数理生物学演習

第4回　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4回：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（２種系）

被食-捕食系

被食者 $\dfrac{dx}{dt} = ax - bxy$

捕食者 $\dfrac{dy}{dt} = cxy - dy$

- 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

$\mathbf{x}$に関する十分小さな”ずれ”$\mathbf{n}$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

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

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

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

$\mathbf{n}$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$\mathbf{M}$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.

- 固有値の全て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

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

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

$$\begin{aligned} f_1(x,y) &= ax - bxy = 0 \\ f_2(x,y) &= cxy - dy = 0 \end{aligned} \quad \Rightarrow \quad (x^*, y^*) = (0,0), \left(\frac{d}{c}, \frac{a}{b}\right)$$

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

$\lambda_1 = a > 0,\ \lambda_2 = -d < 0$ ⇒ $(x^*, y^*) = (0,0)$ のとき

常に不安定

より詳しくいえば，鞍点（あんてん）saddle

実際にプログラムを組んでみよう！

型変換

明示的型変換 キャスト　計算の途中で一時的に変数の型を変更したい場合，等

- str(x)：xの文字列（str型）を返す
- int(x)：実数や文字列xに対してint型を返す
- float(x)：実数や文字列xに対してfloat型を返す

他にもbool型やcomplex型についても，キャストする関数がある．
詳しく知りたい人は調べてみて！

```
#4-1. キャスト
## 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（１）

- Pythonおける関数とは，ある処理を行うモジュールのこと
- これまで使ってきた，print()やmath.exp()などは関数
- 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

```
# 4-2．シンプルな関数
def simplefunc():
    print("関数を呼び出しました．")

simplefunc()
```

「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

```
# 4-3．シンプルな関数 その２
def simplefunc2():
    print("１．関数を")
    print("２．呼び出し")
    print("３．ました．")

simplefunc2()
```

```
# 出力
１．関数を
２．呼び出し
３．ました．
```

繰り返し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 = math.exp(2.0)

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

偶数ならeven，奇数ならoddを表示させる関数

```
# 4-4．引数をもつ関数
def myprint(i):
    if(i%2==0):
        print("even\n")
    else:
        print("odd\n")

for i in range(10):
    myprint(i)
```

```
# 出力
even
odd
even
odd
⋮
odd
```

```
# 4-5．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)
```

２変数の足し算

```
# 出力
6
```

関数（3）：パラメータいろいろ

パラメータのデフォルト値

```
def 関数名(パラメータ1=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

```
# 4-6. 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

被食-捕食系

```
# 4-7. 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
tEnd = 10
iEnd = int(tEnd/dt)+1

xList = [x]
yList = [y]
tList = [t]
```

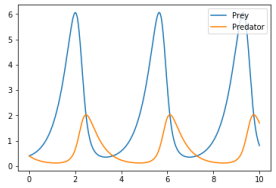
被食者 $\frac{dx}{dt} = (a - by)x$

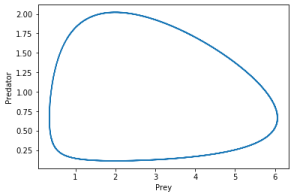
捕食者 $\frac{dy}{dt} = (cx - d)y$

```
for i in range(iEnd):
    t = dt*i
    xx = x + dt*(a-b*y)*x
    yy = y + dt*(c*x-d)*y
    x = xx
    y = yy
    tList.append(t)
    xList.append(x)
    yList.append(y)

# 時間発展のプロット
plt.plot(tList, xList)
plt.plot(tList, yList)
plt.legend(["Prey",
"Predator"],loc='upper right')

# 相図
plt.plot(xList,yList)
plt.xlabel("Prey")
plt.ylabel("Predator")
```

- legend(ラベル, オプション)：凡例を追加する
- xlabel(ラベル名, オプション), ylabel(ラベル名, オプション)：各軸にラベルを追加

より詳しく知りたい人は，
matplotlibの公式ドキュメント参照！





ニュートン法 Newton's method

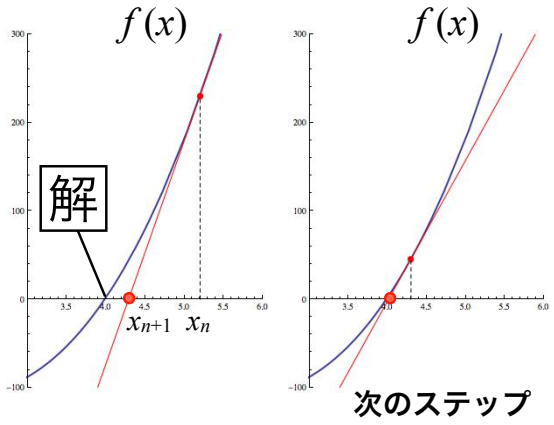
方程式を解くためのアルゴリズム

解を求めたい方程式を $f(x)=0$ とすれば,

解は $f(x)$ と x 軸との交点の x 座標になる.

では，どうやって「数値的に」求めるか？

1.解の近似値を x_n とし，適当なその初期値 x_0 を決める
2.解の近似値 x_n での接線を求める
3.この接線と x 軸との交点を求める
4.交点の x 座標を新たに近似値 x_{n+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.



x_n の漸化式を求めてみてください

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

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

```
# 4-8. 被食-捕食系
# 初期値
x = 90

# パラメータ設定
K = 100

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

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

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

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

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

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

```
# 4-9. 被食-捕食系
# 初期値
x = 1.5
y = 0.5

# パラメータ設定
a = 2
b = 3
c = 1
d = 2

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

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)
```

競争系

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

共生系

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

それぞれ,
被食-捕食系の場合を参考に
プログラムを作成してください
4-10. 競争系，4-11．共生系

本日の課題

注意：氏名，学籍番号，所属を必ず書く！

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．

ハード 5. 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．

6. 質問，意見，要望等をどうぞ．

課題をPDFファイルにまとめて，Moodleにて提出すること

次回予告

第5回：疫学モデル

5月２０日

予習・復習推奨

- SIRモデル
- 基本再生産数

宣伝

数理生物学 第5回

かたちの数理モデル（１）

5月１５日（水）

内容

- 理論形態学
  - “巻き”の理論モデル
  - 分岐の理論モデル