y = np.array([3,4,5,6])
X, Y = np.meshgrid(x, y)
X = X.flatten()
Y = Y.flatten()
print('X=', X)
print('Y=', Y)
```

```
X= [0 1 2 0 1 2 0 1 2 0 1 2]
Y= [3 3 3 4 4 4 5 5 5 6 6 6]
```

Using kron



```
x = np.array([0,1,2])
y = np.array([3,4,5,6])
X = np.kron(np.ones_like(y), x)
Y = np.kron(y, np.ones_like(x))
print('X=', X)
print('Y=', Y)
```

```
X= [0 1 2 0 1 2 0 1 2 0 1 2]
Y= [3 3 3 4 4 4 5 5 5 6 6 6]
```

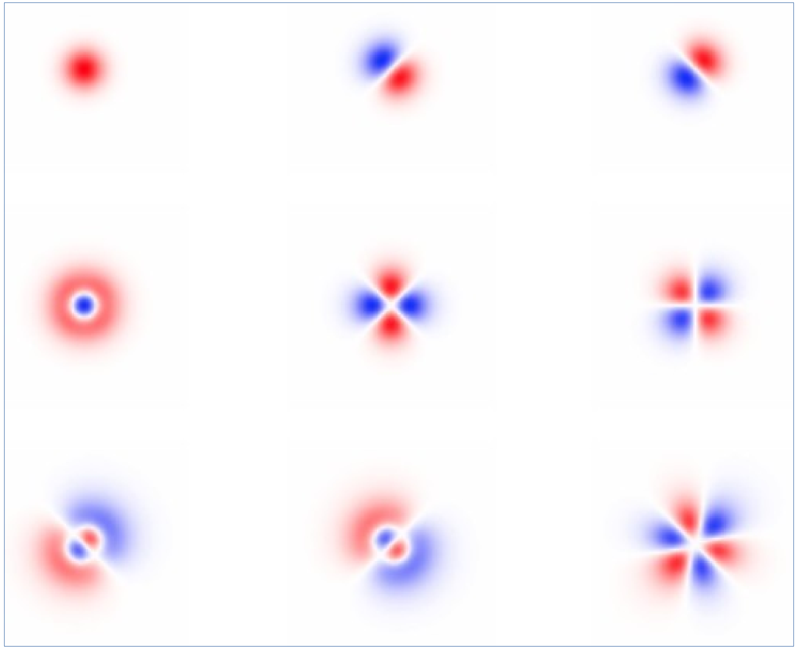
Results for 2D atoms or molecules

Atomic potential = inverse Gaussian

Results for 2D atoms or molecules

Atomic potential = inverse Gaussian

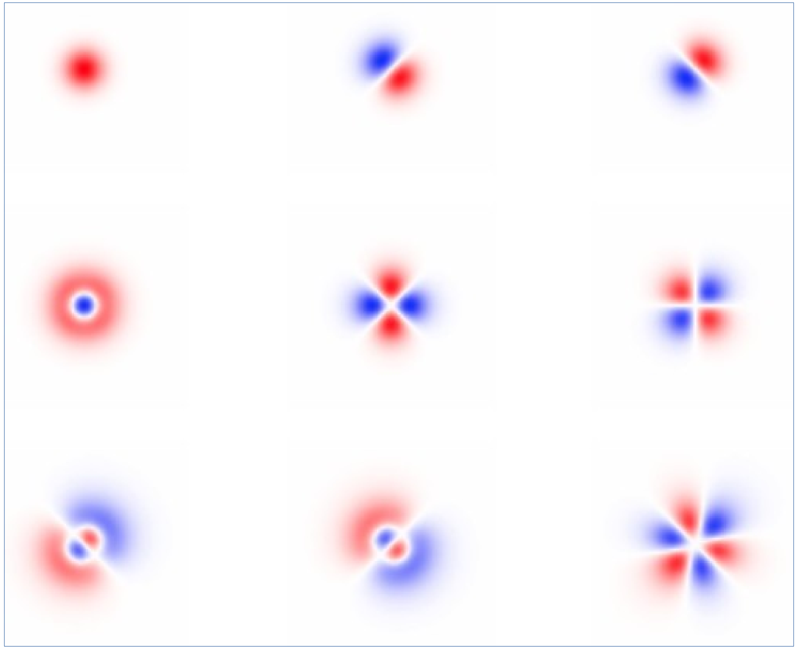
1 atom



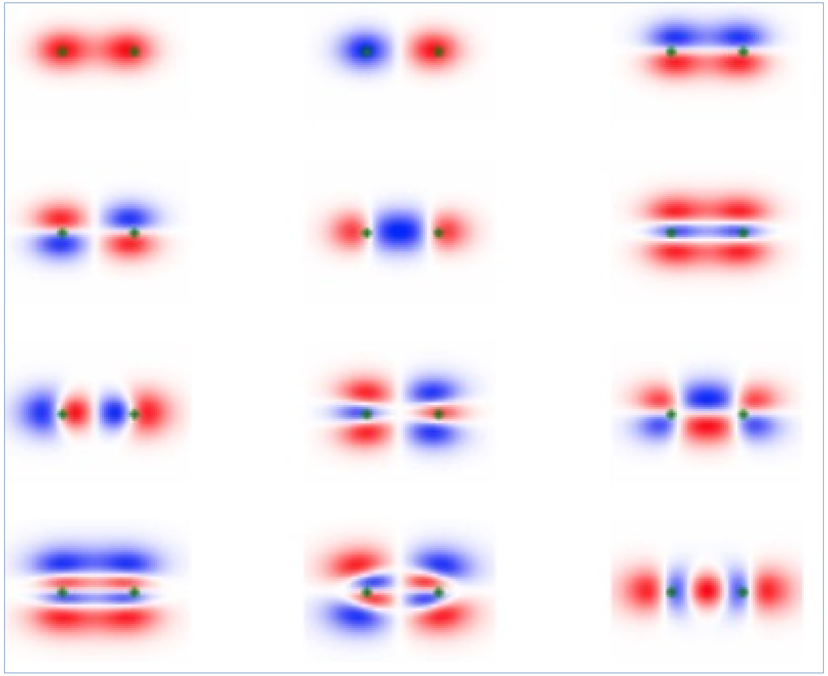
Results for 2D atoms or molecules

Atomic potential = inverse Gaussian

1 atom

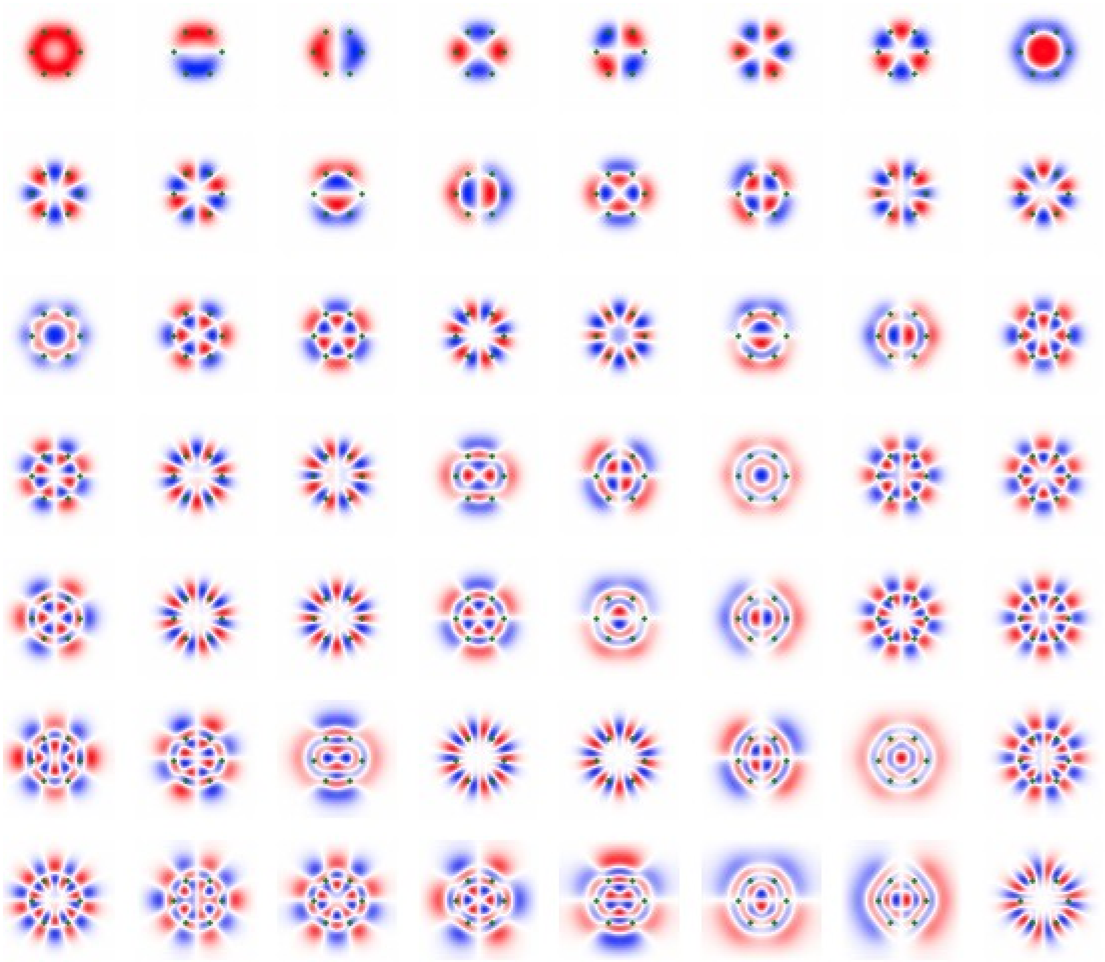


2 atoms



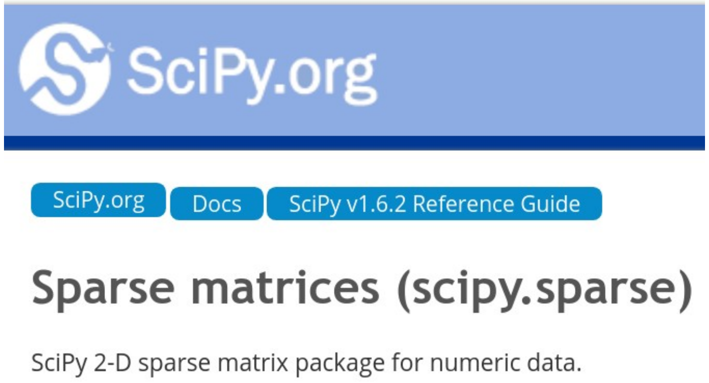
Results for 2D atoms or molecules

6 atoms
hexagon



How to improve? For large matrices: use scipy sparse matrices

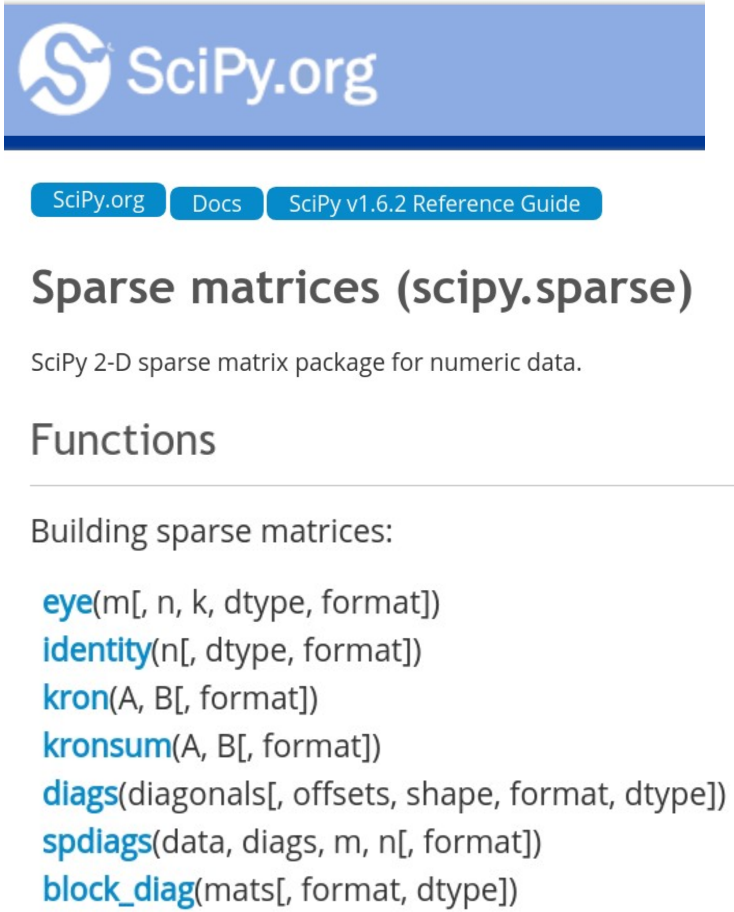
How to improve? For large matrices: use scipy sparse matrices



Sparse matrices (scipy.sparse)

SciPy 2-D sparse matrix package for numeric data.

How to improve? For large matrices: use scipy sparse matrices



SciPy.org

SciPy.org | Docs | SciPy v1.6.2 Reference Guide

Sparse matrices (scipy.sparse)

SciPy 2-D sparse matrix package for numeric data.

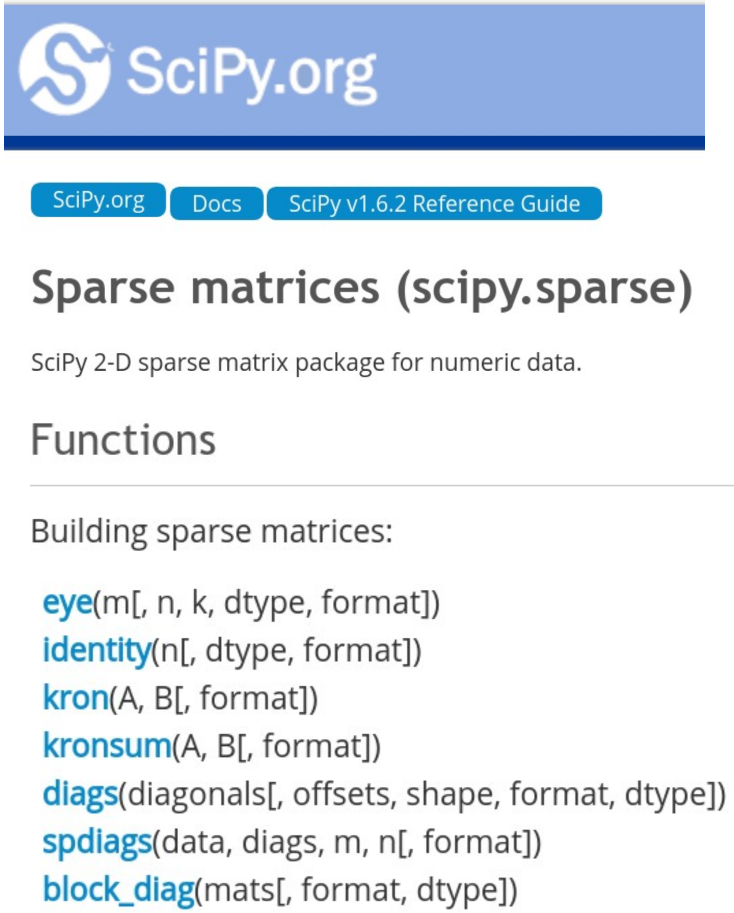
Functions

Building sparse matrices:

- eye(m[, n, k, dtype, format])
- identity(n[, dtype, format])
- kron(A, B[, format])
- kronsum(A, B[, format])
- diags(diagonals[, offsets, shape, format, dtype])
- spdiags(data, diags, m, n[, format])
- block_diag(mats[, format, dtype])

INSTITUTO DE FÍSICA
Universidade Federal de Uberlândia

How to improve? For large matrices: use scipy sparse matrices



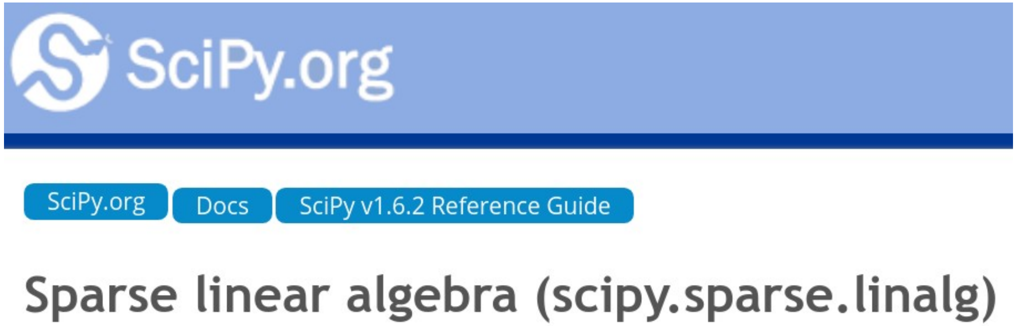
Sparse matrices (scipy.sparse)

SciPy 2-D sparse matrix package for numeric data.

Functions

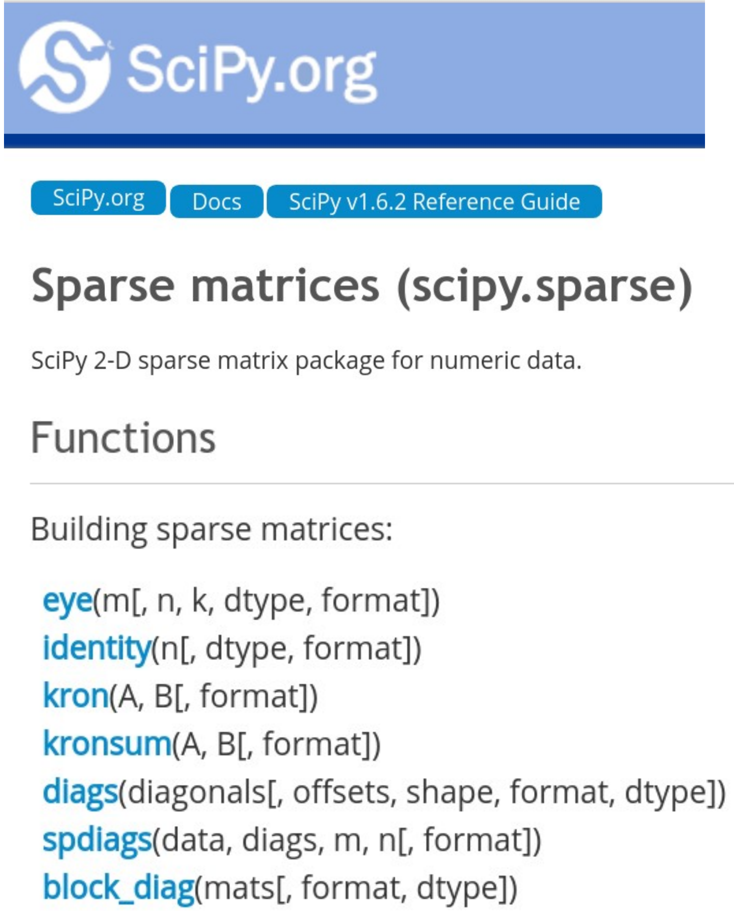
Building sparse matrices:

- **eye**(m[, n, k, dtype, format])
- **identity**(n[, dtype, format])
- **kron**(A, B[, format])
- **kronsum**(A, B[, format])
- **diags**(diagonals[, offsets, shape, format, dtype])
- **spdiags**(data, diags, m, n[, format])
- **block_diag**(mats[, format, dtype])



Sparse linear algebra (scipy.sparse.linalg)

How to improve? For large matrices: use scipy sparse matrices



SciPy.org

SciPy.org | Docs | SciPy v1.6.2 Reference Guide

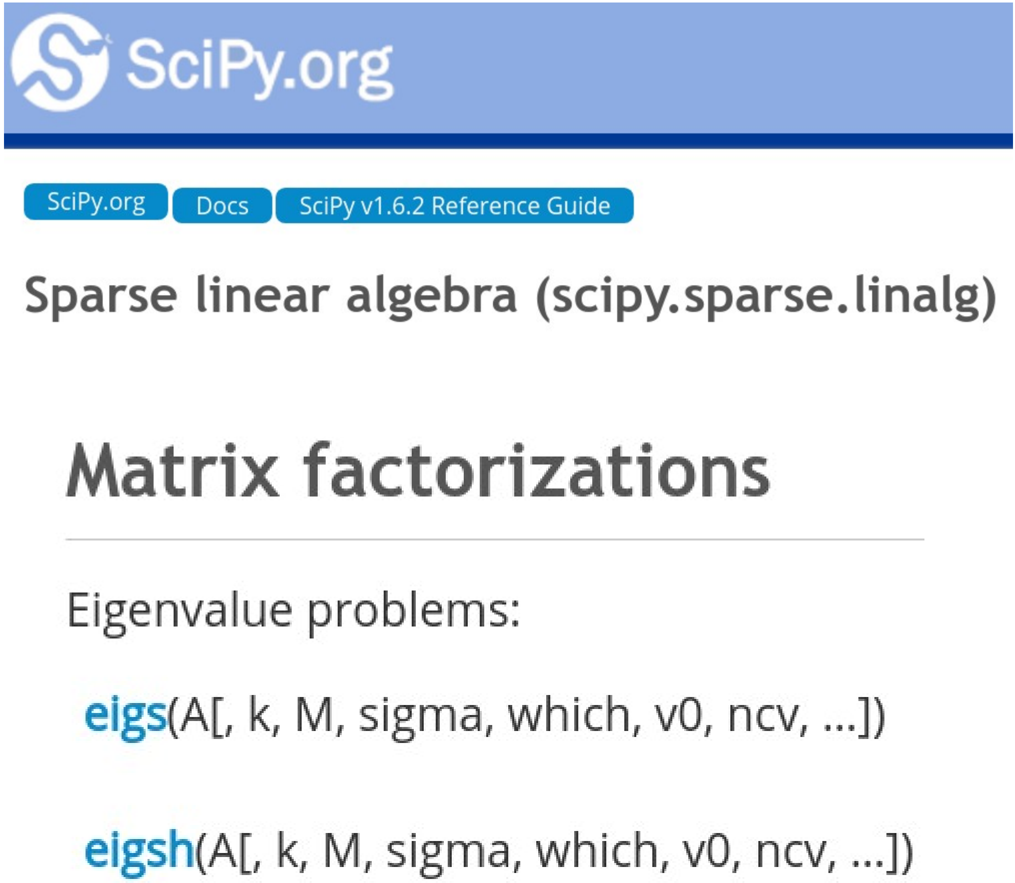
Sparse matrices (scipy.sparse)

SciPy 2-D sparse matrix package for numeric data.

Functions

Building sparse matrices:

- eye(m[, n, k, dtype, format])
- identity(n[, dtype, format])
- kron(A, B[, format])
- kronsum(A, B[, format])
- diags(diagonals[, offsets, shape, format, dtype])
- spdiags(data, diags, m, n[, format])
- block_diag(mats[, format, dtype])



SciPy.org

SciPy.org | Docs | SciPy v1.6.2 Reference Guide

Sparse linear algebra (scipy.sparse.linalg)

Matrix factorizations

Eigenvalue problems:

- eigs(A[, k, M, sigma, which, v0, ncv, ...])
- eigsh(A[, k, M, sigma, which, v0, ncv, ...])