NUMPY クイックリファレンス

配列の作成、数学演算、線形代数など

配列の作成

リストから

```
import numpy as np
a = np.array([1, 2, 3])         # 1次元
b = np.array([[1, 2], [3, 4]])  # 2次元
```

組み込みコンストラクタ

```
np.zeros((2, 3))     # 2x3のゼロ配列
np.ones((3, 3))      # 3x3の1配列
np.eye(4)            # 4x4の単位行列
np.arange(0, 10, 2)  # [0, 2, 4, 6, 8]
np.linspace(0, 1, 5) # 5個の等間隔値
```

配列のプロパティ

a.shape 次元数のタプル：`(3, 4)`
a.ndim 次元数
a.size 要素の総数
a.dtype データ型：`float64`、`int32`など

インデックスとスライス

基本インデックス

```
a = np.array([[1, 2, 3], [4, 5, 6]])
a[0, 1]    # 2 (行0、列1)
a[1]       # [4, 5, 6] (行1)
a[:, 0]    # [1, 4] (全行、列0)
```

スライス

```
a[0, 1:]   # [2, 3] (行0、列1以降)
a[:, :2]   # 最初の2列
a[::2]     # 1行おき
```

ブールインデックス

```
a = np.array([10, 20, 30, 40])
a[a > 15]   # [20, 30, 40]
a[a % 20 == 0]  # [20, 40]
```

配列演算

要素ごとの演算

```
a = np.array([1, 2, 3])
a + 10    # [11, 12, 13]
a * 2     # [2, 4, 6]
a ** 2    # [1, 4, 9]
a + a     # [2, 4, 6]
```

比較

```
a = np.array([1, 2, 3, 4])
a > 2          # [False, False, True, True]
np.where(a > 2, a, 0)  # [0, 0, 3, 4]
```

集計

a.sum()	全要素の合計
a.mean()	算術平均
a.std()	標準偏差
a.min() / a.max()	最小値/最大値
a.argmin() / a.argmax()	最小値/最大値のインデックス
a.cumsum()	累積和

軸ごとの結果には`axis=0`（列）または`axis=1`（行）を追加

数学関数

一般的な関数

np.sqrt(a)	各要素の平方根
np.abs(a)	絶対値
np.exp(a)	各要素の e^x
np.log(a)	自然対数（ln）
np.log10(a)	常用対数（底 10）
np.sin(a) / np.cos(a)	三角関数（ラジアン）
np.round(a, 2)	小数点以下 2 桁に丸め
np.clip(a, lo, hi)	値を[lo, hi]にクランプ

線形代数

行列演算

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
A @ B             # 行列の積
np.dot(A, B)      # A @ Bと同じ
A.T               # 転置
```

分解と解法

```
np.linalg.inv(A)       # 逆行列
np.linalg.det(A)       # 行列式
np.linalg.eig(A)       # 固有値/固有ベクトル
np.linalg.solve(A, b)  # Ax = bを解く
```

乱数

乱数生成

```
rng = np.random.default_rng(42)  # シード固定
rng.random((2, 3))     # 一様分布[0, 1)
rng.integers(1, 10, 5) # [1, 10)の5個の整数
rng.normal(0, 1, 100)  # N(0,1)から100個
rng.choice([1, 2, 3], size=2)  # サンプリング
```

レガシー API

```
np.random.seed(42)
np.random.rand(3, 3)     # 一様分布3x3
np.random.randn(3, 3)    # 標準正規分布
np.random.shuffle(arr)   # インプレースシャッフル
```

形状変換

形状操作

```
a = np.arange(12)
a.reshape(3, 4)    # 3x4の行列
a.reshape(3, -1)   # 列数を推論
a.flatten()        # 1次元に戻す（コピー）
a.ravel()          # 1次元に戻す（ビュー）
```

スタックと分割

```
np.vstack([a, b])  # 垂直方向にスタック
np.hstack([a, b])  # 水平方向にスタック
np.concatenate([a, b], axis=0)
np.split(a, 3)     # 3つに分割
```

ブロードキャスト

ブロードキャストの仕組み

```
a = np.array([[1, 2, 3],
              [4, 5, 6]])  # 形状(2,3)
b = np.array([10, 20, 30]) # 形状(3,)
a + b  # bが(2,3)にブロードキャスト
```

ルール

ルール 1 ランクが一致するまで短い形状の先頭に 1 を追加
ルール 2 次元が等しいか一方が 1 の場合に一致
ルール 3 サイズ 1 の次元はもう一方に合わせて伸長

ファイル I/O

NumPy バイナリ

```
np.save("data.npy", arr)     # 単一配列
arr = np.load("data.npy")
np.savez("data.npz", a=x, b=y) # 複数配列
d = np.load("data.npz"); d["a"]
```

テキストファイル

```
np.savetxt("data.csv", arr, delimiter=",")
arr = np.loadtxt("data.csv", delimiter=",")
arr = np.genfromtxt("data.csv", delimiter=",",
                    skip_header=1)
```

よくあるパターン

[0, 1]に正規化

```
normalized = (a - a.min()) / (a.max() - a.min())
```

ユークリッド距離

```
dist = np.sqrt(np.sum((a - b) ** 2))
# または：np.linalg.norm(a - b)
```

ユニーク値とカウント

```
vals, counts = np.unique(a, return_counts=True)
dict(zip(vals, counts))
```

ソート

```
np.sort(a)             # ソート済みのコピー
idx = np.argsort(a)    # ソート順のインデックス
a[idx]                 # ソート順を適用
```