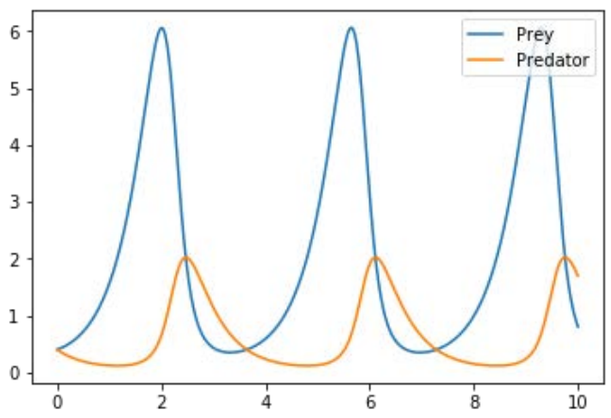
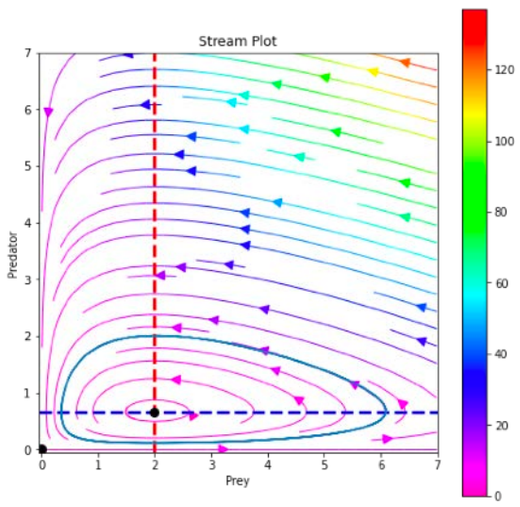
数理生物学演習

第５回　個体群動態の数理モデル（３）：
ロトカ-ボルテラ モデル





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

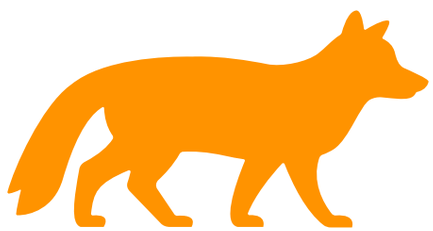
第５回：個体群動態の数理モデル（３）：ロトカ-ボルテラ モデル

本日の目標

- ロトカ-ボルテラ モデルの解析
- 固有値による力学系の局所安定性解析
- ニュートン法による平衡点の計算

ロトカ-ボルテラ モデル（被食-捕食系）

被食者



捕食者

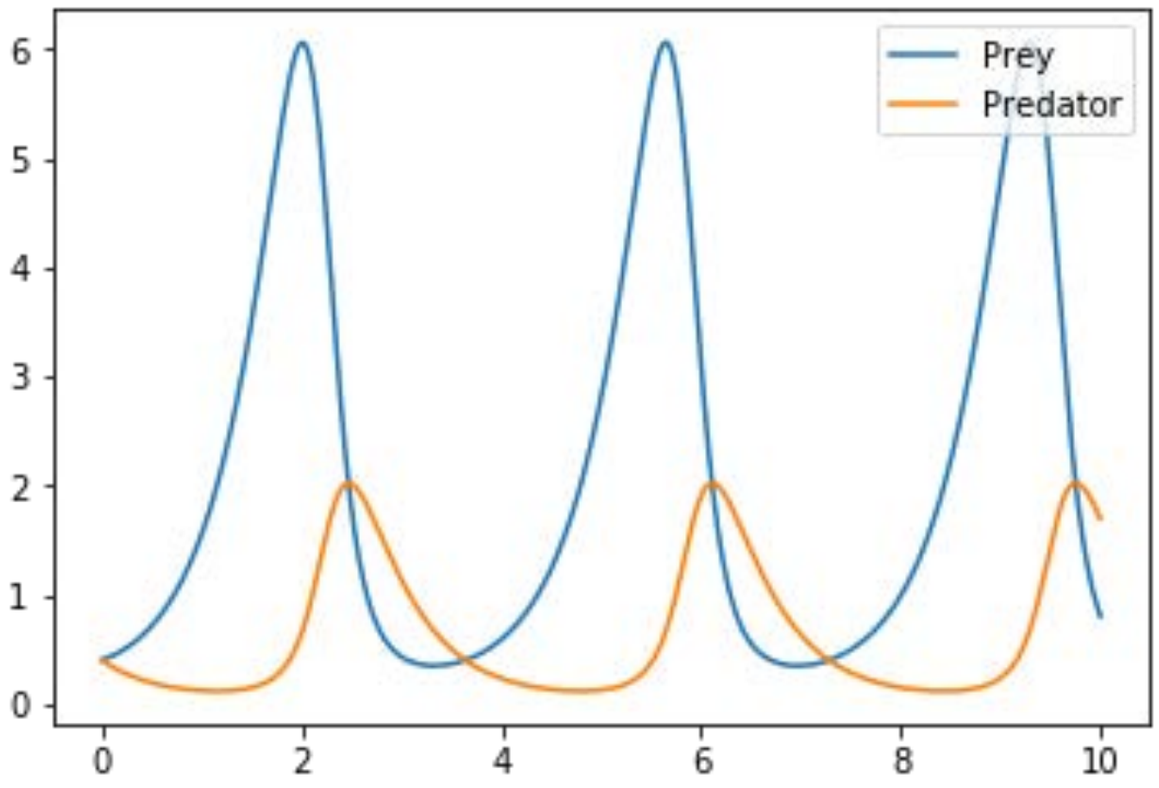
$$\begin{cases} \dfrac{dx}{dt} = ax - bxy \\ \dfrac{dy}{dt} = cxy - dy \end{cases}$$

指数的な増殖

被食者が捕食者に出会うと
一定の割合で捕食される

捕食量に応じて増殖

一定の死亡率で減少



$a, b, c, d > 0$ とする

- 捕食者がいなければ被食者は指数増殖
- 捕食者は被食者がいなければ
  死亡率一定で減っていく

振動するパタンが観察できる

ロトカ-ボルテラ モデル（２種系）

被食-捕食系

$$\begin{cases} \text{被食者} & \dfrac{dx}{dt} = ax - bxy \\ \text{捕食者} & \dfrac{dy}{dt} = cxy - dy \end{cases}$$

- 捕食者がいなければ被食者は指数増殖
- 捕食者は被食者がいなければ死亡率一定で減っていく

$a, b, c, d > 0$ とする

競争系

$$\begin{cases} \dfrac{dx}{dt} = ax - bx^2 - cxy \\ \dfrac{dy}{dt} = dy - exy - fy^2 \end{cases}$$

- 競争種がいるほど個体群成長率は減少する
- 競争種がいなければロジスティック成長

$a, b, c, d, e, f > 0$ とする

（相利）共生系

$$\begin{cases} \dfrac{dx}{dt} = ax - bx^2 + cxy \\ \dfrac{dy}{dt} = dy + exy - fy^2 \end{cases}$$

- 共生種がいるほど個体群成長率は増加する
- 共生種がいなければロジスティック成長

$a, b, c, d, e, f > 0$ とする

固有値による力学系の局所安定性解析

目的

個体群動態 $\dfrac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x})$ ただし, $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}$

の平衡点まわりの局所安定性を調べる

xに関する十分小さな”ずれ”**n**を考えると

$$\frac{d(\mathbf{x}+\mathbf{n})}{dt} = \mathbf{f}(\mathbf{x}+\mathbf{n}) \quad \cdots(1)$$

ただし, $\mathbf{n} = \begin{pmatrix} n_1 \\ n_2 \\ \vdots \\ n_m \end{pmatrix}$

右辺をテイラー展開すると

$$\mathbf{f}(\mathbf{x}+\mathbf{n}) = \mathbf{f}(\mathbf{x}) + \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\mathbf{n} + \textbf{2次以上の項}$$

平衡点 $\mathbf{x}^*$について考えると $\left.\dfrac{d\mathbf{x}}{dt}\right|_{\mathbf{x}=\mathbf{x}^*} = f(\mathbf{x}^*) = 0$

式（1）は

$$\frac{d\mathbf{n}}{dt} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}^*} \mathbf{n} + \textbf{２次以上の項}$$

nは十分小さいので

２次以上の項を無視すると

$$\frac{d\mathbf{n}}{dt} = \mathbf{M}\mathbf{n}$$

ただし,

$$\mathbf{M} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}^*} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}^*) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}^*) & \cdots & \frac{\partial f_1}{\partial x_m}(\mathbf{x}^*) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}^*) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}^*) & \cdots & \frac{\partial f_2}{\partial x_m}(\mathbf{x}^*) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(\mathbf{x}^*) & \frac{\partial f_m}{\partial x_2}(\mathbf{x}^*) & \cdots & \frac{\partial f_m}{\partial x_m}(\mathbf{x}^*) \end{pmatrix}$$

このm次の正方行列**M**の固有値を求めることで，平衡点の局所安定性を調べることができる.

- すべての固有値が負の実部をもつならば安定平衡点
- １つでも正の実部をもつと不安定平衡点

固有値による力学系の局所安定性解析の流れ

1. 平衡点を求める
2. 対象となる力学系を線形化する
3. 平衡点まわりでの固有値（と固有ベクトル）を求める
4. 固有値の実部の正負，虚部の有無を調べる

被食-捕食系

$$\begin{cases} \dfrac{dx}{dt} = ax - bxy & \text{被食者} \\ \dfrac{dy}{dt} = cxy - dy & \text{捕食者} \end{cases} \qquad a, b, c, d > 0 \text{ とする}$$

被食-捕食系について平衡点の局所安定性を調べてみよう

固有値による力学系の局所安定性解析の流れ

例.

1. 平衡点を求める

$$f_1(x,y) = ax - bxy = 0$$
$$f_2(x,y) = cxy - dy = 0$$

$$\Rightarrow \quad (x^*, y^*) = (0,0)\,, \left(\frac{d}{c}, \frac{a}{b}\right)$$

2. 対象となる力学系を線形化する

$$\mathbf{J}(x,y) = \begin{pmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{pmatrix} = \begin{pmatrix} a - by & -bx \\ cy & cx - d \end{pmatrix}$$

3. 平衡点まわりでの固有値（と固有ベクトル）を求める

$$|\mathbf{J}(0,0) - \lambda \mathbf{I}| = \begin{vmatrix} a - \lambda & 0 \\ 0 & -d - \lambda \end{vmatrix}$$

4. 固有値の実部の正負，虚部の有無を調べる

$$\lambda_1 = a > 0,\ \lambda_2 = -d < 0 \quad \Rightarrow \quad (x^*, y^*) = (0,0) \quad \text{のとき}$$

常に不安定

より詳しくいえば，鞍点（あんてん）saddle

型変換，関数を鍛える

型変換（1）

暗黙の型変換：いくつかのケースでは自動的に型の変換がおこなわれる

```
# 01-01. 暗黙の型変換
# 数値同士の演算, print関数
a = 5
b = 2
c = 3.3
d = 6.5 + 2j

print("type(a): ", a, type(a))
print("type(b): ",b, type(b))
print("type(c):", c, type(c))
print("type(d): ", d, type(d))

e = a + c
f = a + d
g = a / b
h = a * c

print("type(e): ",e, type(e))
print("type(f):", f, type(f))
print("type(g): ", g, type(g))
print("type(h): ", h, type(h))
```

int型 + float型 → float型

int型 + complex型 → complex型

int型 / int型 → float型

int型 * float型 → float型

!注 Pythonでは数値と文字列の演算では，暗黙の型変換が行われない．
例．1 + “1” はエラーになる．
後述の**明示的型変換**が必要．

- print(引数, …)
  では，引数をstr型に変換して表示している

割り切れる場合のint型/int型でもfloat型に変換される．興味のある人は試してみよう．

型変換（2）

明示的型変換：Pythonでは基本的には型変換したい場合は明示的に指定してやる必要がある

ある演算子が同じ型や特定の型にのみ対応している場合
ある関数の引数が特定の型にのみ対応している場合 などに利用

```
# 01-02. 明示的型変換

## str
a = str(1)
b = str(True)
c = str(5.2)

print("str(1): ", a, type(a))
print('str(True): ',b, type(b))
print('str(5.2): ', c, type(c))

# 文字列の結合への利用
print("str" + str(21))

## int
a = int("1")
b = int("010")
c = int(False)
d = int(10.0)

print('int("1"): ', a, type(a))
print('int("010"): ',b, type(b))
print("int(True): ", c, type(c))
print("int(10.0): ", d, type(d))

## float
a = float("-6.31")
b = float("5e-006")
c = float(1)
d = float(False)

print('float("-6.31"): ',a, type(a))
print('float("5e-006")', b, type(b))
print("float(1): ", c, type(c))
print("float(False): ", d, type(d))
```

int, bool, floatなど
→str

str, bool, floatなど
→int

str, int, boolなど
→float

関数を鍛える（１）：ローカル変数とグローバル変数

```
# 01-03. グローバル変数


s = "Hello, World!"

def print_hello_global():
  print(s)

print_hello_global()
```

```
# 出力
Hello, World!
```

```
# 01-04. グローバル変数とローカル変数


s = "Hello, World!"

def print_hello_local():
  s = "こんにちは世界！"
  print(s)

print_hello_local()
print(s)
```

```
# 出力
こんにちは世界！
Hello, World!
```

何が起こっている？

関数を鍛える（2）：変数のスコープ

- **ローカル変数は関数ブロック内でだけ利用可能**
  → 関数の処理が終了（関数の終端まで実行，return文に到達，など）すると消滅する
- **同じ名前の変数が存在する場合，ローカル変数が優先的に参照される**

ローカル変数sが参照される

```
# 01-04. グローバル変数とローカル変数

s = "Hello, World!"

def print_hello_local():
  s = "こんにちは世界！"
  print(s)

print_hello_local()
print(s)
```

```
# 出力
こんにちは世界！
Hello, World!
```

```
# 01-05. グローバル変数は関数内で書き換えられない

s = "Hello, World!"

def print_hello_local():
  print(s)
  s = "こんにちは世界！"
  print(s)

print_hello_local()
```

エラーになる

関数内からグローバル変数を参照する方法もあるが，あまり推奨しないのでここでは説明しない．
興味ある人はglobal宣言などを調べてみて．

プログラムが複雑になると，どこでどんな処理をおこなっているか解読が難しくなってくる（スパゲッティコード化）．ローカル変数を使うことで影響の及ぶ範囲を制御でき，可読性が上がる．

関数を鍛える（3）：パラメータのデフォルト値

```
def 関数名(パラメータ1，パラメータ2=デフォルト値):
    処理1
    処理2
    …
    処理n
    return 戻り値
```

- 関数定義の際に，特定のパラメータに対してデフォルト値を設定することができる．例ではパラメータ２にデフォルト値が設定されている．

！注意：デフォルト値があるパラメータを普通（デフォルト値がない）パラメータよりも先に書くとエラーになる．

```
# 01-06. パラメータのデフォルト値

def func1(a, b = "str1"):
    print(a,b)

func1(1,2)
func1(a=1)
func1(1,b=2)
func1(b=1) # エラー
```

```
# 出力
1 2
1 str1
1 2
```

うまく組み合わせて使いやすい関数を作ってみよう

関数を鍛える（４）：複数の戻り値

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

- return文でカンマを使って区切ることで複数の戻り値を返すことができる.
- 戻り値はタプル（変更できないリストみたいなもの）で与えられる.
- 変数もカンマで区切ることで代入できる

```
# 01-07. 複数の戻り値

def func_multi(a):
    return a, a**2, a**3


x, y, z = func_multi(2)
print(x,y,z)
```

入力（パラメータa）に対して，a，a^2，a^3を返す

2，2^2，2^3をx, y, zにそれぞれ代入する

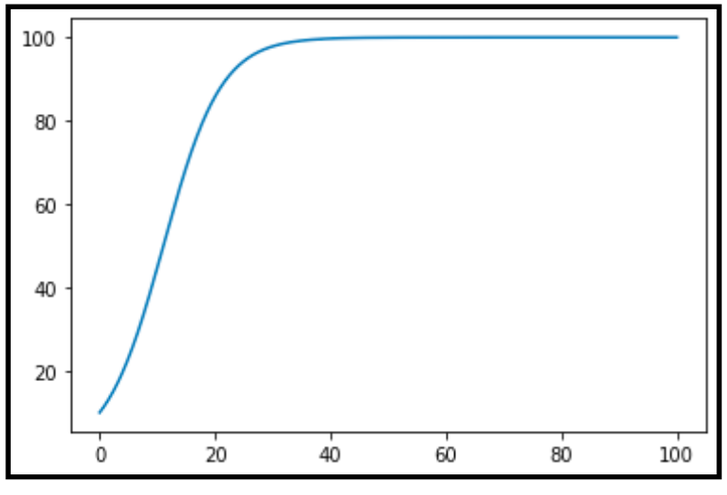
```
# 出力
2 4 8
```

うまく組み合わせて使いやすい関数を作ってみよう

ロジスティック成長モデルのシミュレーションを関数を使って書き換えてみる

```python
# 01-08. ロジスティック成長（数値解）
import matplotlib.pyplot as plt

def logistic_growth(r, K, x0, t_end, dt = 0.1):
  dt = dt
  x = x0
  t_list = [0]
  x_list = [x]
  i_end = int(t_end/dt)
  for i in range(i_end):
    t = dt*(i+1)
    x = x + dt*r*(1-x/K)*x

    t_list.append(t)
    x_list.append(x)

  return t_list, x_list

t_list, x_list = logistic_growth(0.2, 100, 10, 100)

plt.plot(t_list, x_list)
```

ロジスティック成長モデルの関数
t_listとx_listを戻り値としてもつ



ロトカ-ボルテラ モデルのシミュレーションとプロット

被食-捕食系

```
# 02-01. 被食–捕食系
# モデルのパラメータ
a = 2.0
b = 3.0
c = 1.0
d = 2.0

# 初期値
x = 0.4
y = 0.4
t = 0.0

# 時間の設定
dt = 0.0001
t_end = 10
i_end = int(t_end/dt)+1
```

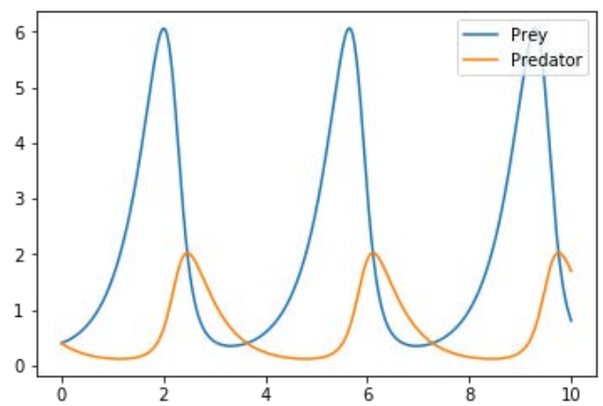
被食者 $\frac{dx}{dt} = (a - by)x$

捕食者 $\frac{dy}{dt} = (cx - d)y$

```
t_list = [t]
x_list = [x]
y_list = [y]
for i in range(i_end):
    t = dt*(i+1)
    x_new = x + dt*(a-b*y)*x
    y_new = y + dt*(c*x-d)*y
    x = x_new
    y = y_new

    t_list.append(t)
    x_list.append(x)
    y_list.append(y)

# 時間発展のプロット
plt.plot(t_list, x_list)
plt.plot(t_list, y_list)
plt.legend(["Prey", "Predator"],
           loc="upper right")
```

matplotlib.pyplot
- legend(ラベル, loc = 表示位置)：凡例を追加する
  ラベルは複数あればリストで渡す.

より詳しく知りたい人は,
matplotlibの公式ドキュメント参照！



オイラー法による離散化：被食-捕食系

$$\begin{cases} \dfrac{dx}{dt} = ax - bxy \\ \dfrac{dy}{dt} = cxy - dy \end{cases}$$

微分の近似（Δtは十分小さいとする）

$$\frac{dx}{dt} \approx \frac{x(t+\Delta t) - x(t)}{\Delta t}$$

$$ax(t) - bx(t)y(t) \approx \frac{x(t+\Delta t) - x(t)}{\Delta t}$$

式を整理

$$x(t+\Delta t) \approx x(t) + \Delta t \left(ax(t) - bx(t)y(t)\right)$$

$x(0) = x_0$ とし，　$x_1, x_2, \ldots, x_n$

また，　$t_n = \Delta t \cdot n$

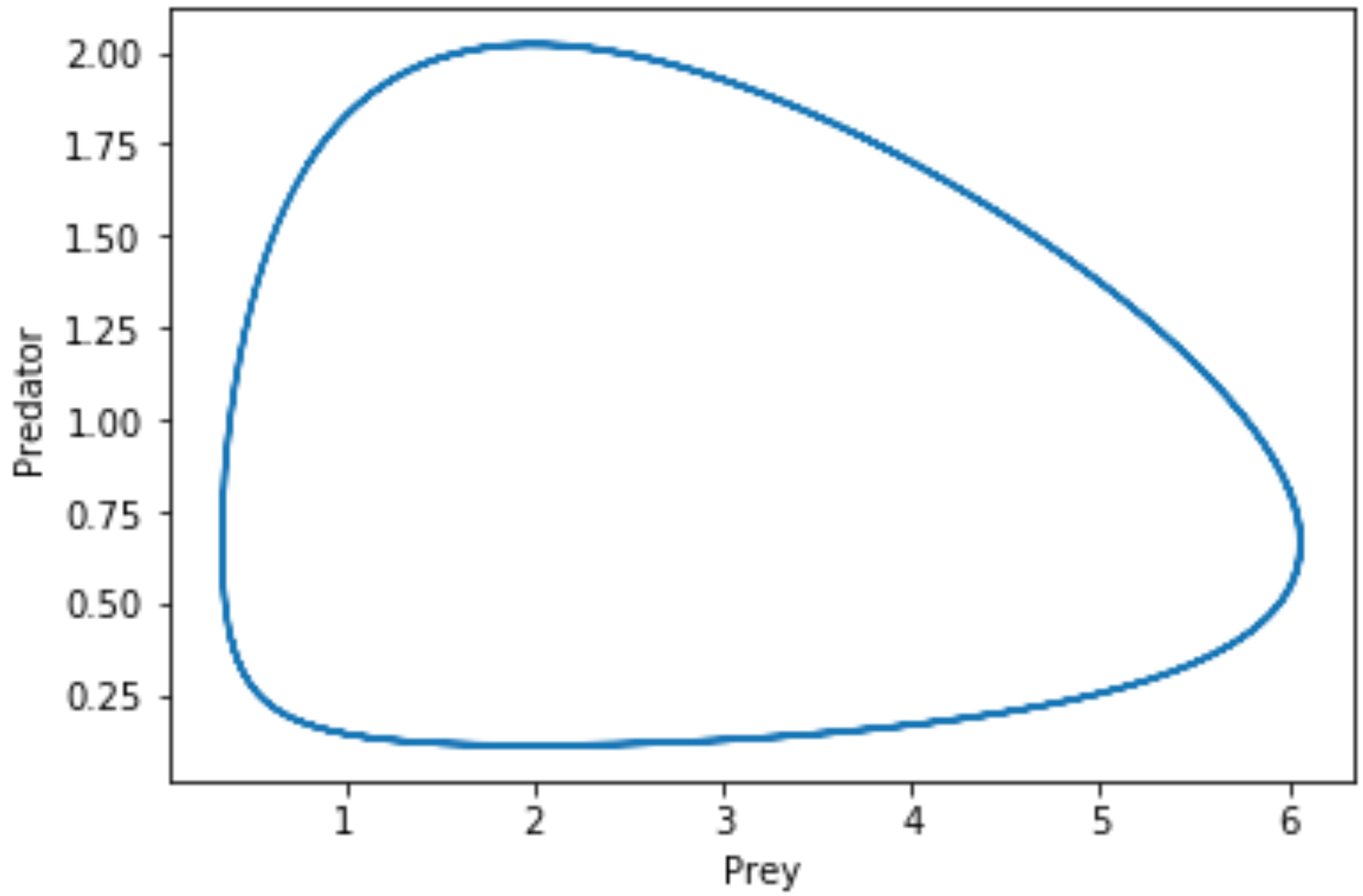
$$x_{n+1} \approx x_n + \Delta t \left(ax_n - bx_n y_n\right)$$

$$X_{n+1} = X_n + \Delta t \left(aX_n - bX_n Y_n\right)$$

$$Y_{n+1} = Y_n + \Delta t \left(cX_n Y_n - dY_n\right)$$

相図 phase diagram

```
# 02-02. 相図 被食-捕食系


plt.plot(x_list, y_list)
plt.xlabel("Prey")
plt.ylabel("Predator")
```



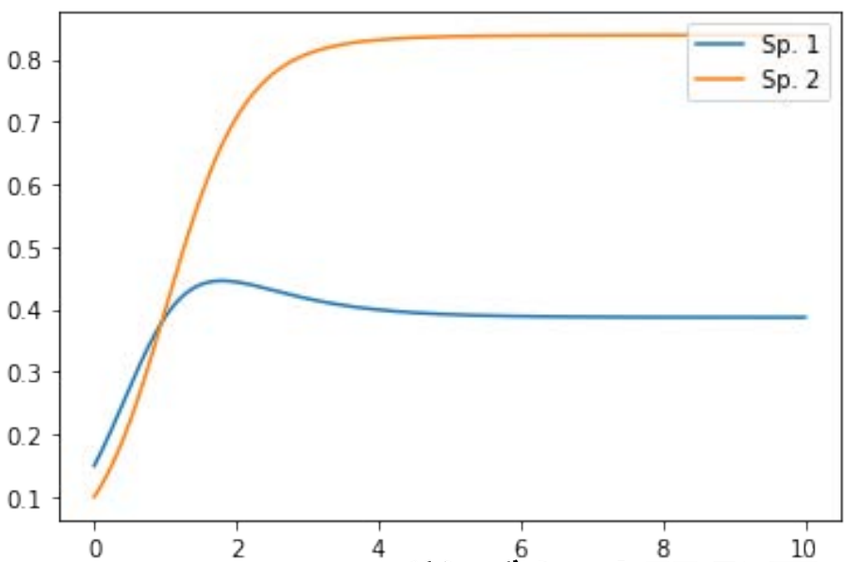
相平面上での軌道を可視化できる.

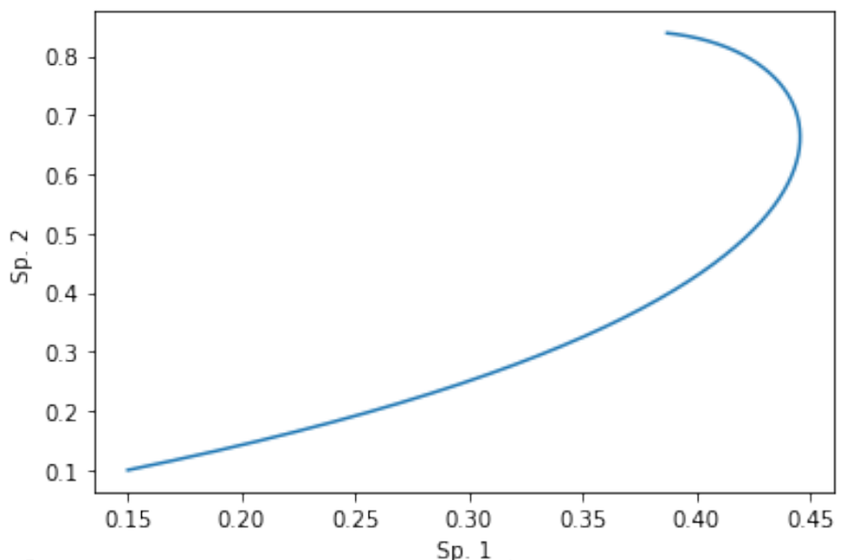
競争系

$$\begin{cases} \dfrac{dx}{dt} = ax - bx^2 - cxy \\ \dfrac{dy}{dt} = dy - exy - fy^2 \end{cases}$$

被食-捕食系の場合を参考に
時間発展や相図に関するプログラムを作成してください

02-03. 競争系





ここに挙げた時間発展や相図はあくまで一例.

これ以外の挙動も示すので色々なパラメータを試してみよう.

オイラー法による離散化：競争系

$$\begin{cases} \dfrac{dx}{dt} = ax - bx^2 - cxy \\ \dfrac{dy}{dt} = dy - exy - fy^2 \end{cases}$$

微分の近似（Δtは十分小さいとする）

$$\frac{dx}{dt} \approx \frac{x(t+\Delta t) - x(t)}{\Delta t}$$

$$ax(t) - bx^2(t) - cx(t)y(t) \approx \frac{x(t+\Delta t) - x(t)}{\Delta t}$$

式を整理

$$x(t+\Delta t) \approx x(t) + \Delta t \left(ax(t) - bx^2(t) - cx(t)y(t)\right)$$

$x(0) = x_0$ とし，　$x_1, x_2, \ldots, x_n$

また，　$t_n = \Delta t \cdot n$

$$x_{n+1} \approx x_n + \Delta t \left(ax_n - bx_n^2 - cx_n y_n\right)$$

$$X_{n+1} = X_n + \Delta t \left(aX_n - bX_n^2 - cX_n Y_n\right)$$

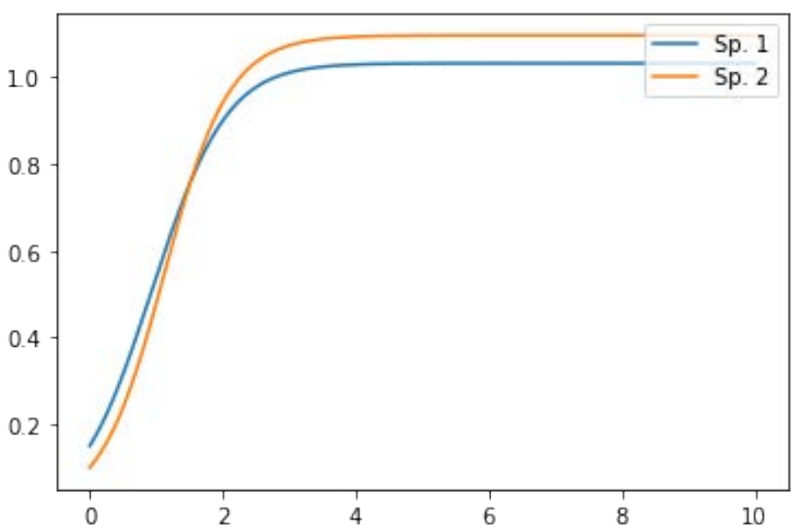
$$Y_{n+1} = Y_n + \Delta t \left(dY_n - eX_n Y_n - fY_n^2\right)$$

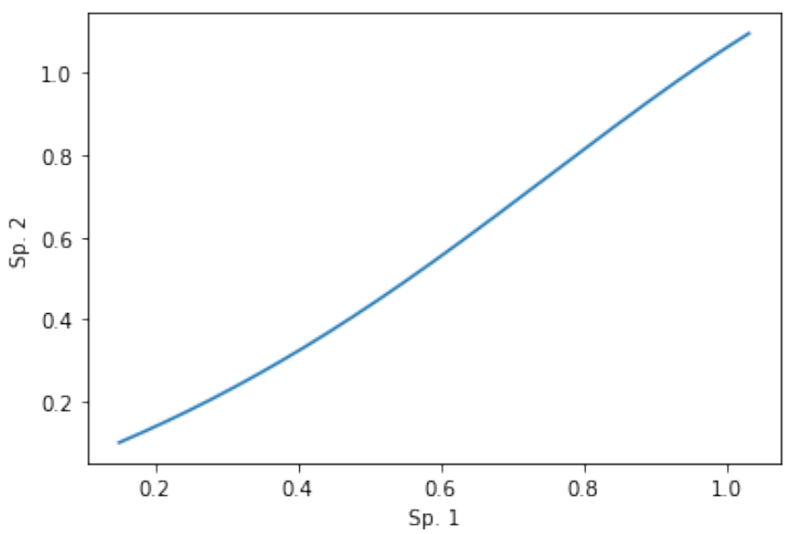
共生系

$$\begin{cases} \dfrac{dx}{dt} = ax - bx^2 + cxy \\ \dfrac{dy}{dt} = dy + exy - fy^2 \end{cases}$$

被食-捕食系の場合を参考に
時間発展や相図に関するプログラムを作成してください

02-04. 共生系





ここに挙げた時間発展や相図はあくまで一例.
これ以外の挙動も示すので色々なパラメータを試してみよう.

本日の課題 ノーマル

競争系か共生系のどちらかやればOK

1. 競争系の平衡点の局所安定性を解析的に調べ，考察せよ.
2. 1.の解析の結果から，観察されることが期待される系の挙動をすべて数値的に再現せよ.
3. 共生系の平衡点の局所安定性を解析的に調べ，考察せよ.
4. 3.の解析の結果から，観察されることが期待される系の挙動をすべて数値的に再現せよ.
5. 質問，意見，要望等をどうぞ.

競争系（1＋2のセット），もしくは，共生系（3＋4のセット）のいずれかに取り組めばOK

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．05_n.ipynb

次回予告

第６回：ランダムな現象：遺伝的浮動，ライト-フィッシャーモデル

5月29日

予習・復習推奨

- 遺伝的浮動
- ライト-フィッシャー モデル

以降は興味ある人のみ

本日の課題 ハード

1. 競争系，共生系について，ニュートン法を用いて数値的に平衡点を求めよ.
2. [被食-捕食系，競争系，共生系のいずれかについて取り組めば十分（もちろん全部やってもOK）]
   相図（相平面上の軌道と平衡点）をプロットせよ．また，そのアイソクラインも重ねてプロットせよ.

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．05_h.ipynb

ニュートン法

ニュートン法 Newton's method（1）

方程式を解くためのアルゴリズム

解を求めたい方程式を $f(x)=0$ とすれば，解は $f(x)$ とx軸との交点になる.

では，どうやって「数値的に」求めるか？

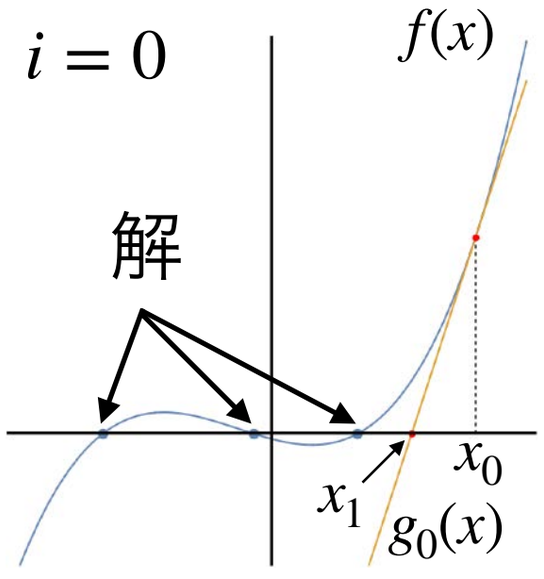
方針

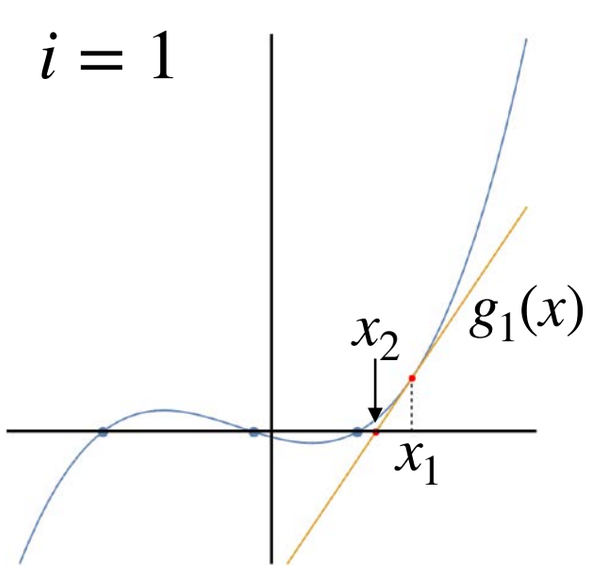
1. 解の近似値をx_iとし，適当なその初期値 x_0を決める
2. 解の近似値 x_iでの接線 $g(x)$ を求める
3. この接線$g(x)$とx軸との交点（$g(x)=0$を満たすx）を求める
4. 交点のx座標を新たに近似値 x_{i+1} として採用する

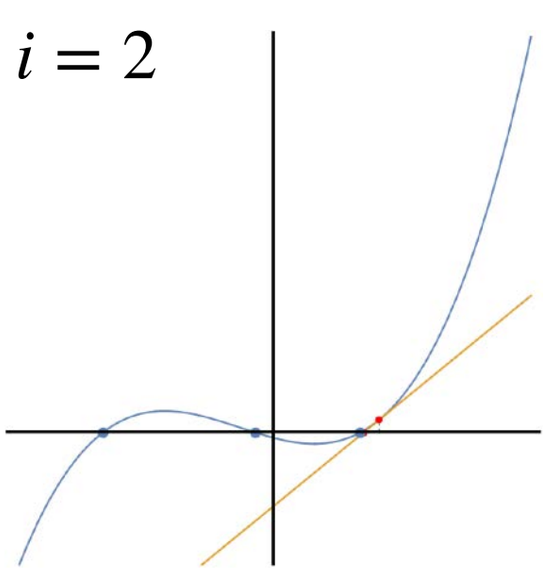
・以後，近似値が収束するまで2.～4.を繰り返す.

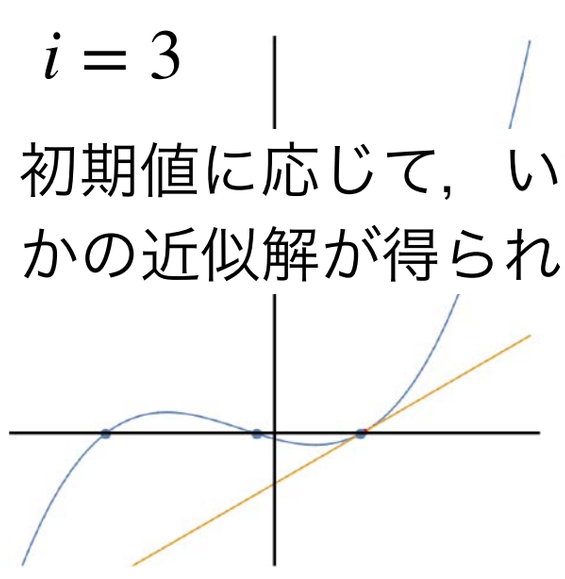
$g(x)$は$f(x)$を用いて表現可
$g(x)=f(x_i)+f'(x_i)(x-x_i)$

$g(x)=0$を満たすxをx_{i+1}とする
$f(x_i)+f'(x_i)(x_{i+1}-x_i)=0$.
なので，
整理すると$x_{i+1}=x_i-\dfrac{f(x_i)}{f'(x_i)}$









初期値に応じて，いずれかの近似解が得られる

ニュートン法 Newton's method（2）

ロジスティック成長モデル

$$X_{t+1} = X_t + r\left(1 - \frac{X_t}{K}\right) X_t$$

r：内的自然増加率．個体数が十分小さい場合（$X \approx 0$）の１世代あたりの増殖率．$r \geq 0$.

K：環境収容力．ある環境で維持されうる個体数，$K > 0$.

このモデルの平衡点（$r\left(1 - \frac{\bar{X}}{K}\right)\bar{X} = 0$を満たす$\bar{X}$）を数値的に求めたい.

考えること

- ロジスティック成長モデルにおけるニュートン法で用いる漸化式

  $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$は具体的にはどのような形になるか？

- どのようなループを組めばよいか？

いろいろな初期値からスタートして，両方の平衡点を求めてみよう.

多次元ニュートン法については,

https://koji.noshita.net/materials/compbio/compbio2018/06/NM.pdf

を参照

平衡点を数値的に見つける：ロジスティック成長モデル

ロジスティック成長モデル

$$\frac{dx}{dt} = r\left(1 - \frac{x}{K}\right)x$$

解析解 $x_1^* = 0, x_2^* = K$

xにおける接線の傾き

$$f'(x) = r\left(1 - \frac{2}{K}x\right)$$

ニュートン法に利用する漸化式

$$x_{i+1} = \frac{x_i^2}{2x_i - K}$$

```
# 03-01. ロジスティック成長モデル

# パラメータ設定
K = 100

# 初期値
x = 90


for i in range(1000):
    x = x*x/(2*x-K)

print("解析解: ", 0, ",", K)
print("近似解: ", x)
```

```
# 出力
解析解:  0 , 100
近似解:  100.0
```

初期値に応じて,
平衡点のいずれかが数値的に求まる

平衡点を数値的に見つける：被食-捕食系

被食-捕食系

$$\begin{cases} \frac{dx}{dt} = ax - bxy \\ \frac{dy}{dt} = cxy - dy \end{cases}$$

解析解

$$(x_1^*, y_1^*) = (0,0), (x_2^*, y_2^*) = \left(\frac{d}{c}, \frac{a}{b}\right)$$

ヤコビ行列

$$\mathbf{J}(x,y) = \begin{pmatrix} \frac{\partial f_1}{dx} & \frac{\partial f_1}{dy} \\ \frac{\partial f_2}{dx} & \frac{\partial f_2}{dy} \end{pmatrix} = \begin{pmatrix} a - by & -bx \\ cy & cx - d \end{pmatrix}$$

ニュートン法に利用する漸化式

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{J}_{\mathbf{x}_i}^{-1}\mathbf{f}(\mathbf{x}_i)$$

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \frac{x_i y_i}{acx_i + bdy_i - ad} \begin{pmatrix} bd \\ ac \end{pmatrix}$$

```python
# 03-02. 被食–捕食系

# パラメータ設定
a = 2
b = 3
c = 1
d = 2

# 初期値
x = 1.5
y = 0.5

for i in range(1000):
    x = b*d*x*y/(a*c*x+b*d*y-a*d)
    y = a*c*x*y/(a*c*x+b*d*y-a*d)

print("解析解: ", (0,0), ",", (d/c, a/b))
print("近似解: ", (x, y)
```

```
# 出力
解析解:  (0, 0) , (2.0, 0.6666666666666666)
近似解:  (2.0, 0.6666666666666666)
```

アイソクライン

アイソクライン isocline

２種系の場合だと，$\frac{dx}{dt} = 0$，または $\frac{dy}{dt} = 0$ を満たす直線や曲線のこと

アイソクラインを境界として個体数の増減のパタンが変わるので，
これを相平面上にプロットできると全体のダイナミクスを見通しやすくなる

被食-捕食系

$$\begin{cases} \frac{dx}{dt} = ax - bxy \\ \frac{dy}{dt} = cxy - dy \end{cases}$$

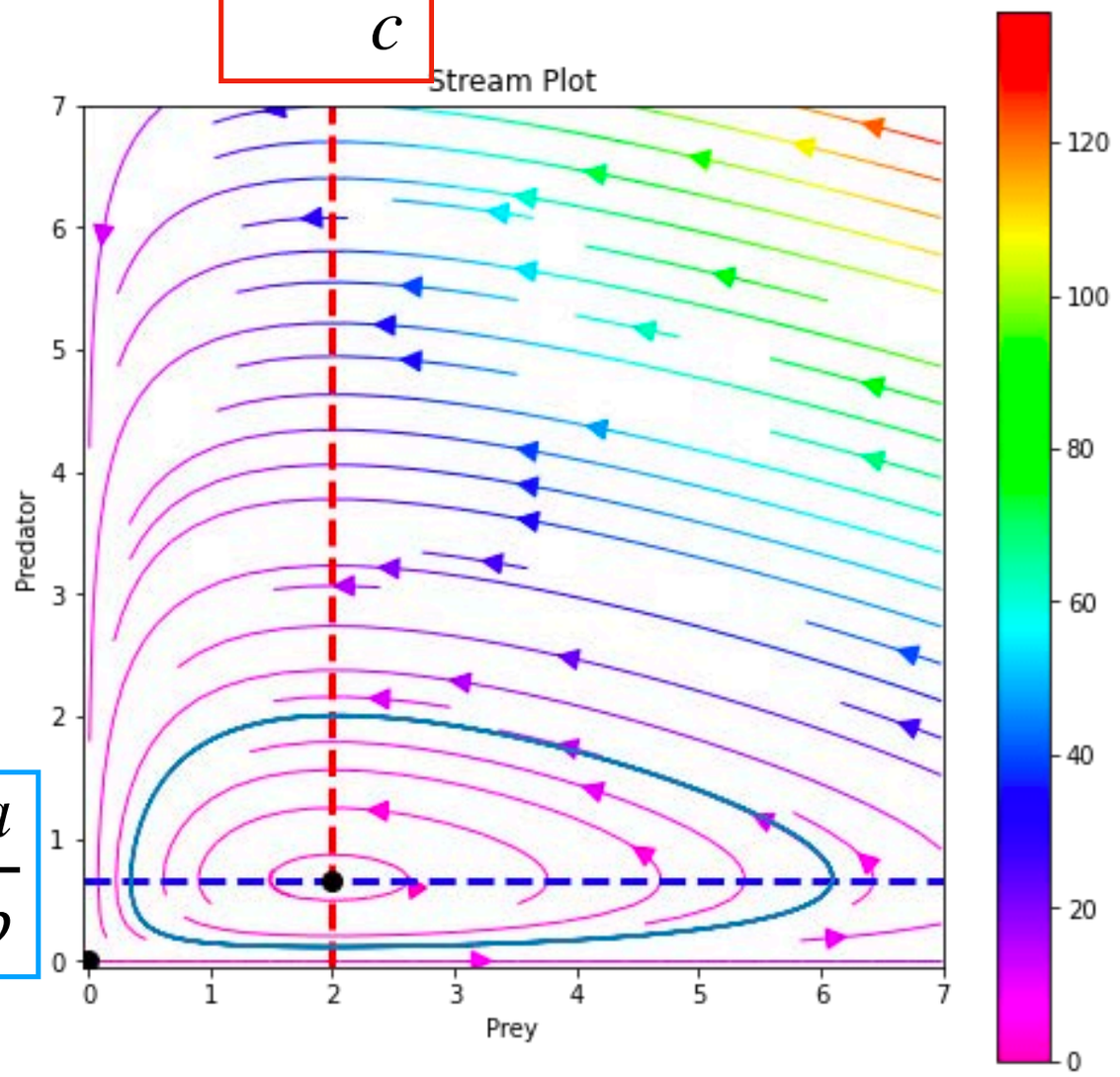
アイソクライン

$$\frac{dx}{dt} = 0 \rightarrow y = \frac{a}{b}$$

$$\frac{dy}{dt} = 0 \rightarrow x = \frac{d}{c}$$

$x = \frac{d}{c}$

$y = \frac{a}{b}$



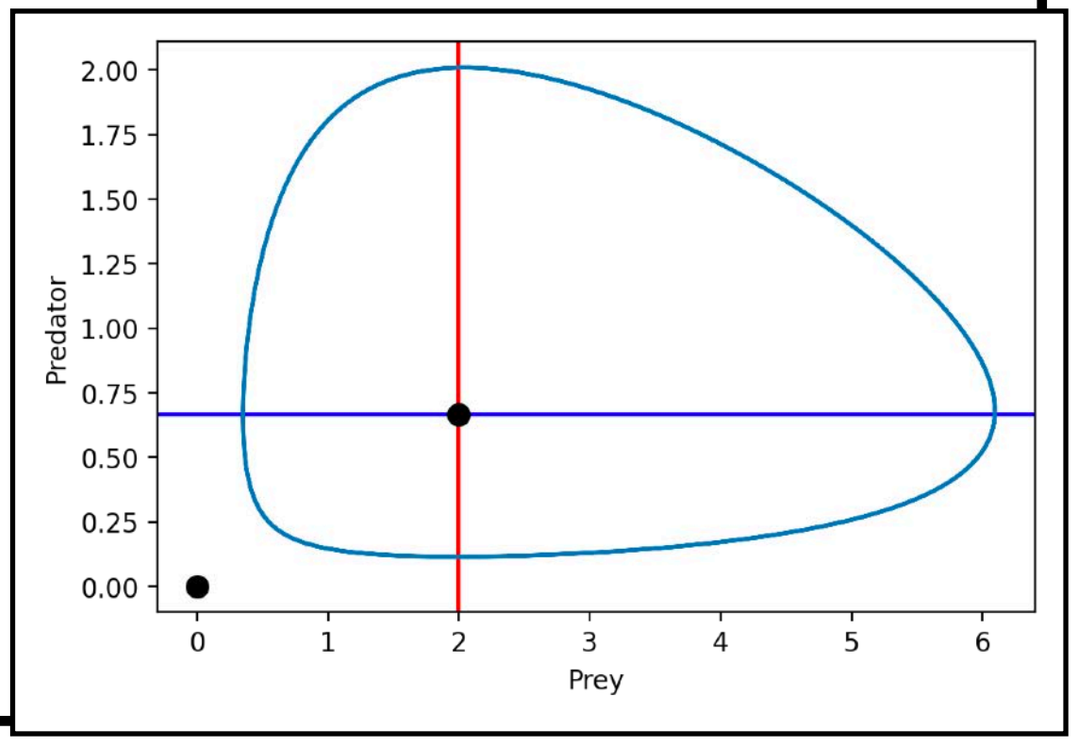
ロトカ-ボルテラ モデルの相図とアイソクラインを使ったダイナミクスの解析

```
# 03-03. 被食–捕食系のダイナミクスとアイソクライン
# モデルのパラメータ
a = 2.0
b = 3.0
c = 1.0
d = 2.0

# 初期値
x = 0.4
y = 0.4
t = 0.0

# 時間の設定
dt = 0.0001
t_end = 10
i_end = int(t_end/dt)+1

t_list = [t]
x_list = [x]
y_list = [y]
for i in range(i_end):
    t = dt*(i+1)
    x = x + dt*(a-b*y)*x
    y = y + dt*(c*x-d)*y

    t_list.append(t)
    x_list.append(x)
    y_list.append(y)
```

被食-捕食系の場合は，アイソクラインは水平，垂直の直線なのでplotではなくaxhline，axvlineを使っている．

平衡点を黒い丸でプロット，マーカーサイズも指定して視認性を上げている

```
# アイソクライン
plt.axhline(a/b, color = "b")
plt.axvline(d/c, color = "r")

# 平衡点
plt.plot(0,0, "ko",d/c, a/b ,"ko", markersize = 8)

# 相平面上でのダイナミクス
plt.plot(x_list, y_list)
plt.xlabel("Prey")
plt.ylabel("Predator")
```



左半分は，被食-捕食系のシミュレーション（演習資料台５回 #02-01 ）と（プロットの部分を除き）同じ

matplotlib.pyplot
- axhline(yの値，オプション)：水平な直線をプロット
- axvline(xの値，オプション)：垂直な直線をプロット
  ここでは色のオプションで青（b）と赤（r）を指定