数理生物学演習

第２回　Pythonの基本的な使い方と数理生物学演習で使う数学の復習

野下 浩司（Noshita, Koji）

noshita@morphometrics.jp

https://koji.noshita.net

理学研究院 数理生物学研究室

第２回：Pythonの基本的な使い方と数理生物学演習で使う数学の復習

本日の目標

- 数学的なツールの復習
- プログラミングの基礎
- 可視化

皆さんへのお願い

- わからないところがあればすかさずググろう！
  調べる習慣をつける.

- 質問や回答をSlackへ投稿しよう.
  情報が共有できる．一人の質問が皆の質問に！

- 困ったら（Slack上）で助けを呼ぼう（特に，TAがサポートしてくれる）．困っている人がいれば助けてあげよう.

- 演習中の休憩は自由．疲れ果てる前に休もう.

操作解説動画

https://www.youtube.com/playlist?list=PLNP5gU_8uAjejxZOd7TljPo6iQ4sWloD8

Colabやその他ツールで操作方法の説明が必要そうなものは補足資料を用意する.
補足説明が欲しい場合は課題の感想欄で要望を．可能な範囲で対応します.

数学的なツールの復習

この演習で必要になる数学的なツール

- 解析
  - 微分，積分
  - テイラー展開
  - 微分方程式の解法（変数分離ができればOK）
- 線形代数
  - ベクトルや行列の演算
  - 行列式
  - ヤコビ行列
  - 固有値・固有ベクトル
- その他いろいろ

解法などを暗記する必要はないが，（ここで挙げたもの以外でも）わからないものが出てきたら調べて，理解し，利用できるようになろう．

変数分離で微分方程式を解く

ある微分方程式

$$\frac{dx}{dt} = f(x, t)$$

を解きたい.

この式が

$$\frac{dx}{dt} = g(t)h(x)$$

と表せるとき，これを変数分離形と呼ぶ.

解法

$$\frac{dx}{dt} = g(t)h(x)$$

$$\frac{1}{h(x)}dx = g(t)dt \quad (h(x) \neq 0 \text{とする})$$

$$\int \frac{1}{h(x)}dx = \int g(t)dt + C \quad (C\text{は積分定数})$$

これを両辺積分して，xについて整理してやれば良い．Cは初期値や境界条件から決まる.

他にも解析的に解くことができる微分方程式はあるが，すべての微分方程式が解析的に解くことができるわけではない．そのような場合に計算機を使ったシミュレーションの出番となる.

変数分離：指数増殖モデルの例

ある集団のサイズ（個体数）を x とし，
その増加速度（dx/dt）が集団サイズ $x(t)$ に比例する場合，
ダイナミクスは以下の式で表すことができる．

$$\frac{dx}{dt} = ax$$

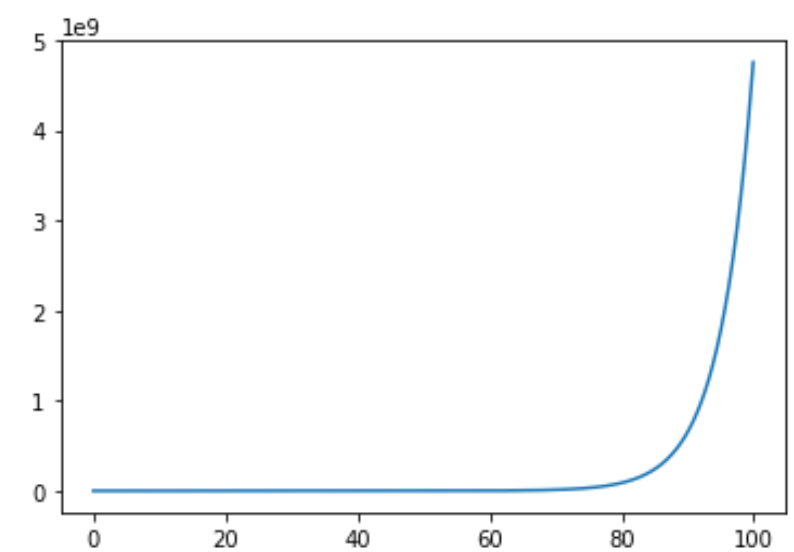
初期条件

$$x(0) = x_0$$

a：単位時間あたり一個体あたりの増加率（マルサス係数）

$$x(t) = x_0 e^{at}$$

解いてみよう



変数分離：指数増殖モデルの例

$$\frac{dx}{dt} = ax$$

$$\frac{1}{x}dx = adt$$

$$\int \frac{1}{x}dx = \int adt$$

よって，

$$\log x = at + C_0$$

$$x(t) = e^{at+C_0}$$

初期値$x(0) = x_0$とし，xについて整理してやれば，

$$x(t) = x_0 e^{at}$$

ヤコビ行列 Jaccobian matrix

微分係数（ある関数の接線の傾き）の高次元版

n個の変数をもつm列のベクトル値関数

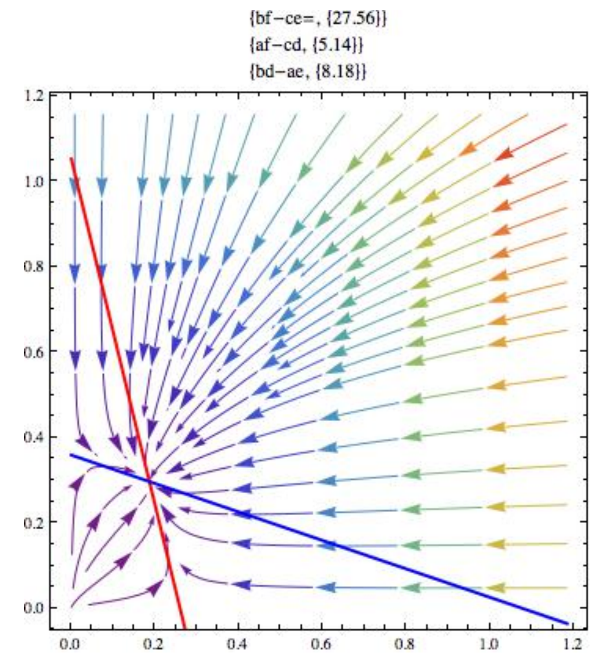
$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_1\left(x_1, x_2, \cdots, x_n\right) \\ f_2\left(x_1, x_2, \cdots, x_n\right) \\ \vdots \\ f_m\left(x_1, x_2, \cdots, x_n\right) \end{pmatrix}$$

について,

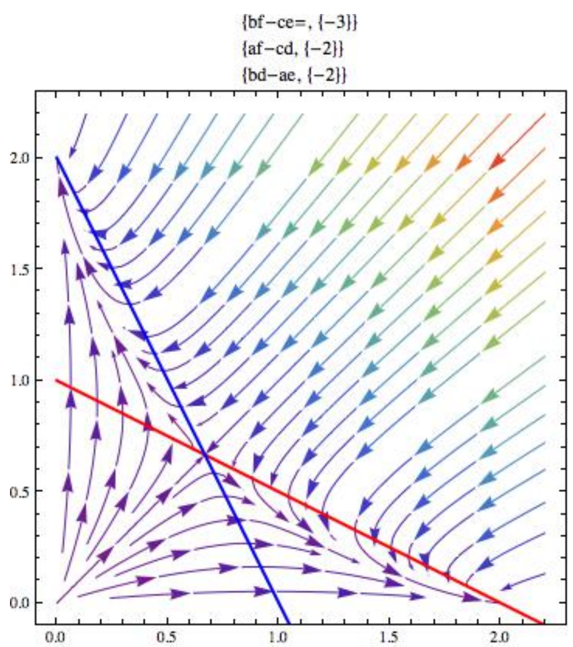
$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_2}{\partial x_n}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(\mathbf{x}) & \frac{\partial f_m}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_m}{\partial x_n}(\mathbf{x}) \end{pmatrix}$$

となる $m \times n$ 行列をヤコビ行列という.

例）



安定共存



双安定

例えば，ある連立微分方程式のヤコビ行列を考え，平衡点周りでの固有値・固有ベクトルを計算すれば，その局所安定性を調べることができる（第５回でロトカ-ボルテラモデルについてやります）．

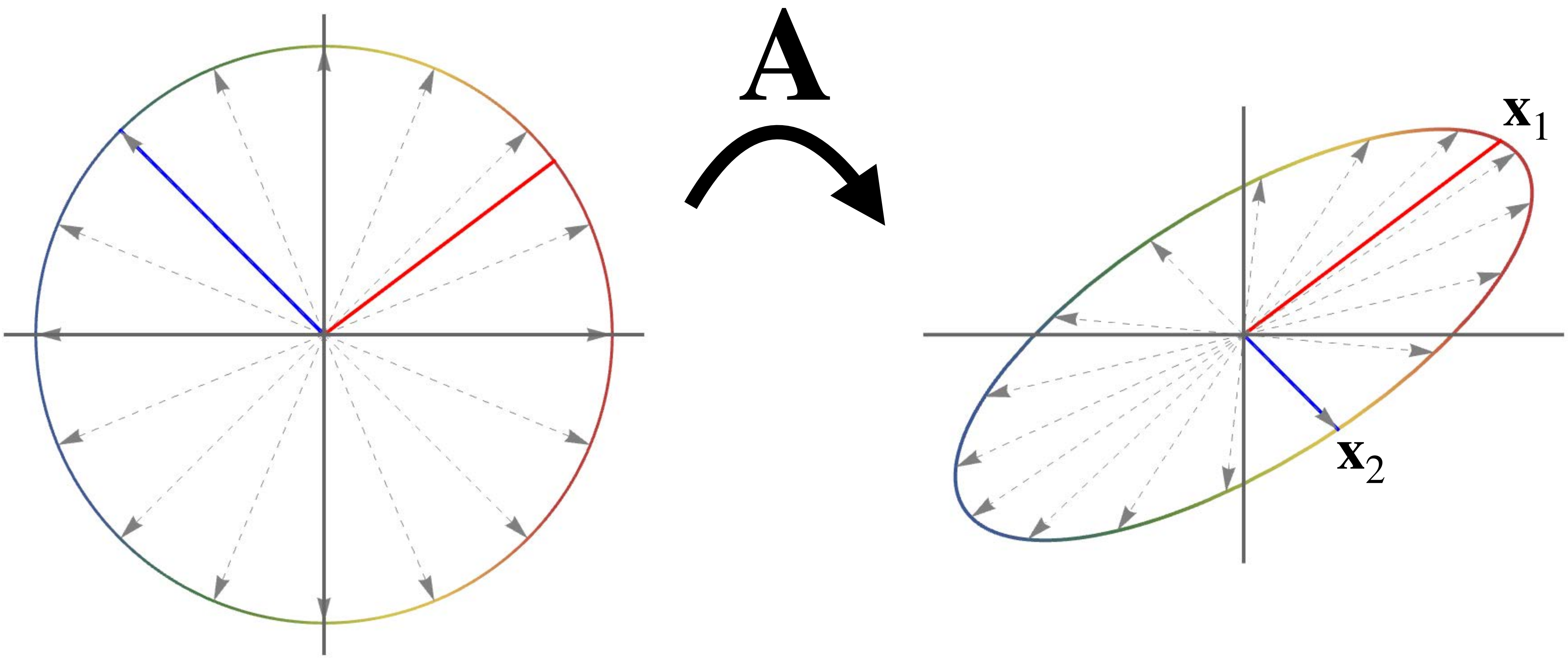
固有値・固有ベクトル

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} a_{11} a_{12} \cdots & a_{1n} \\ a_{21} a_{22} \cdots & a_{2n} \\ \vdots \quad \vdots \quad \ddots & \vdots \\ a_{n1} a_{n2} \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

n行ベクトル$\mathbf{x}$からn行ベクトル$\mathbf{y}$への線形変換（回転，拡大縮小，剪断変形，ミラーリング，の合成）を与える(n, n)型の正方行列$\mathbf{A}$を考える．

$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$となるような$\mathbf{x}$を$\mathbf{A}$の固有ベクトル，$\lambda$を$\mathbf{A}$の固有値という．



固有値・固有ベクトルを求める

$$\mathbf{A} = \begin{pmatrix} a & 0 \\ 0 & -d \end{pmatrix}$$

この行列の固有値・固有ベクトルを求めてみよう.

固有値の求め方

$$\mathbf{Ax} = \lambda \mathbf{x}$$

$$\left(\mathbf{A} - \lambda \mathbf{I}\right) \mathbf{x} = \mathbf{0}$$

よって,

$$\left| \mathbf{A} - \lambda \mathbf{I} \right| = 0$$

を満たす，λを求める.

固有ベクトルの求め方

$\lambda = a$について,

$$\left(\mathbf{A} - a\mathbf{I}\right) \mathbf{x} = \mathbf{0}$$

$$\begin{pmatrix} 0 & 0 \\ 0 & -d-a \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

よって，$x_1 = t, x_2 = 0$ （ただし，tは任意の定数）

もう一方についても同じように求める.

変数と型

復習 第１回参照

Colab

https://colab.research.google.com/

```
# 01-01. Hello, World!ノートブック
print("Hello, World!")
```

出力

```
Hello, World!
```

- コメントアウト（#）
  Pythonでは#から文末までが（実行時に）無視される
- print(オブジェクト)
  オブジェクトを出力

コードの実行：▶をクリック

本演習では主にColab上でノートブックを利用して進めていきます

Pythonの基本的なルール

- Pythonのプログラムは**論理行(logical line)**に分割され，解釈・実行される．論理行は一行以上の**物理行（physical line）**からなる．
- 複合文などのコードブロックは**インデント（indent，字下げ）**により表す．

上から順に実行される

```
# コード構造の例
a = 1
b = 1 + 2
c = 1 + 2 + \
  + 3

if a < b:
  print("a: ", a)
  print("b: ", b)
  print("c: ", c)
```

２つの物理行からなる１つの論理行

複合文ヘッダ

コードブロック

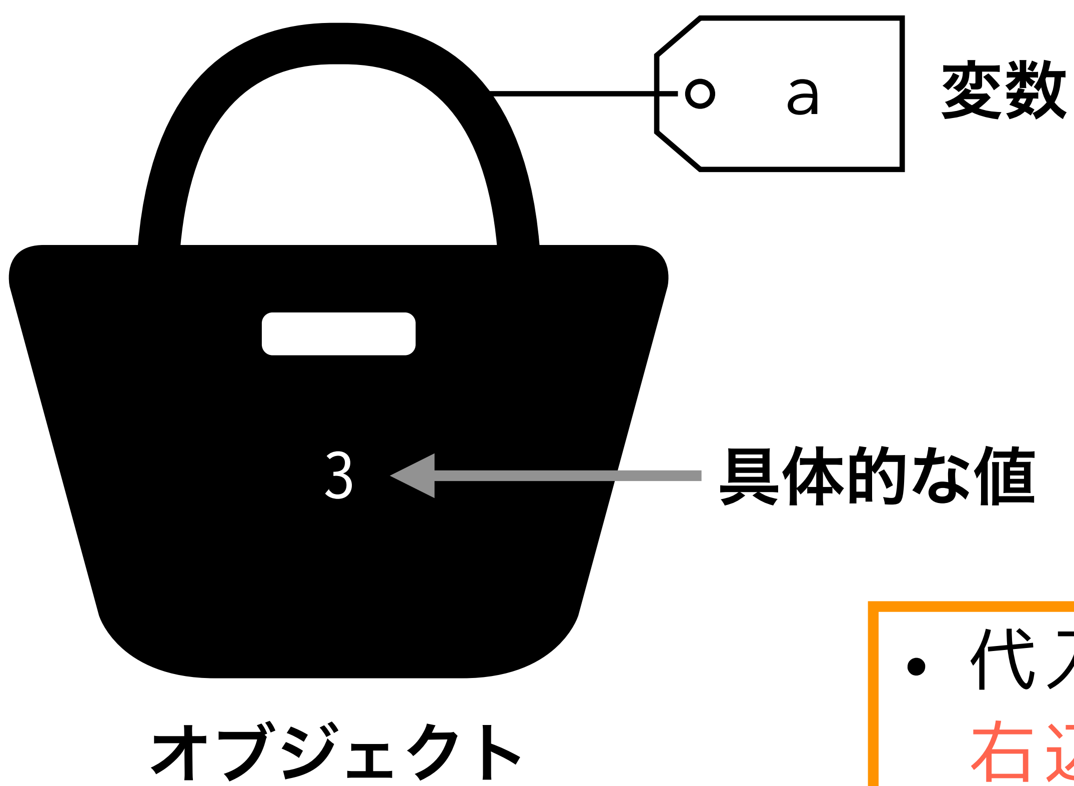
同じコードブロックは同じインデント（同じ個数の空白 or タブ）をもつ必要あり

コードの内容の詳細は省く

オブジェクト：「数値」や「文字」などの“容器”

Pythonではすべてのものはオブジェクトとして表現される

- オブジェクト：「数値」や「文字」の“容器”
- 変数：オブジェクトにつけられた”ラベル”
- リテラル：コード中に直接書かれた数値や文字列




```
# オブジェクトと代入
a = 3
print(a)
```

- 代入 左辺 = 右辺
  右辺のオブジェクト（もし右辺がリテラルならそれを格納したオブジェクト）に左辺の変数でラベル付けする
  数学の「=」とは異なる

オブジェクトの型 type

なかに入れる「数値」や「文字列」などの種類毎に「型」がある

→ 型に応じて，可能な処理が決まる or 想定される処理が異なる

この演習では当面は

- bool型 True（真），False（偽）
- int型　整数
- float型　実数
- complex型　複素数（虚部はjをつけて表す）
- str型　文字列データ
- list型　リスト

と理解すれば良い.

型は代入をしたタイミングで決まる

```python
# 01-01. 変数と型
a = 3
b = 6.2
c = True
print(type(a))
print(type(b))
print(type(c))

a = 3.3
b = False
c = 3 + 2j
print(type(a))
print(type(b))
print(type(c))
```

- type(オブジェクト)
  オブジェクトの型を返す

aに3が代入されたのでint型

aに3.3が代入されたのでfloat型

```
# 出力
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'float'>
<class 'bool'>
<class 'complex'>
```

Pythonの四則演算とその周辺

・加算　+
・減算　−
・乗算　*
・除算　/
・剰余　%

・指数　a**b
aのb乗

```
# 01-02. 四則演算など
a = 2
b = 5
d = 6.0
e = 7.5

# 加算・減算
c = a + b
f = d - e
print(c)
print(f)

# 乗算
c = a * b
f = d * e
print(c)
print(f)
```

```
# 除算
c = a / b
f = d / e
print(c)
print(f)

# 剰余
c = b % a
print(c)

# 指数
f = d**e
g = e ** (0.5)
print(f)
print(g)
```

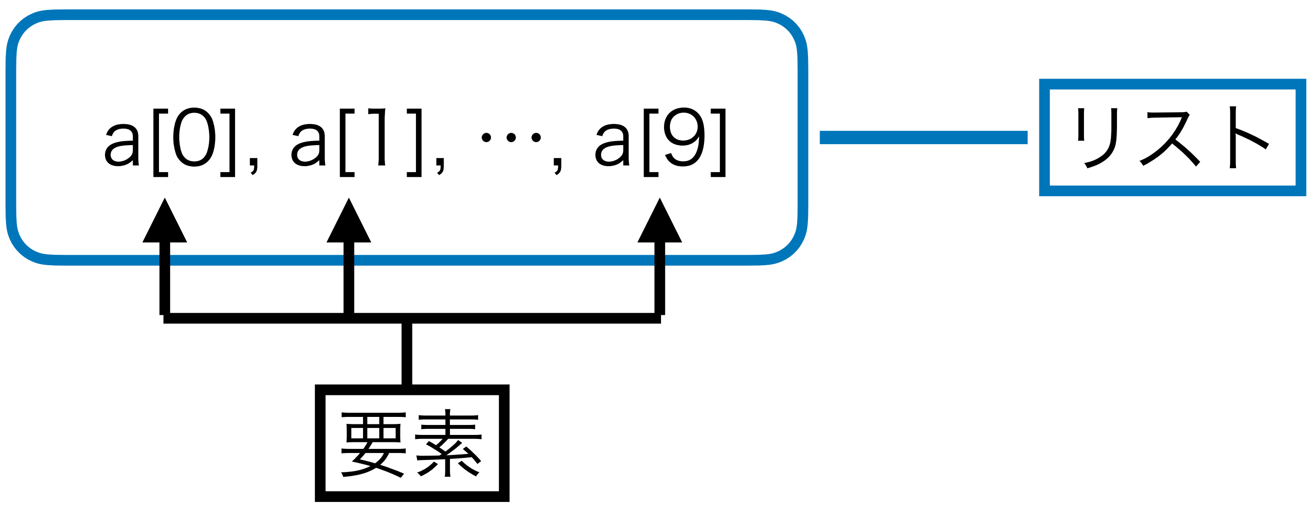
リストとループ

リスト

複数の要素の集まり

リスト = [要素１，要素２，…，要素n]

たくさんの変数を
個別に用意するのは面倒！




```
# 01-03. リストの作成と表示
a = [1, 2, 3, 4, 5]
print(a)

# 要素へのアクセス
print(a[1])  # 2が表示される

# 要素への代入
a[1] = 12  # 二番目の要素に12を代入
print(a)
```

各要素へは添字によってアクセスする

特に注意！！
添字は0から始まり，(サイズ-1)で終わる
a = [1, 2, 3, 4, 5] として作成したならば,
a[0]〜a[4]までの要素が存在する

反復処理　ループ（1）

同じ処理を何度も繰り返したい時に，その数だけコードを書くのは面倒！

forループを覚えよう！

・forループ

```
for イテレータ in イテラブル:
  文１
  文2
   ⋮
  文n
```

1. イテレータが終端に到達していれば終了
2. イテレータが要素を一つイテラブルから取り出し返す
3. 文１〜文nまでを評価．１へ戻る．

```
# 01-04. forループ
for i in [0, 1, 2]:
    print(i)
```

出力
0
1
2

特に注意！！
インデントされた文がfor文のブロックとみなされる

- イテレータ（iterator）：反復処理（イテレーション iteration）毎にイテラブル（リストなど）から要素を一つずつ取り出して返すもの
- イテラブル（iterable object）：イテレータを使って要素を一つずつ返すことができるオブジェクト．リスト，文字列など．

反復処理　ループ（2）

```
# 01-05a.
# forループ range 1
for i in range(50, 100):
    print(i)
```

出力
50
51
52
⋮
99

```
# 01-05b.
# forループ range 2
for i in range(10, 100, 3):
    print(i)
```

出力
10
13
16
⋮
97

```
# 01-05c.
# forループ range 3
for i in range(100):
    print(i)
```

出力
0
1
2
⋮
99

- 連番 range
  - range(開始, 終了)：開始から終了-1までの連番を表す.
  - range(開始, 終了, ステップ)：開始から終了-1までのステップおきの連番を表す
  - range(終了)：0から終了-1までの連番を表す．range(0, 終了)を意味する.

注意
rangeもイテラブルだが，リストとは異なる（ジェネレータ）．リストに変換したい場合は
list(range(100))
のようにする必要がある.

反復処理　ループ（３）

forループはネスト（入れ子構造）にできる

```
# 01-06. ネスト
for i in range(100):
    for j in range(100):
        print(i, j)
```

```
出力.
0 0
0 1
0 2
…
99 99
```

特に注意！！
ネストする場合も，インデントをによりブロック構造を記述する

```
# 01-07. ３重ネスト
for i in range(10):
    for j in range(10):
        for k in range(10):
            print(i, j, k)
```

```
出力.
0 0 0
0 0 1
0 0 2
…
9 9 9
```

何重にもネスト
することが可能

関数，モジュール・パッケージ

関数（1）

- Pythonおける関数とは，ある一連の処理を行うコードをまとめたもの
- これまで使ってきた，print()やtype()は関数
- 使う前に定義し，使うときに呼び出す必要がある．

関数定義

```
def 関数名():
    処理1
    処理2
    ...
    処理n
```

関数定義

関数呼び出し

```python
# 02-01. シンプルな関数
def simple_func():
    print("関数を呼び出しました．")


simple_func()
```

「関数を呼び出しました．」と表示させる関数

```
# 出力
関数を呼び出しました．
```

関数定義

関数呼び出し

```python
# 02-02. シンプルな関数 その2
def simple_func_2():
    print("1．関数を")
    print("2．呼び出し")
    print("3．ました．")


simple_func_2()
```

```
# 出力
1．関数を
2．呼び出し
3．ました．
```

あまり気にしなくても良い補足
printやtype関数は予め用意されている「組み込み関数」．はじめから使える．

- 組み込み関数 | 公式ドキュメント
  https://docs.python.org/ja/3/library/functions.html

何度も利用する処理を関数にまとめることで再利用性を高める

関数（2）：引数と戻り値

- **引数**により，入力に応じた処理をおこなうことができる
  例．print()が表示する文字列が変わる
- 処理した結果を，**戻り値**として返すことができる
  例．a = abs(-2) # aに2が代入される

関数定義

```
def 関数名(パラメータ):
    処理1
    処理2
    …
    処理n
    return 戻り値
```

- パラメータ（仮引数）：関数内でのみ利用される変数．関数に渡す値やリストなどを入力としてもつ．
- return文：関数を終了して，戻り値を返す

入力した値とその絶対値を表示させる関数

```
# 02-03. 引数をもつ関数
def my_abs_print(x):
    y = abs(x)
    print("入力", x)
    print("絶対値", y)


my_abs_print(-9)
```

```
# 出力
入力 -9
絶対値 9
```

２変数の足し算

```
# 02-04. 引数と戻り値をもつ関数
def add(a, b):
    c = a + b
    return c


x = add(2, 4)
print(x)
```

```
# 出力
6
```

モジュール・パッケージ（1）

Pythonコードをまとめたファイルやその集合

- モジュール：コードをまとめたファイル
- パッケージ：モジュールを階層的にまとめたもの

この演習ではこれらの区別はあまりしない．モジュール，パッケージ，ライブラリなど異なる名前で呼称するが，「必要なときに呼び出せる便利な機能をまとめたもの」ぐらいのニュアンスで理解しておけばOK

使い方

- import モジュール（もしくはパッケージ）
  モジュール（もしくはパッケージ）を読み込む

```python
# 02-05. mathモジュールの読み込み
import math

a = math.log(2)
print(a)
```

```python
# 02-06. osパッケージの読み込み
import os

filepath = os.path.join("parent", "child", "file.txt")
print(filepath)
```

便利な機能をまとめたものを再利用することで１から作る必要がなくなる！

mathモジュール

基本的な数学関係の関数

https://docs.python.org/ja/3/library/math.html

よく使いそうな関数の例

- log：自然対数
- sqrt：平方根
- sin,cos,tan, …：三角関数関係

数学関係の定数

- pi:円周率
- e:自然対数の底

```
# 02-07. mathモジュール
import math

print("円周率：", math.pi)
print("自然対数の底", math.e)

print("log(2)：", math.log(2))
print("√3：", math.sqrt(3))

print("sin(π/2)：", math.sin(math.pi / 2))
print("cos(π)：", math.cos(math.pi))
print("tan(π/4)", math.tan(math.pi / 4))
```

print関数の補足

- print(obj1, obj2, …)
  obj1, obj2, …を（デフォルトだと空白で）区切って表示

その他の標準ライブラリ（デフォルトで使えるモジュールやパッケージ）もあるので興味のある人は使ってみよう.

- **Python 標準ライブラリ | 公式ドキュメント** https://docs.python.org/ja/3/library/index.html

さらに，Colabには標準ライブラリ以外にもデータサイエンス向けのパッケージが多数インストール済み（特に追加インストールの必要なく呼び出せる）.

モジュール・パッケージ（2）

その他の読み込み方

- from パッケージ import モジュール
  パッケージ内のモジュールを読み込む
- import モジュール（もしくはパッケージ） as 省略名
  パッケージを省略名として読み込む
- from パッケージ import モジュール as 省略名
  パッケージ内のモジュールを省略名として読み込む

```
# 02-08.
# matplotlibパッケージのpyplotモジュールをpltとして読み込む
import matplotlib.pyplot as plt
```

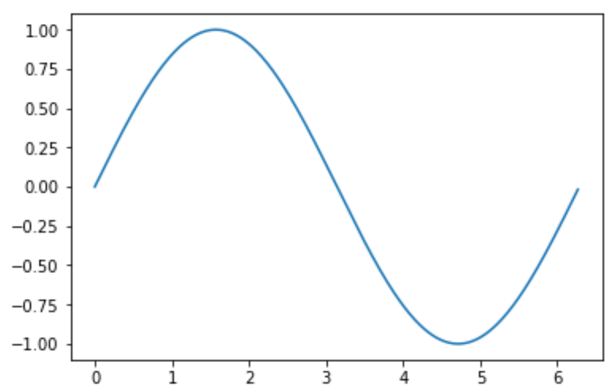
有名ライブラリの省略名はだいたい慣例があるので，それに従う（例. matplotlib.pyplot→plt）．また，自作のモジュールやパッケージを作る場合には，そうした有名ライブラリの名前や省略名との重複を避けるのが無難.

Matplotlib

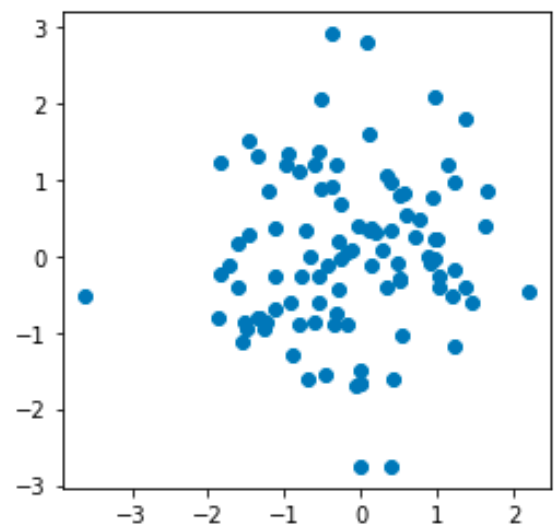
データ可視化・作図ライブラリ

https://matplotlib.org/

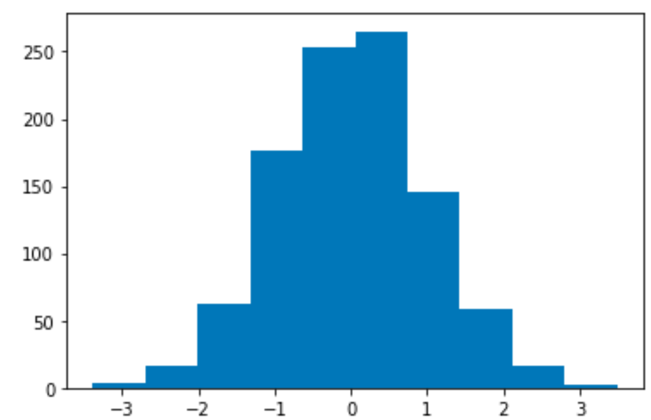
基本的なプロット



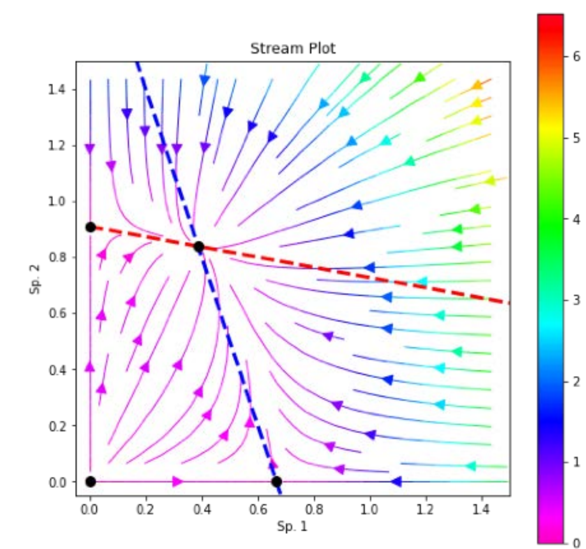
散布図



ヒストグラム



ベクトル場



ここでは例をいくつか示すだけで，個別の関数の詳細な使い方は説明しない．例に挙げた例以外にも様々なプロットが可能．公式のサンプル集を眺めてみると，使いたいプロット方法が見つかるかも．

- Gallery | 公式ドキュメント
  https://matplotlib.org/gallery/index.html

プロット

結果を図として可視化する

```
# 02-09. sin関数のプロット
import matplotlib.pyplot as plt
import math

x_start = 0
x_end = 2 * math.pi
step = math.pi / 36

i_start = 0
i_end = int((x_end - x_start) / step)

x_list = []
y_list = []
for i in range(i_start, i_end + 1):
    x = x_start + step * i
    y = math.sin(x)
    x_list.append(x)
    y_list.append(y)

plt.plot(x_list, y_list)
```

パッケージの読み込み

どこまで計算するか？（xの最大値）

xの軸方向の刻み幅 **刻み幅を変えてプロットしてみよう！**

forループの開始

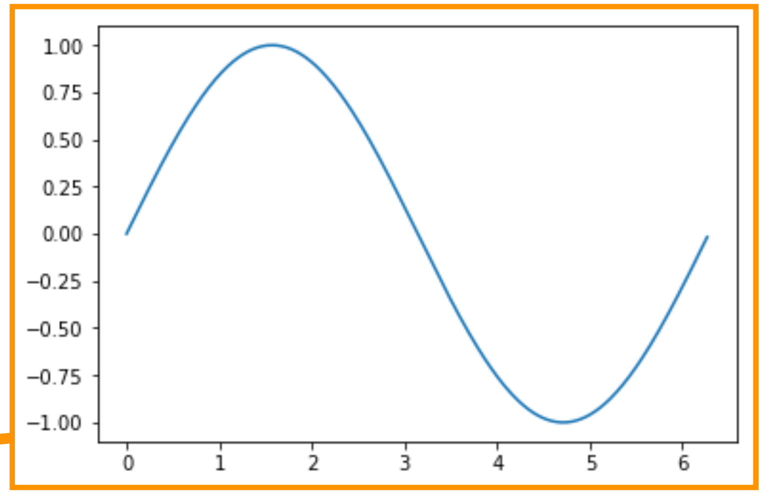
forループの終了

イテレータがi_startからi_endまで1刻みで増加するループ

x座標，y座標の値を格納するリスト

同じ高さのインデントによりforループのブロックを表現

出力



matplotlib.pyplot
- plot(横軸値リスト，縦軸値リスト)
  (横軸値，縦軸値)で与えられる座標値をプロットする

- リスト.append(要素)
  リストの末尾に要素を付け加える
- int(数値)
  数値を切り捨てて整数にする

本日の課題 ノーマル

1. $\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$の固有値・固有ベクトルを導出せよ.

2. ~~$f(\theta) = \cos(\theta)$を$0 \leq \theta \leq 4\pi$の範囲でプロットせよ.~~

3. その他質問，感想，要望をどうぞ.

課題をノートブック（.ipynbファイル）にまとめて，Moodleにて提出すること

ファイル名は[回数，01~15]_[難易度，ノーマル nかハード h].ipynb．例．02_n.ipynb

次回予告

第３回：離散ロジスティック成長

5月8日

復習推奨

- 離散指数増殖モデル
- 離散ロジスティックモデル
- 平衡点の導出
- 平衡点の局所安定性解析