NumPyの配列操作

ダウンロード・データあります

佐藤 聖

● 科学技術計算と言えばNumPy

NumPy[1]は，Pythonで科学技術計算を行うためのライブラリです．量子コンピューティング，統計計算，信号処理，画像処理，グラフとネットワーク，天文学のプロセス，認知心理学，バイオ・インフォマティクス，ベイズ推定，数学的分析，シミュレーション・モデリング，多変量解析，地理的処理，インタラクティブ・コンピューティングなどに利用されます．

データ・サイエンス分野で抽出，変換，読み込み，探索的分析，モデル化と評価，ダッシュボード注1でのレポートなどのワークフローに活用されています．

数値演算がPython標準の関数やライブラリよりも高速に処理できるので，科学技術計算を行わない場合もメリットがあります．

特に配列の要素の並び替え，追加，削除などは，処理時間を短縮できるので，ラズベリー・パイのようにコンピュータ性能が低いデバイスにも向いています．

● 最新版のインストール方法

NumPyは，Raspberry Pi OSに標準インストールされています．

もし最新版をインストールする場合には，以下のコマンドを実行してインストールが可能です．

```
# sudo apt-get update↵
# sudo apt-get -y upgrade↵
# sudo␣apt-get␣install␣-y␣
                    python3-numpy↵
```

apt-getコマンドでインストールされるのは，必ずしもリリース済みNumPyの最新版ではなく，Raspberry Pi OSのリポジトリに登録されている最新版になります．

注1：各種データをグラフィカルに表示し，1つの画面にまとめたものがダッシュボードです．データ可視化アプリやウェブ・アプリなどでダッシュボードが使われます．PythonではNumPyでデータの収集・集計して，Dashライブラリでダッシュボード付きのウェブ・アプリを作る例があります．

9-1 NumPy配列を作るarray関数

● 標準ライブラリより処理が速い

Pythonには同種類の複数データを扱うために，C言語などの配列に似たデータ構造としてリストがあります．リストは利便性は高いのですが，高速な処理には不向きなため，NumPyでは異なるデータ構造を利用できるようになっています．

このデータ構造(配列)を使うと，NumPyライブラリの機能を使って科学技術計算を高速に処理できるようになります．

以下で標準のリストとNumPy配列の違いを説明します．

▶標準のリスト

Pythonに組み込まれているlist関数は，外部ライブラリをimportしなくても使え，1つのリストの要素として異なる型を格納することもできます．list関数で作るリストによって，多次元配列を表現することも可能ですが，狭義の配列注2とは異なる場合があります．

配列ライクな簡単な処理を行うのであればlist関数で作るリストで十分な場合が多いです．この場合，文字列，整数，小数，ブール値を1つのリストに格納できます．

▶NumPy配列

NumPy配列は数値を扱うことに最適化されている配列です．同じ型のオブジェクトしか格納できません．その代わりほとんどの場合，list関数で作ったリストよりも処理が速いです．

数値以外のブール型，ユニコード文字列，Pythonオブジェクト型も扱えます．

Pythonで標準のリストを作成したあとで，NumPyのarray関数を使うことでNumPy配列に変換できます．

これ以外にも，NumPy配列の作成には，arange関数，ones関数，zeros関数などが使えます．

● Pythonのオブジェクトを NumPy配列に変換するarray関数の書式

array関数の書式は次の通りです．

```
numpy.array(object, dtype=None, *,
copy=True, order='K', subok=False,
                    ndmin=0, like=None)
```

● array関数の引数

よく使うarray関数の引数を次に示します．

▶object

NumPy配列に変換するリスト，タプルなどのシーケンスを代入します．

注2：狭義の配列とは，コンピュータ・プログラムにおける配列という意味です．これは同一型のデータ(オブジェクト)をメモリ上に1列に間隔を空けずに並べた構造です．
しかし，list関数では異なる型のデータが混在して1列に並べられた構造も許されるため，狭義の配列と同一とは認められず，このようなデータ構造をリストと呼んでいます．

▶dtype

デフォルトではNoneです．通常は省略しますが，配列を指定するときにデータ型を設定できます．

▶subok

デフォルトのFalseでは，返される配列が強制的に基本クラスの配列になります．Trueの場合サブクラスはパス・スルーされます．

パス・スルーとは素通りのことで，入力したものを加工や処理せずに出力することです．

引数subokがTrueの場合，array関数に入力されたシーケンスはオブジェクトのサブクラスを引き継いで配列を生成します．

▶copy

めったに使用することはないと思いますが，引数copyはデフォルトでTrueです．この場合は，オブジェクトがコピーされます．True以外のブール値，またはオプションを指定できます．

▶order

引数orderは高速に処理したい場合に，最適化のため配列のメモリ・レイアウトを指定するときに使用できます．デフォルトの順序はKが指定されています．その他にも引数ndmin，引数likeがあります．

● 実行例

ここでは標準のリストをNumPy配列に変換します．

▶array関数で変換

図1の1行目でNumPyライブラリを読み込み，npの別名で利用できるようにしています．

2行目でリストをarray関数に渡してNumPy配列に変換し，変数xに格納します．このリストでは，全てのオブジェクトを小数に統一しています．もし，整数と小数が含まれていれば，配列ではオブジェクトは全て小数に変換されます．

3行目で変数xに格納されているオブジェクトを画面表示しています．

5行目のtype関数で変数xのオブジェクトの種類を取得し，print関数で画面出力しています．このようにリストをarray関数に渡すことでNumPy配列に変換できます．

変換例を次に示します．

```
import numpy as np
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
x
```

```
array([1., 2., 3., 4., 5.])
```

```
print(type(x))
```

```
<class 'numpy.ndarray'>
```

図1　array関数による変換1…標準リストの変換

▶ネストしたリストの場合

次にネスト構造のリストをNumPy配列に変換します．**図2**の1行目でNumPyライブラリを読み込み，npの別名を付けます．

2～3行目で変数list1，変数list2にリストを代入，4行目で変数list1と変数list2でネストされたリストを作成し，変数listsに代入します．

5行目で変数listsをarray関数でNumPy配列に変換し，変数array1に代入します．

6行目で変数array1の内容を画面出力しています．

このようにネスト構造のリストもNumPy配列に変換できます．

```
import numpy as np
list1 = [1, 2, 3, 4, 5]
list2 = [10, 20, 30, 40, 50]
lists = [list1, list2]
array1 = np.array(lists)
array1
```

```
array([[ 1,  2,  3,  4,  5],
       [10, 20, 30, 40, 50]])
```

図2　array関数による変換2…ネストしたリストの変換

9-2　オブジェクトを行列でカウントするshape関数

● shape関数の利用目的

NumPy配列が大きくなると配列のオブジェクトを数えて形状を把握することが困難です．shape関数を使うと配列のオブジェクトを行列でカウントできます．配列の形状を使ってループ処理するときに，あらかじめ行と列のサイズが分かっていれば条件式を作成してループ処理を最適化できます．

● shape関数の書式

shape関数の書式は次の通りです．

```
numpy.shape(a)
```

shape関数の引数aには，NumPy配列を格納した変数を設定します．戻り値は整数型のタプルで対応する配列の次元の長さ(要素数)を示します．

● 実行例

▶要素数を数える

最初に1次元のNumPy配列の形状を調べます．図3の1行目でNumPyライブラリを読み込み，npの別名で利用できるようにしています．

2行目でリストをarray関数に渡して，NumPy配列に変換し，変数aに格納します．

3行目でshape関数に変数aを代入するとタプルが返されます．結果は，(4,)となっており，1次元のNumPy配列に4つのオブジェクトが格納されていることが分かります．この例では，オブジェクト数が少ないですが，大量のオブジェクトを格納するNumPy配列の要素数を調べるのにも利用できます．

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

```
(4,)
```

図3　shape関数1…要素数を数える

▶ネストされたリスト

図4の2行目のようにネストされたリストをarray関数でNumPy配列に変換できます．

3行目のようにソースコードを記述して，変数aに格納されたNumPy配列の形状を調べることもできます．

返されたタプルは(2, 3)となっており，2次元のNumPy配列だと分かります．2つの配列に3つのオブジェクトがあると読み取れます．このように配列の外側からカウントされ，最後の数字がオブジェクト数になります．

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

```
(2, 3)
```

図4　shape関数2…ネストされたリスト

9-3　高速に数列を生成するarange関数

● 利用目的

arange関数で数列を作りNumPy配列を生成できます．これを使って等差数列，等比数列などを高速に処理できます．

for文やwhile文の条件式によく利用されるPythonのrange関数も数列を生成しますが，処理速度が遅いため，大きな数列を処理するのに向いていません．

arange関数でNumPy配列を生成するとNumPyで処理できるようになるので，Pythonプログラムで配列を使った処理を高速化できます．

● arange関数の書式

arange関数の書式は次の通りです．

```
numpy.arange([start, ]stop, [step, ]dtype=None, *, like=None)
```

[]で囲まれた引数は省略可能です．

● 引数

▶start

デフォルトの開始値は0で，省略できます．数列の開始値を指定することができます．

▶stop

整数を指定します．省略できません．

▶step

隣接する2つの値の差です．デフォルトは1です．負の値を設定することもできます．stepが位置引数として指定されている場合は，startも指定する必

要があります.

▶dtype

出力配列のタイプの指定に利用します. デフォルトでは, 他の入力引数からデータ型を推測します.

● 実行例

▶等差数列

等差数列を作成してみます. 図5の1行目でNumPyライブラリを読み込み, npの別名で利用できるようにしています.

2行目でarange関数の引数stopに10を代入してNumPy配列に変換し, 変数xに格納します. 引数start, 引数step, 引数dtypeは省略しました. 引数startはデフォルトの0, 引数stepは1のままです.

3行目のprint関数でNumPy配列を画面出力しています.

```
import numpy as np
x = np.arange(10)
print(x)
```

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

図5　arange関数1…等差数列

▶配列同士の計算

NumPy配列を生成して, 配列同士を使って計算します. 図6の1行目でNumPyライブラリを読み込み, npの別名を付けます.

2行目でarange関数の引数startに1, 引数stopに10を代入し, 生成されるNumPy配列を3つのオブジェクトごとに配列にし, 3つの配列を作ります. 作成した2次元配列は変数xに代入します. 3行目で変数xのオブジェクトを変数yに代入します.

4行目と8行目で変数xと変数yに格納されているオブジェクトを画面出力しています. この2つの変数に格納されているNumPy配列を使って計算してみます.

12行目では変数xと変数yの配列を使って同じ配列の要素番号にあるオブジェクト同士を足し算しています.

16行目では変数xと変数yの配列を使って引き算, 20行目では掛け算を行っています. このように同じ要素番号の値同士で計算できるので, 例えば画像のフィルタ処理, 機械学習の畳み込み処理や重み付けなどにも応用できます.

```
import numpy as np
x = np.arange(1, 10).reshape(3, 3)
y = x
print(x)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

8行目 →

```
print(y)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

12行目 →

```
print(x + y)
```

```
[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
```

16行目 →

```
print(x - y)
```

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
```

20行目 →

```
print(x * y)
```

```
[[ 1  4  9]
 [16 25 36]
 [49 64 81]]
```

図6　arange関数2…配列同士の計算

9-4　NumPy配列のデータ構造の操作

● 利用目的

NumPy配列は，自由に配列を操作できます．任意の行・列の削除，並び替え，結合，ソート，置換，抽出などです．

非常にたくさんの機能があります．ここでは，よく使う機能に絞って紹介します．

● 配列操作の書式

NumPy配列の操作の書式は次の通りです．

▶1次元配列を多次元配列に変換

```
numpy.reshape(a, newshape,
order='C')
```

▶配列の転置（行と列の入れ替え）

```
numpy.transpose(a, axes=None)
```

▶配列の結合

```
numpy.concatenate((a1, a2, …),
axis=0, out=None, dtype=None,
casting="same_kind")
```

▶配列要素の削除

```
numpy.delete(arr, obj, axis=None)
```

▶reshape関数

引数aにNumPy配列を，引数newshapeに整数または整数のタプルを指定します．引数orderはインデックス順序を使って要素を読み取り，このインデックス順序を使って要素を再生成した配列を配置します．

▶transpose関数

引数aに配列を指定します．引数axesを省略するとデフォルトで転置行列を返します．

▶concatenate関数

引数(a1, a2, …)にはNumPy配列を指定します．引数axisは配列を結合する軸の指定です．デフォルトは0で配列同士（行方向）を連結させ，1のときには列方向に連結します．1次元配列もしくは多次元配列で使えます．

▶delete関数

引数arrにNumPy配列，引数objに削除する行番号や列番号を整数，スライス，リスト（配列）で指定します．引数axisが0で行を，1で列を削除対象にします．

● 実行例

▶1次元配列を多次元配列に変換

図7の1行目でNumPyライブラリを読み込み，npの別名で利用できるようにしています．

2行目でarange関数の引数stopに10を代入してNumPy配列に変換し，変数aに格納します．

3行目のprint関数でNumPy配列を画面に出力しています．

5行目のreshape関数にNumPy配列が格納されている変数aと，配列の形状を指定する整数型のタプルを渡しています．ここでは2つの配列にそれぞれ5つのオブジェクトを格納しています．このように一度生成した配列を2次元配列に変換できます．

```
import numpy as np
a = np.arange(10)
print(a)
```

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

5行目

```
print(np.reshape(a, (2, 5)))
```

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

図7　NumPy配列の操作1…1次元配列を多次元配列に変換

▶配列の転置（行と列の入れ替え）

図8の1～2行目は上記のソースコードと同じです．

3行目で変数aに格納されたNumPy配列をreshape関数に2と5を指定して2次元配列に変換しています．

この配列を5行目でtranspose関数に渡します．配列の軸を並び替えてリスト型オブジェクトが返され，変数aに代入します．

6行目のprint関数で変数aのオブジェクトを画面出力します．

```
import numpy as np
a = np.arange(10)
a.reshape(2, 5)
```

```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

5行目

```
a = np.transpose(a)
print(a)
```

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

図8　NumPy配列の操作2…配列の転置（行と列の入れ替え）

▶配列の結合

図9の1行目でNumPyライブラリを読み込み，npの別名を付けています．

2～3行目でネストされたNumPy配列を作成し，変数aと変数bに代入します．

4行目のconcatenate関数で変数aと変数bを列方向に連結し，print関数で画面出力します．

6行目のconcatenate関数で変数aと変数bを行方向に連結し，print関数で画面出力します．

このように行・列の方向を指定して配列を連結できます．

```
import numpy as np
a = np.array([[1, 2, 3],[4, 5, 6]])
b = np.array([[7, 8, 9],[10, 11, 12]])
print(np.concatenate([a, b], axis=1))
```

4行目

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

6行目

```
print(np.concatenate([a, b], axis=0))
```

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

図9　NumPy配列の操作3…配列の結合

▶配列の要素を削除

図10の1行目でNumPyライブラリを読み込み，npの別名を付けています．

2行目のarange関数で配列を生成し，reshape関数で再成形して変数aに代入しました．

3行目のprint関数で変数aのオブジェクトを画面出力します．

7行目のdelete関数で変数aの配列から行方向に数えて1番目の要素を削除し，変数aに代入します．

8行目のprint関数で変数aを画面表示します．

```
import numpy as np
a = np.arange(12).reshape(3, 4)
print(a)
```

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

7行目

```
a = np.delete(a, 1, 0)
print(a)
```

```
[[ 0  1  2  3]
 [ 8  9 10 11]]
```

図10　NumPy配列の操作4…配列の要素を削除

◆参考文献◆

(1) NumPy公式サイト，
https://numpy.org/

さとう・せい