NumPy

October 28, 2025

NumPy Library in Python

Dr.Labeed Al-Saad

What is NumPy?

NumPy (Numerical Python) is the fundamental package for scientific computing in Python. It provides:

- A powerful N-dimensional array object.
- Sophisticated broadcasting functions.
- Tools for integrating C/C++ and Fortran code.
- Useful linear algebra, Fourier transform, and random number capabilities.

1. Importing NumPy

[1]:
```
import numpy as np
```

2. Creating Arrays Basic Arrays

[7]:
```
import numpy as np
# 1D array
arr1d = np.array([1, 2, 3, 4, 5])
print("1D array:", arr1d)
```

```
1D array: [1 2 3 4 5]
```

[8]:
```
#2D array
import numpy as np
arr2d = np.array([[1, 2, 3], [4, 5, 6]])
print("2D array: \n", arr2d)
```

```
2D array:
 [[1 2 3]
 [4 5 6]]
```

[11]:
```
#3D array
import numpy as np
arr3d = np.array([[[1,2], [3, 4]], [[5, 6], [7, 8]]])
print("3D array: ", arr3d.shape) # .shape to show the array dimentions
print("3D array: \n", arr3d)
```

```
3D array:  (2, 2, 2)
3D array:
 [[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
```

Special Arrays

[13]:
```
# Array with zeros
import numpy as np
zeros_arr = np.zeros((3, 4)) # 3, 4 represents the dimensions of the array␣
 ↪(shape)
print ("Zeros array: \n", zeros_arr)
```

```
Zeros array:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

[14]:
```
# Array with ones
import numpy as np
ones_arr = np.ones((2, 3))
print("Ones array: \n", ones_arr)
```

```
Ones array:
 [[1. 1. 1.]
 [1. 1. 1.]]
```

[15]:
```
# Identity matrix
import numpy as np
identity = np.eye(3)
print("Identity matrix:\n", identity)
```

```
Identity matrix:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

[16]:
```
# Array with a range
import numpy as np
range_arr = np.arange(0, 10, 2)  # start, stop, step
print("Range array:", range_arr)
```

```
Range array: [0 2 4 6 8]
```

[17]:
```
# Linear spaced array
import numpy as np
linspace_arr = np.linspace(0, 1, 5)  # start, stop, num_points
```

```
print("Linspace array:", linspace_arr)
```

```
Linspace array: [0.   0.25 0.5  0.75 1.  ]
```

[18]:
```
# Random array
import numpy as np
random_arr = np.random.random((2, 3)) # his calls the `random()` function from␣
 ↪NumPy's random module.
print("Random array:\n", random_arr)
```

```
Random array:
 [[0.74313008 0.40526029 0.96543796]
 [0.43370957 0.96162971 0.59925989]]
```

3. Array Properties

[19]:
```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])

print("Array:\n", arr)
print("Shape:", arr.shape)      # Dimensions
print("Size:", arr.size)        # Total elements
print("NDim:", arr.ndim)        # Number of dimensions
print("Dtype:", arr.dtype)      # Data type
print("Itemsize:", arr.itemsize) # Bytes per element
```

```
Array:
 [[1 2 3]
 [4 5 6]]
Shape: (2, 3)
Size: 6
NDim: 2
Dtype: int64
Itemsize: 8
```

4. Array Indexing and Slicing

[20]:
```
import numpy as np
arr = np.array([[1, 2, 3, 4],
                [5, 6, 7, 8],
                [9, 10, 11, 12]])

print("Original array:\n", arr)
```

```
Original array:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

[21]:
```
# Indexing
print("Element at [0, 1]:", arr[0, 1])
print("Element at [2, 3]:", arr[2, 3])
```

```
Element at [0, 1]: 2
Element at [2, 3]: 12
```

[22]:
```
# Slicing
print("First row:", arr[0, :])
print("First column:", arr[:, 0])
print("Subarray:\n", arr[1:3, 1:3]) #selects rows and columns with indices 1␣
 ↪and 2 (up to but not including 3)
```

```
First row: [1 2 3 4]
First column: [1 5 9]
Subarray:
 [[ 6  7]
 [10 11]]
```

[23]:
```
# Boolean indexing
bool_idx = arr > 5
print("Boolean mask:\n", bool_idx)
print("Elements > 5:", arr[arr > 5])
```

```
Boolean mask:
 [[False False False False]
 [False  True  True  True]
 [ True  True  True  True]]
Elements > 5: [ 6  7  8  9 10 11 12]
```

5. Array Operations

Mathematical Operations

[28]:
```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print("a + b:", a + b)    # Element-wise addition
print("a - b:", a - b)    # Element-wise subtraction
print("a * b:", a * b)    # Element-wise multiplication
print("a / b:", a / b)    # Element-wise division
print("a ** 2:", a ** 2)  # Element-wise power
print("np.sin(a):", np.sin(a))  # Trigonometric functions
```

```
a + b: [5 7 9]
a - b: [-3 -3 -3]
a * b: [ 4 10 18]
a / b: [0.25 0.4  0.5 ]
a ** 2: [1 4 9]
```

```
np.sin(a): [0.84147098 0.90929743 0.14112001]
```

Aggregate Operations

[30]:
```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])

print("Sum of all elements:", np.sum(arr))
print("Sum along columns:", np.sum(arr, axis=0))
print("Sum along rows:", np.sum(arr, axis=1))
print("Mean:", np.mean(arr))
print("Standard deviation:", np.std(arr))
print("Min:", np.min(arr))
print("Max:", np.max(arr))
```

```
Sum of all elements: 21
Sum along columns: [5 7 9]
Sum along rows: [ 6 15]
Mean: 3.5
Standard deviation: 1.707825127659933
Min: 1
Max: 6
```

6. Array Manipulation

[37]:
```
# Reshaping
import numpy as np
arr = np.arange(12) # "arange" function generates a sequence of integers␣
 ↪starting from 0 (default) up to, but not including, 12
print("Original:", arr)
reshaped = arr.reshape(4, 3)
print("Reshaped: \n", reshaped)

reshaped2 = arr.reshape(3, 4)
print("Reshaped2: \n", reshaped2)

reshaped3 = arr.reshape(-1)
print("Reshaped3: \n", reshaped3)

# Or

flattened = reshaped.flatten()
print("flattened array 'vector':", flattened)
```

```
Original: [ 0  1  2  3  4  5  6  7  8  9 10 11]
Reshaped:
 [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

```
Reshaped2:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
Reshaped3:
 [ 0  1  2  3  4  5  6  7  8  9 10 11]
flattened array 'vector': [ 0  1  2  3  4  5  6  7  8  9 10 11]
```

Note: The `-1` parameter in `reshape()` is a special placeholder that tells NumPy to automatically calculate the appropriate size for that dimension, based on the total number of elements and the other specified dimensions. The other way of converting an array to a vector is by using **flatten()** function:

Transposing

[38]:
```
# Transpose the reshaped array of the example above
transposed = reshaped.T
print("Original Reshaped array: \n", reshaped)
print("Transposed: \n", transposed)
```

```
Original Reshaped array:
 [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
Transposed:
 [[ 0  3  6  9]
 [ 1  4  7 10]
 [ 2  5  8 11]]
```

Stacking n NumPy, "stacking" refers to the process of combining multiple arrays along a new or existing axis. It's a way to join arrays together to create a larger array. There are several stacking operations in NumPy:

1. **General stacking** (`np.stack`): Stacks arrays along a new axis.
2. **Concatenation** (`np.concatenate`): A more general function that can stack along any existing axis.
3. **Vertical stacking** (`np.vstack` or `np.row_stack`): Stacks arrays row-wise (along axis 0).
4. **Horizontal stacking** (`np.hstack` or `np.column_stack`): Stacks arrays column-wise (along axis 1).
5. **Depth stacking** (`np.dstack`): Stacks arrays along the third axis (depth).

[49]:
```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([3, 4, 5])
c = np.array([5, 6, 7])

print("General stack: \n", np.stack((a, b, c))) # Output: 2D (3, 3) array
print("Concatination: \n", np.concatenate((a, b, c))) # output will be a␣
 ↪vector,
```

```
print("Vertical stack: \n", np.vstack((a, b, c))) # output will be 2D array,␣
 ↪work like np.stack
print("Horizontal stack: \n", np.hstack((a, b, c))) # output will be 2D array,␣
 ↪work like np.concatenate
print ("Depth stack: \n", np.dstack((a, b, c))) # output will be 3D array
```

```
General stack:
 [[1 2 3]
 [3 4 5]
 [5 6 7]]
Concatination:
 [1 2 3 3 4 5 5 6 7]
Vertical stack:
 [[1 2 3]
 [3 4 5]
 [5 6 7]]
Horizontal stack:
 [1 2 3 3 4 5 5 6 7]
Depth stack:
 [[[1 3 5]
  [2 4 6]
  [3 5 7]]]
```

7. Broadcasting NumPy can perform operations on arrays of different shapes:

[56]:
```
# Array + array +scalar
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print("Array + 2: \n", arr +2)

# Array + 1D array
row = np.array([10, 20, 30])
print("\n Array + row: \n", arr + row)

#Array + column
col = np.array([[10], [20]])
print("\n Array + col: \n", arr + col)

# Array -, *, / 1D array
print("\n Array - row: \n", arr - row)
print("\n Array - col: \n", arr - col)
print("\n Array * row: \n", arr * row)
print("\n Array * col: \n", arr * col)
print("\n Array / row: \n", arr / row)
print("\n Array / col: \n", arr / col)
```

```
Array + 2:
 [[3 4 5]
 [6 7 8]]
```

```
Array + row:
[[11 22 33]
[14 25 36]]

Array + col:
[[11 12 13]
[24 25 26]]

Array - row:
[[ -9 -18 -27]
[ -6 -15 -24]]

Array - col:
[[ -9  -8  -7]
[-16 -15 -14]]

Array * row:
[[ 10  40  90]
[ 40 100 180]]

Array * col:
[[ 10  20  30]
[ 80 100 120]]

Array / row:
[[0.1  0.1  0.1 ]
[0.4  0.25 0.2 ]]

Array / col:
[[0.1  0.2  0.3 ]
[0.2  0.25 0.3 ]]
```

8. Linear Algebra

[72]:
```
# Matrix multiplication

import numpy as np
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
C = np.array([[8, 9], [10, 11], [12, 13]])
D = np.array([[1, 2], [7, 8], [3, 4]])
print("Matrix 2*2 multiplication: \n", np.dot(A, B))
print("Matrix 3*2 multiplication: \n", np.dot(C, D.T)) # array "D" should be␣
 ↪transposed

# Determinant
print("\n Determinant of A: \n", np.linalg.det(A))
```

```
# Inverse
print("\n Inverse of A:\n", np.linalg.inv(A))

# Eigenvalues and eignvectors
# The `np.linalg.eig(A)` function returns a tuple containing two elements:
# 1. A 1D array of eigenvalues
# 2. A 2D array where each column is an eigenvector
eigenvalues, eigenvector = np.linalg.eig(A)
print("\n Eigenvalues:", eigenvalues)
print("\n Eigenvector:\n", eigenvector)
```

```
Matrix 2*2 multiplication:
 [[19 22]
 [43 50]]
Matrix 3*2 multiplication:
 [[ 26 128  60]
 [ 32 158  74]
 [ 38 188  88]]

 Determinant of A:
 -2.0000000000000004

 Inverse of A:
 [[-2.   1. ]
 [ 1.5 -0.5]]

 Eigenvalues: [-0.37228132  5.37228132]

 Eigenvector:
 [[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
```

9. Random Module

[78]:
```
# various random distributions
import numpy as np
print ("\n Random uniform:\n", np.random.uniform(0,1,5)) #The function returns␣
 ↪an array of 5 random floating-point
                                                        # numbers, each between␣
 ↪0 (included) and 1 (excluded),
                                                        # with all values in␣
 ↪that range having equal probability
                                                        # of being generated.
print ("\n Random normal:\n", np.random.normal(0,3,5))
print ("\n Random randint:\n", np.random.randint(0,10,5)) # The result will be␣
 ↪a NumPy array containing 5 random
```

```
                                                    # integers, each␣
 ↪between 0 and 9 inclusive.

# Random choice
choices = ['a', 'b', 'c', 'd']
print("\n Random choice:", np.random.choice(choices, 3)) # Choosing random 3␣
 ↪elements of choices

# Shuffling
arr = np.arange(10)
np.random.shuffle(arr)
print("\n Shuffled array:", arr) # re-arrange the elements"arr" randomly
```

```
Random uniform:
[0.33939918 0.98913336 0.41688163 0.72991983 0.52389888]

Random normal:
[-1.65988703 -0.63875863  1.8247653  -1.77001411 -1.45885415]

Random randint:
[0 4 6 3 7]

Random choice: ['b' 'b' 'c']

Shuffled array: [6 0 4 7 3 2 1 5 9 8]
```

Practical Example: Data Analysis

[80]:
```
# Simulating student grades
import numpy as np
np.random.seed(42)  # For reproducible results

# Generate random grades (0-100) for 50 students in 5 subjects
grades = np.random.randint(0, 101, (50, 5))
print("Grades shape:", grades.shape)
print("\n The grades array:\n", grades)

# Calculate statistics
print("Average grade per subject:", np.mean(grades, axis=0)) # `axis=0`␣
 ↪specifies that the mean should be
                                                    # calculated along␣
 ↪the first axis (down each column).
                                                    # This means it␣
 ↪will compute the average grade for
                                                    # each subject␣
 ↪across all 50 students
print("Highest grade per subject:", np.max(grades, axis=0))
```

```python
print("Lowest grade per subject:", np.min(grades, axis=0))
print("Standard deviation per subject:", np.std(grades, axis=0))

# Students with average > 80
average_grades = np.mean(grades, axis=1)
top_students = average_grades > 80
print("Number of top students:", np.sum(top_students))
```

```
Grades shape: (50, 5)

 The grades array:
 [[ 51  92  14  71  60]
 [ 20  82  86  74  74]
 [ 87  99  23   2  21]
 [ 52   1  87  29  37]
 [  1  63  59  20  32]
 [ 75  57  21  88  48]
 [ 90  58  41  91  59]
 [ 79  14  61  61  46]
 [ 61  50  54  63   2]
 [100  50   6  20  72]
 [ 38  17   3  88  59]
 [ 13   8  89  52   1]
 [ 83  91  59  70  43]
 [  7  46  34  77  80]
 [ 35  49   3   1   5]
 [ 53   3  53  92  62]
 [ 17  89  43  33  73]
 [ 61  99  13  94  47]
 [ 14  71  77  86  61]
 [ 39  84  79  81  52]
 [ 23  25  88  59  40]
 [ 28  14  44  64  88]
 [ 70   8  87   0   7]
 [ 87  62  10  80   7]
 [ 34  34  32   4  40]
 [ 27   6  72  71  11]
 [ 33  32  47  22  61]
 [ 87  36  98  43  85]
 [ 90  34  64  98 100]
 [ 46  77   2   0   4]
 [ 89  13  26   8  78]
 [ 14  89  41  76  50]
 [ 62  95  51  95   3]
 [ 93 100  22  14  42]
 [ 28  35  12  31  70]
 [ 58  85  27  65  41]
 [ 44  61  56   5  27]
```

```
 [ 27  43  83  29  61]
 [ 74  91  88  61  96]
 [  0  26  61  76   2]
 [ 69  71  26   8  61]
 [ 36  96  50  43  23]
 [ 78  58  31  95  87]
 [ 51  61  57  51  11]
 [ 38   1   2 100  55]
 [ 80  58   1   1  91]
 [ 53  86 100  95  96]
 [  0  18   1  52  43]
 [ 89  31  69  31  67]
 [ 54  74  55  16  37]]
Average grade per subject: [50.76 52.86 46.16 51.72 48.36]
Highest grade per subject: [100 100 100 100 100]
Lowest grade per subject: [0 1 1 0 1]
Standard deviation per subject: [28.32564915 31.0869812  29.48040705 32.80855986
28.47086932]
Number of top students: 2
```

File I/O

Note: np.array will save the file `my_array.npy` in the current working directory of your Python session. This is typically the directory from which you launched your Jupyter Notebook or Python script. If you want to save the file in a different location, you would need to provide a full path instead of just the filename, such as `np.save('/path/to/directory/my_array.npy', arr)`.

[81]:
```python
# Save and load arrays

import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])

# Save to file
np.save('my_array.npy', arr)

# Load from file
loaded_arr = np.load('my_array.npy')
print("Loaded array:\n", loaded_arr)

# Text files
np.savetxt('array.txt', arr)
loaded_txt = np.loadtxt('array.txt')
print("Loaded from text:\n", loaded_txt)
```

```
Loaded array:
 [[1 2 3]
 [4 5 6]]
Loaded from text:
 [[1. 2. 3.]
```

```
[4. 5. 6.]]
```

Key Advantages of NumPy

1. Performance: Much faster than Python lists for numerical operations.
2. Memory efficiency: Uses less memory than Python lists.
3. Convenience: Rich set of mathematical functions.
4. Interoperability: Works well with other scientific Python libraries

Best Practices

1. Always use vectorized operations instead of loops when possible.
2. Be mindful of array shapes and dimensions.
3. Use appropriate data types to save memory.
4. Leverage broadcasting for operations on different-shaped arrays

Comprehensive NumPy Commands & Methods Reference

ARRAY CREATION

Command	Description	Example
np.array() np.zeros()	Create array Array of zeros	np.array([1,2,3]) np.zeros((3,4))
np.ones() np.eye()	Array of ones Identity matrix	np.ones((2,3)) np.eye(3)
np.arange() np.linspace()	Evenly spaced values Specified	np.arange(0,10,2)
np.random.random()	num points Random values [0,1)	np.linspace(0,1,5)
np.random.normal()	Normal distribution	np.random.random((2,3))
		np.random.normal(0,1,100)

ARRAY PROPERTIES

Command	Description	Example
.shape .size	Dimensions tuple Total elements	arr.shape arr.size
.ndim .dtype	Number dimensions Data type	arr.ndim arr.dtype
.itemsize	Bytes per element Total bytes	arr.itemsize arr.nbytes
.nbytes .flags	Memory layout Transpose	arr.flags arr.T
.T		

INDEXING & SLICING

Command	Description	Example
[i,j] [s:e:step]	Basic indexing Basic slicing	arr[0,1] arr[1:4:2] arr[arr>5]
[bool_arr]	Boolean indexing Fancy indexing	arr[[0,2,4]] np.where(arr>5)
[idx_arr]	Condition indices Add dimension	arr[:,np.newaxis]
np.where()		
np.newaxis		

ARRAY MANIPULATION

Command	Description	Example
.reshape() .flatten()	Reshape array Flatten copy	arr.reshape(3,4) arr.flatten()
.ravel() np.transpose()	Flatten view Permute dimensions	arr.ravel() np.transpose(arr)
np.vstack() np.hstack()	Vertical stack Horizontal stack	np.vstack((a,b))
np.concatenate()	Join arrays Copy array	np.hstack((a,b))
.copy()		np.concatenate((a,b))
		arr.copy()

MATH OPERATIONS

Command	Description	Example
+,-,*,/ ** @	Element math Exponentiation	a + b arr ** 2 A @ B
np.dot() np.sqrt()	Matrix multiply Dot product	np.dot(a,b) np.sqrt(arr)
np.exp() np.log()	Square root Exponential Natural	np.exp(arr) np.log(arr)
np.log10()	log Base-10 log Sine function	np.log10(arr) np.sin(angles)
np.sin() np.abs()	Absolute value	np.abs(arr)

STATISTICS

Command	Description	Example
np.sum() np.mean()	Sum elements Arithmetic mean	np.sum(arr) np.mean(arr)
np.median() np.std()	Median value Standard deviation	np.median(arr) np.std(arr)
np.var() np.min()	Variance Minimum Maximum	np.var(arr) np.min(arr)
np.max()	Percentiles Correlation	np.max(arr)
np.percentile()	Covariance	np.percentile(arr,75)
np.corrcoef() np.cov()		np.corrcoef(x,y) np.cov(x,y)

LINEAR ALGEBRA

Command	Description	Example
np.linalg.solve()	Solve Ax=b Matrix inverse	np.linalg.solve(A,b)
np.linalg.inv()	Determinant	np.linalg.inv(A)
np.linalg.det() np.linalg.eig()	Eigenvalues/vectors SVD	np.linalg.det(A)
np.linalg.svd()	decomposition Cholesky	np.linalg.eig(A)
np.linalg.cholesky()	decomposition Matrix norm	np.linalg.svd(A)
np.linalg.norm()	Condition number	np.linalg.cholesky(A)
np.linalg.cond()		np.linalg.norm(A)
		np.linalg.cond(A)

SET OPERATIONS

Command	Description	Example

np.unique()	Unique elements Intersection	np.unique(arr)
np.intersect1d()	Union Set difference Boolean	np.intersect1d(a,b)
np.union1d()	operations All True Any True	np.union1d(a,b)
np.setdiff1d() &,|,~		np.setdiff1d(a,b)
np.all() np.any()		(arr>2)&(arr<8) np.all(arr>0)
		np.any(arr>0)

ADVANCED OPERATIONS

Command	Description	Example
np.vectorize()	Vectorize function Numerical	np.vectorize(func)
np.gradient() np.einsum()	gradient Einstein summation	np.gradient(arr)
np.fft.fft() np.fft.ifft()	FFT Inverse FFT Convolution	np.einsum('ij,jk',A,B)
np.convolve()	Evaluate polynomial Polynomial	np.fft.fft(signal) np.fft.ifft(freq)
np.polyval() np.roots()	roots	np.convolve(a,b)
		np.polyval([1,2],x)
		np.roots([1,-3,2])

MASKED ARRAYS & FILE I/O

Command	Description	Example
np.ma.masked_where()	Mask by condition No	np.ma.masked_where(arr==-999,arr)
.compressed() .filled()	masked values Fill masked	masked.compressed() masked.filled(0)
.mean() np.save()	values Mean ignore mask	masked.mean()
np.load() np.savetxt()	Save to .npy Load from .npy	np.save('data.npy',arr)
np.loadtxt()	Save to text Load from text	np.load('data.npy')
		np.savetxt('data.txt',arr)
		np.loadtxt('data.txt')

MEMORY & SPECIAL ARRAYS

Command	Description	Example
np.add.accumulate()	Cumulative sum Memory	np.add.accumulate(arr)
order='C'/'F' out parameter	layout In-place operations	np.array(data,order='C')
np.meshgrid() np.diag()	Coordinate grids Extract	np.multiply(a,b,out=a)
np.triu() np.tril() np.pad()	diagonal Upper triangle Lower	np.meshgrid(x,y) np.diag(A)
	triangle Pad array	np.triu(A) np.tril(A)
		np.pad(arr,1)

COMPLEX NUMBERS & RANDOM

Command	Description	Example

np.real() np.imag()	Real part Imaginary part Phase
np.angle() np.conj()	angle Conjugate Set random
np.random.seed()	seed Shuffle array Poisson
np.random.shuffle()	distribution
np.random.poisson()	