線形代数で扱う実体

線形代数は，ベクトル空間とベクトル空間上の線形写像を体系的理論として完成させたものです．そこで取扱う主な数学的実体として，次のようなものがあります．

- ベクトル空間
- ベクトル空間の点
- ベクトル空間上の線形写像
- ベクトル空間の部分空間

ベクトル空間が何であるかの説明はしばらく保留します．とりあえず，ここでは私達が普通に思っている空間と考えてください．

これらの実体を実現するオブジェクトについて2段階で説明します．まず，数学的表現を説明し，そのあとPythonによる表現を説明します．

線形代数のオブジェクト

線形代数では，次の3種類の数学的対象を扱います．

- スカラー（Scalar）
- ベクトル（Vector）
- 行列（Matrix）

これらについて簡単に説明します．

スカラー

スカラーとは，実数または複素数のどちらかを言います．一般的に数学の理論体系はできる限り広い範囲で構築すます．線形代数についても例にもれず複素数を前提として展開します．しかし，現実的な問題への適合性を考えて実数に制限することもよくあります．また，その幾何学的な構造を考えた場合，スカラーを実数で扱ったほうが明確になります．この講座では，状況によってスカラーとして実数と複素数を使い分けます．

集合として実数は$\mathbb{R}$と表し，複素数は$\mathbb{C}$と表します．そしてスカラーを実数か複素数か特定しない場合は$\mathbb{K}$と記載することとします．

Pythonにおけるスカラーの扱い

Pythonにおいて実数は，float型ないしはint型の数値として扱われます． Pythonでは変数のデータ型を明示的に指定しませんが，変数に数値を代入することによって適切なデータ型になります．

オブジェクトのデータ型は，type()関数の引数に対象となるオブジェクトを渡すことによって確認できます． type()関数の構文は次の通りです．

```
type(*object*)</font>
```

幾つかのデータについて，そのデータ型を確認します．

```
type(123)
type(12.3)
type(1+1j)
```

```
In [1]: type(123)
Out[1]: int

In [2]: type(12.3)
Out[2]: float

In [3]: type(1+1j)
Out[3]: complex
```

複素数の表現で分かるようにPythonでは虚数単位を i ではなく j を用いている点に気を付けてください．また，$1+j$ と記載すると j を変数と認識してしまうので，j の前には必ず数値を明記することにも注意してください．

数値のデータ型について意識する必要は殆どありませんが，非常に小さな数値による割り算でオーバーフローの危険を回避するためにデータ型を指定することもあります．

ベクトル

ベクトルとは，複数個のスカラーの順序のついた組み合わせによって表現されるものです．ベクトルを構成するそれぞれのスカラーを成分と言います．ベクトルを扱う場合，その前提としてベクトル成分の個数を固定します．ベクトル成分の個数がn個の場合はn次元と呼びます．線形代数においてはベクトル成分の個数は有限個の範囲で論理展開します．

ベクトルの表現

n次元のベクトルは，n個のスカラーの組を括弧で囲って表しますが，その並べ方として縦に並べる列ベクトルと横に並べる行ベクトルがあります．列ベクトルは縦ベクトルとも言います．また，行ベクトルは横ベクトルとも言います．

列ベクトルの表記法

n次元の列ベクトルは次にように表記します．ここで x_i 各はスカラーです．

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

行ベクトルの表記法

n次元の行ベクトルは次にように表記します．

$\mathbf{x} = (x_1\, x_2\, \cdots\, x_n)$

ただし数値を横に並べたときに各成分の区別がつくように間にカンマを入れて$\mathbf{x} = (x_1, x_2, \cdots, x_n)$と表記することもあります．

線形代数では，ベクトルと言えば列ベクトルのことを言います．従いまして，単にベクトルと記載した場合は列ベクトルを示します．また，文中に列ベクトルを展開して記載する場合に省スペースのために$(x_1\, x_2\, \cdots\, x_n)^{\mathsf{T}}$という記載を行うことがあります．これは転置行列の記載方法を利用して行ベクトルを横長に記載する表現です．

Pythonにおけるベクトルの扱い

n個の数字の順序がある組はリスト配列によって表現することができます．しかし，ベクトルをリスト配列で保持すると線形代数の演算に対応することが出来ません．Pythonで線形代数の演算を行うには高機能配列演算を有するNumPyライブラリーを利用します．

NumPyライブラリーはPythonの科学技術計算には欠かせないものとなっています． Pythonの導入においてAnacondaディストリビューションを採用した場合，NumPyは同梱されています．したがって，単にNumPyをimport文で搬入するだけて利用可能となります．

NumPyを搬入するimport文は次にようになります．

```
import numpy as np
```

ここで別名にnpを指定していますが，Pythonの利用者の間では慣用的に使われている別名ですので，私たちもこの別名を使うことにします．

In [4]:
```
import numpy as np
```

さて，Pythonにおいてベクトルはndarrayオブジェクトとして保有します．例として3次元のベクトル$(1,2,3)$を持つndarrayオブジェクトの変数myArrayを生成します．

In [5]:
```
myArray = np.array([1,2,3])
myArray
```

Out[5]:
```
array([1, 2, 3])
```

このように，myArrayという変数ができましたが，この変数のオブジェクト型をtype()関数で確認します．

In [6]:
```
type(myArray)
```

Out[6]:
```
numpy.ndarray
```

myArrayは，numpy.ndarrayオブジェクトであることが確認できました．線形代数の多くの書籍ではベクトルの例として2次元か3次元しか扱っていません．しかし，数学とITを同時に学習することによって，もっと大きな次元についても取り扱うことが可能になります．その例として10次元のベクトルを作成してみましょう．

In [7]:
```
np.array(range(1,11))
```

Out[7]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

ベクトルの特徴や演算などについては，後々学習していきます． Pythonによる実践としてはNumPyのndarrayオブジェクトで対応すると理解してください．

行列

行列とは，縦と横に複数のスカラーが並んだものです．縦を行といい，横を列といいます．行の個数をmとし，列の個数をnとした行列をm行n列の行列，あるいは(m,n)型行列と言います． $\mathbf{A}$をm行n列の行列とし，そのi行j列目の成分の値をa_{ij}としたとき，この行列を次にように表現します．

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

ちなみに，m行1列の行列は列ベクトルです．1行n列の行列は行ベクトルです．

行列を記載すると多くの場所を占めてしまいますので，行列を簡略に表記する省略記法があります．

$$\mathbf{A} = (a_{ij})_{mn} = (a_{ij})$$

m行n列であることがはっきりしていれば，(a_{ij})と記載します．

行列の例

例として3行4列の行列について見て見ましょう．

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

この行列で2行3列目の値は$a_{23} = 7$となっています．ちなみに，この行列を行列要素をインデックスによる計算値で表すと，$a_{ij} = 4i + j - 4$となります．

Pythonにおける行列の扱い

Pythonでは行列をベクトルと同様にNumPyのndarrayオブジェクトとして保有します．それでは，上記の例に示した行列をmyMatrixというオブジェクトとして表現してみましょう．

ndarrayオブジェクトにおいて行列は二重配列で表現します．

```
In [8]: myMatrix = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
        myMatrix
```

```
Out[8]: array([[ 1,  2,  3,  4],
               [ 5,  6,  7,  8],
               [ 9, 10, 11, 12]])
```

2行3列目の要素にアクセスしてみましょう．ただし，Pythonのインデックスは0から始まるので，行と列のインデックスの値をそれぞれ-1して指定する必要があります．

```
In [9]: myMatrix[1,2]
```

```
Out[9]: 7
```

行列やベクトルについて，幾つかの演算が定義されています．この後，それらの演算について学習します．