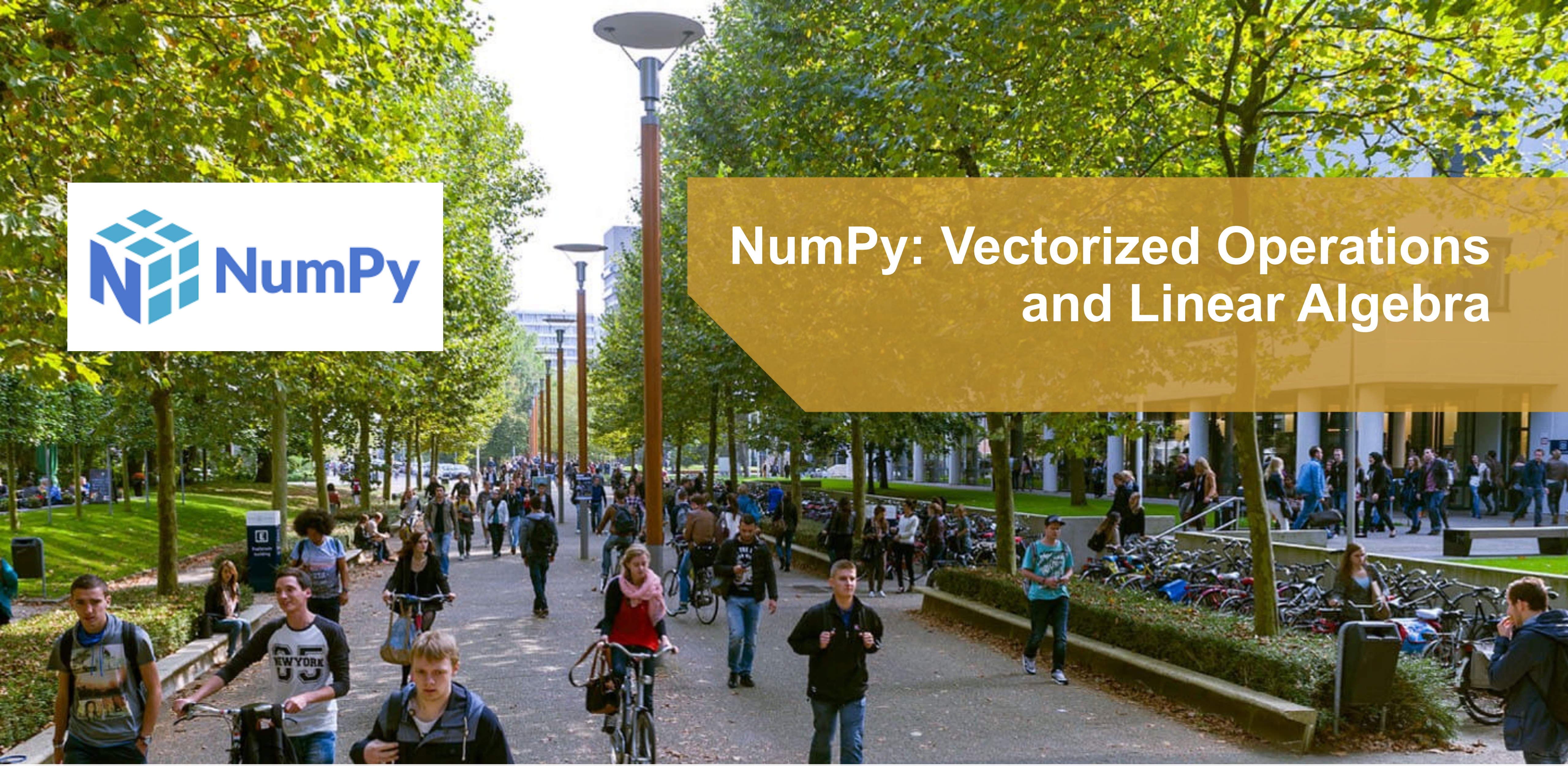
NumPy: Vectorized Operations and Linear Algebra



What is a vectorized operation in NumPy?

❖ A vectorized operation is an operation that can be evaluated on all elements of an array at once instead of looping over individual entries.

NumPy Array

```
a = numpy.random.random((100000))
b = 2
c = a + b
```

Python List

```
a = numpy.random.random((100000)).tolist()
b = 2
c = []
for aa in a:
    c.append(aa + b)
```

What is a vectorized operation in NumPy?

- A vectorized operation is an operation that can be evaluated on all elements of an array at once instead of looping over individual entries.
- The main advantage of vectorized operation is speed in computation. Example: adding a number to an array:

NumPy Array

```
a = numpy.random.random((100000))
b = 2
c = a + b
```

```
1.65 ms ± 50.4 µs per loop (mean ± std.
```

Python List

```
a = numpy.random.random((100000)).tolist()
b = 2
c = []
for aa in a:
    c.append(aa + b)
```

```
28.2 ms ± 1.8 ms per loop (mean ± std. dev.
```

All NumPy functions are implemented as vectorized operations.

Vectorized Operations in NumPy

There are in general three types of vectorized operations in NumPy:

1) Vectorized Transformation: that refers to vectorized operations that are applied to an array and produce an array of the same shape.
    a) Simple scalar transformations
    b) Boolean operations
    c) Mathematical functions

Vectorized Operations in NumPy

There are in general three types of vectorized operations in NumPy:

1) Vectorized Transformations: that refers to vectorized operations that are applied to an array and produce an array of the same shape.
   a) Simple scalar transformations
   b) Boolean operations
   c) Mathematical functions
2) Vectorized Dimension Reduction: that reduces the dimensionality of the input array.

Questions?

Vectorized Operations in NumPy

There are in general three types of vectorized operations in NumPy:

1) Vectorized Transformations: that refers to vectorized operations that are applied to an array and produce an array of the same shape.
   a) Simple scalar transformations
   b) Boolean operations
   c) Mathematical functions
2) Vectorized Dimension Reduction: that reduces the dimensionality of the input array.
3) Vectorized Linear Algebra Operations

Linear Algebra Operations in NumPy

- Element-wise operations
  - On arrays with the same shape

$$A^{n \times m} + B^{n \times m} = C^{n \times m}$$

$$A^{n \times m} - B^{n \times m} = D^{n \times m}$$

$$A^{n \times m} * B^{n \times m} = E^{n \times m}$$

$$A^{n \times m} / B^{n \times m} = F^{n \times m}$$

Linear Algebra Operations in NumPy

- Element-wise operations
  - On arrays with the same shape
  - On arrays with different shape (using broadcasting)

But what is broadcasting?

Broadcasting in Numpy

- Broadcasting enables NumPy to handle arrays with different shapes during vectorized operations.
- Broadcasting conditions (being "broadcastable"):
  - Array dimensions must be equal, or
  - for non-equal dimensions, either of them must be 1.

A (3d array): 5 x 4 x 3
B (3d array): 5 x 4 x 3 ✓

A (3d array): 1 x 4 x 3
B (3d array): 5 x 4 x 3 ✓

A (3d array): 5 x 4 x 3
B (2d array): 4 x 3 ?

A (3d array): 5 x 4 x 3
B (3d array): 4 x 5 x 3 ✗

A (3d array): 5 x 1 x 3
B (3d array): 1 x 4 x 1 ✓

A (3d array): 5 x 4 x 3
B (2d array): 5 x 4 ?

Broadcasting in Numpy

- Broadcasting is performed in five steps:
  - i. **Shift:** If the number of dimensions is different, shift the array with smaller dimension to the right.
  - ii. **Match:** Add 1 to the left so that the arrays have the same number of dimensions.
  - iii. **Check:** Check if they are broadcastable: if yes continue to the next step.
  - iv. **Stretch:** “copy” the single dimensions to match dimensions between the arrays.
  - v. **Perform** the element-wise operation.

A (3d array): 5 x 4 x 3
B (2d array): 4 x 3

Shift →

A (3d array): 5 x 4 x 3
B (2d array): 4 x 3

Match →

A (3d array): 5 x 4 x 3
B (3d array): 1 x 4 x 3

Check ✓

Stretch ↓

A (3d array): 5 x 4 x 3
B (3d array): 5 x 4 x 3

Broadcasting in Numpy

❖ Broadcasting is performed in five steps:

i. **Shift:** If the number of dimensions is different, shift the array with smaller dimension to the right.

ii. **Match:** Add 1 to the left so that the arrays have the same number of dimensions.

iii. **Check:** Check if they are broadcastable: if yes continue to the next step.

iv. **Stretch:** “copy”* all the single dimensions to match dimensions between the arrays.

v. **Perform** the element-wise operation.

A (3d array): 5 x 4 x 3
B (2d array): 5 x 4

Shift →

A (3d array): 5 x 4 x 3
B (2d array):   5 x 4

Match →

A (3d array): 5 x 4 x 3
B (3d array): 1 x 5 x 4

Check ✗

* In the stretch step, no actual copy occurs in the memory, it can be considered a virtual copy. This is why we call it stretch and not copy.

Questions?

Linear Algebra Operations in NumPy

- Element-wise operations
  - On arrays with the same shape
  - On arrays with different shape (using broadcasting)
- Dot product
  - Between two vectors

Dot Product Between Two Vectors

❖ The dot product is the sum of the products of the corresponding entries of two vectors.

$$\begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = 1 \times 4 + 2 \times 5 + 3 \times 6 = 32$$

Linear Algebra Operations in NumPy

- Element-wise operations
  - On arrays with the same shape
  - On arrays with different shape (using broadcasting)
- Dot product
  - Between two vectors
  - Between two matrices

Matrix Dot Product

$$\boldsymbol{A}^{\boldsymbol{n}\times \mathrm{m}} \cdot \boldsymbol{B}^{\boldsymbol{m}\times\boldsymbol{p}} = \boldsymbol{C}^{\boldsymbol{n}\times\boldsymbol{p}}$$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mp} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{np} \end{bmatrix}$$

$$a_{11}b_{11} + a_{12}b_{21} + \cdots + a_{1m}b_{m1} = c_{11}$$

Matrix Dot Product

$$\boldsymbol{A}^{\boldsymbol{n}\times\mathrm{m}} \cdot \boldsymbol{B}^{\boldsymbol{m}\times\boldsymbol{p}} = \boldsymbol{C}^{\boldsymbol{n}\times\boldsymbol{p}}$$

$$\begin{bmatrix} \boxed{a_{11} & a_{12} & \cdots & a_{1m}} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mp} \end{bmatrix} = \begin{bmatrix} c_{11} & \boxed{c_{12}} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{np} \end{bmatrix}$$

$$a_{11}b_{12} + a_{12}b_{22} + \cdots + a_{1m}b_{m2} = c_{12}$$

Matrix Dot Product

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}\begin{bmatrix}1 & 2\\3 & 4\\5 & 6\end{bmatrix}=\begin{bmatrix}1\times1+2\times3+3\times5 & 1\times2+2\times4+3\times6\\4\times1+5\times3+6\times5 & 4\times2+5\times4+6\times6\end{bmatrix}=\begin{bmatrix}22 & 28\\49 & 64\end{bmatrix}$$

```
A = numpy.array([[1,2,3],[4,5,6]])
B = numpy.array([[1,2],[3,4],[5,6]])
C = A.dot(B)
print(C)
```

```
[[22 28]
 [49 64]]
```

Questions?

Linear Algebra Operations in NumPy

- Element-wise operations
  - On arrays with the same shape
  - On arrays with different shape (using broadcasting)
- Dot product
  - Between two vectors
  - Between two matrices
- Matrix transpose

Matrix Transpose

- The transpose of a matrix (or a vector) switches the row and column indices of the matrix $\boldsymbol{A}$, often denoted by $\boldsymbol{A}^{\boldsymbol{T}}$.

```
A = numpy.array([[1,2],[3,4],[5,6]])
print(A)
print('#################')
print(A.T)
```

```
[[1 2]
 [3 4]
 [5 6]]
#################
[[1 3 5]
 [2 4 6]]
```

Linear Algebra Operations in NumPy

- Element-wise operations
  - On arrays with the same shape
  - On arrays with different shape (using broadcasting)
- Dot product
  - Between two vectors
  - Between two matrices
- Matrix transpose
- Matrix inversion

Matrix Inversion

- The inverse matrix of a square matrix is the inverse element of that matrix with respect to the dot product operation.

- Conditions for an invertible matrix:
  - It must be a square matrix
  - It must be a non-singular matrix, i.e., $\det(\boldsymbol{A}) \neq \mathbf{0}$.

- Important properties:
  - $(\boldsymbol{A}^{-1})^{-1} = \boldsymbol{A}$
  - $\boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}$
  - $(\boldsymbol{A}^{T})^{-1} = (\boldsymbol{A}^{-1})^{T}$

```
from numpy.linalg import inv
A = numpy.random.uniform(0,1,(3,3))
print(inv(A))
```

```
[[-0.92714039  0.11653039  1.41232812]
 [-0.40393881  1.46465429 -0.11835523]
 [ 2.1522805  -0.64040583 -0.23933963]]
```

Thanks!