2021 年度卒業研究レポート

生物の個体数変動と微分方程式

明治大学　総合数理学部　現象数理学科
森嶋紀衣

2022 年 2 月 24 日

目 次

第1章　はじめに

私は、大学の授業で「数理生物学」という授業をとり、そこで数理生物学に興味を持った。自然界の生物達の間の関係をモデル化して、その数理を調べるという今では確立された研究分野があって、その一端に触れるのは有意義であるため、典型例である Lotka-Volterra と、それとはやや異なり、数学だけでは解析し切れずシミュレーションの重要性も高くなる魚種交替の 3 すくみのモデルを先生に紹介してもらい、それらについて研究してみた。

第2章　1種の生物における個体数のモデル

この章の内容は Murray[1]、ブラウン [2] による。
まずは、1 種の生物における個体数のモデルについてみていく。

2.1　Malthus の法則

どの生物の種も常に整数単位で増減するから、その増減を微分可能な時間の関数を使ってモデル化することはできない。しかし、個体数が非常に大きい場合、1 つくらいの個体数の変化は全体に対して非常に小さい。そこで、このように個体数が非常に大きい種の個体数の増減は、近似的に時間に関して連続で、微分可能な関数で表せると仮定する。

時刻 t における生物の個体数を $p(t)$ とし、出生率と死亡率の差を $r(t,p)$ する。この生物の集団の外から中へと個体を入れたり、この集団の中から外へ個体を出したりしない場合、個体数の変化率は

$$dp(t)/dt = rp(t)$$

と表せる。最もシンプルなモデルでは、r を定数とすることができる。

$$\frac{dp(t)}{dt} = ap(t), \quad a = \text{定数}$$

これが Malthus の法則として知られているモデルである。時刻 $t = t_0$ における個体数の初期値が $p(0) = p_0$ であるとすると、微分方程式の初期値問題

$$\frac{dp(t)}{dt} = ap(t), \quad p(0) = p_0$$

を満たす。これを解くと、

$$p(t) = p_0 e^{a(t-t_0)}$$

となる。Malthus の法則を満たす生物の種では、もし、$a > 0$ ならば、個体数は指数関数的に増加し、$a < 0$ ならば、個体数は絶滅することになる。

2.2　logistic 方程式

Malthus の法則は最もシンプルなモデルであるがゆえに、常に正確であるとは言えない。なぜならこのモデルは、生物が生きていくために必要な資源や居住空間や食料は限られている。これらをめぐり生物間で競争が発生することが考慮されていないからだ。したがって、競争に関する項、ある正定数 b を用いて $-bp^2$ を追加する必要がある。単位時間当たりに 2 つの個体が遭遇する回数の統計的平均は p^2 に比例するからである。この項を追加すると次のようになる。

$$\frac{dp(t)}{dt} = ap - bp^2$$

これが logistic 方程式として知られているモデルである。現在では、b は一般に a に比べて非常に小さいとされている。p が極めて大きくない場合、つまり個体数が非常に大きくない場合では、増加係数 $-bp^2$ は ap

に比べて無視できるくらい小さく、個体数は指数関数的に増加していく。p が極めて大きい場合、つまり個体数が非常に大きい場合では、増加係数 $-bp^2$ は無視できなくなり、個体数の増加が抑制される。係数 b は、居住空間が広いほど、食料や資源が豊富であるほど、競争は起こりにくいので、小さくなり、居住空間が狭いほど、食料や資源が欠乏しているほど、競争は起こりやすいので、大きくなる。

この logistic 方程式を用いて、個体数の変動を見ていく。時刻 $t = t_0$ における個体数の初期値が $p(0) = p_0$ であるとすると、微分方程式の初期値問題

$$\frac{dp(t)}{dt} = ap - bp^2, \quad p(0) = p_0$$

を満たす。これは変数分離法で解くことができる。これを解くと、

$$p(t) = \frac{ap_0}{bp_0 + (a - bp_0)e^{-a(t-t_0)}}$$

となる。$t \to \infty$ としたとき、個体数は

$$p(t) \to \frac{ap_0}{bp_0} = \frac{a}{b}$$

となり、個体数は初期値 p_0 によらず、常に a/b に近づくことがわかる。

第3章　2種の生物における個体数のモデル

この章の内容はマレー [1]、ブラウン [3] による。
次に、2 種の生物における個体数のモデルについてみていく。

3.1　Lotoka-Volterra 方程式

複数の生物種が相互作用すると、各々の種の個体群動態は互いに影響を受ける。2 つの生物種における相互作用は、主に 3 種類のモデルが存在する。
(1) 捕食者–被食者モデル
一方の個体群の個体数の増加率は増加するが、もう一方の個体群の個体数の増加率は減少する。
(2) 競争モデル
どちらの個体群の個体数の増加率も減少する。
(3) 相利共生モデル
どちらの個体群の個体数の増加率も増加する。

ここでは (1) の捕食者–被食者モデルについて扱っていく。時刻 t における被食者個体数を $x(t)$ とし、捕食者個体数を $y(t)$ とする。
まず、被食者の個体数の変化率について考える。
(i) 捕食者がいないとき
被食者はマルサスの法則にしたがって増加する。
(ii) 捕食者がいるとき
捕食されることにより被食者は、捕食者と被食者の個体数に比例して減少する。
したがって被食者の個体数の変化率は正定数 a、b を用いて $dx/dt = ax - bxy$ となる。

次に捕食者の個体数の変化率について考える。
(i) 被食者がいないとき
食料がないので捕食者は指数関数的に衰退していく。
(ii) 被食者がいるとき
被食者と捕食者が遭遇する回数に比例して捕食者は増加する。
したがって捕食者の個体数の変化率は正定数 c、d を用いて $dy/dt = -cy + dxy$ となる。
よって捕食者–被食者モデルは

$$\begin{cases} \dfrac{dx}{dt} = ax - bxy \\ \dfrac{dy}{dt} = -cy + dxy \end{cases} \tag{3.1}$$

となる。これが Lotoka-Volterra 方程式として知られているモデルである。

3.2 Lotoka-Volterra 方程式の平衡点の安定性

連立微分方程式 (3.1) の平衡点を求める。連立方程式

$$\begin{cases} ax - bxy = 0 \\ -cy + dxy = 0 \end{cases}$$

を解くと、平衡点は $(x, y) = (0, 0), (\frac{c}{d}, \frac{a}{b})$ の 2 点となる。

次に平衡点の安定性を調べる。連立微分方程式 (3.1) のヤコビ行列は、

$$J(x, y) = \begin{bmatrix} a - by & -bx \\ dy & dx - c \end{bmatrix}$$

となる。平衡点が $(x, y) = (0, 0), (\frac{c}{d}, \frac{a}{b})$ のそれぞれのときの安定性を調べていく。

$(x, y) = (0, 0)$ のとき、ヤコビ行列は、

$$J(0, 0) = \begin{bmatrix} a & 0 \\ 0 & -c \end{bmatrix}$$

となり、固有値は $a, -c$ である。$\mathrm{Re}(a) > 0, \mathrm{Re}(-c) < 0$ であるから、$(0, 0)$ は鞍点である。したがって平衡点 $(0, 0)$ は常に不安定である。

$(x, y) = (\frac{c}{d}, \frac{a}{b})$ のとき、ヤコビ行列は、

$$J\left(\frac{c}{d}, \frac{a}{b}\right) = \begin{bmatrix} 0 & -\dfrac{bc}{d} \\ \dfrac{ad}{b} & 0 \end{bmatrix}$$

となり、固有値は $i\sqrt{ac}, -i\sqrt{ac}$ である。$\mathrm{Re}(i\sqrt{ac}) = \mathrm{Re}(-i\sqrt{ac}) = 0$ である。Lotka-Volterra 方程式の解軌道は、閉軌道である (次の 3.3 参照) ので、平衡点 $(\frac{c}{d}, \frac{a}{b})$ は渦心点である。したがって平衡点 $(\frac{c}{d}, \frac{a}{b})$ は中立的に安定である。

3.3 Lotoka-Volterra 方程式の解軌道

(3.1) は解 $x(t) = x_0 e^{at}, y(t) = 0$ と $x(t) = 0, y(t) = y_0 e^{-ct}$ をもつ。x 軸 y 軸ともに (3.1) の軌道である。つまり、$t = t_0$ において、第 1 象限から始まる (3.1) の全ての解 $x(t), y(t)$ は、軌道の一意性から、未来の $t > t_0$ においても第 1 象限にあり続けることを意味している。

軌道の一意性

$(x_1(t), y_1(t))$ と $(x_2(t), y_2(t))$ が微分方程式

$$\begin{cases} \dfrac{dx_1}{dt} = f(x_1, y_1) \\ \dfrac{dy_1}{dt} = g(x_1, y_1) \end{cases}, \quad \begin{cases} \dfrac{dx_2}{dt} = f(x_2, y_2) \\ \dfrac{dy_2}{dt} = g(x_2, y_2) \end{cases}$$

を満たしており、ある時刻 t_1, t_2 において $x_1(t_1) = x_2(t_2), y_1(t_1) = y_2(t_2)$ を満たし、ある一点を共有しているならば、$(x_1(t), y_1(t))$ と $(x_2(t), y_2(t))$ の軌道は一致することを示す。

証明）

$$\begin{cases} x_3(t) = x_1(t + t_1) \\ y_3(t) = y_1(t + t_1) \end{cases}, \quad \begin{cases} x_4(t) = x_2(t + t_2) \\ y_4(t) = y_2(t + t_2) \end{cases}$$

とおき、$t = 0$ を代入すると、

$$\begin{cases} x_3(0) = x_1(t_1) \\ y_3(0) = y_1(t_1) \end{cases}, \quad \begin{cases} x_4(0) = x_2(t_2) \\ y_4(0) = y_2(t_2) \end{cases}$$

となり、条件より $x_1(t_1) = x_2(t_2), y_1(t_1) = y_2(t_2)$ であるから、

$$x_3(0) = x_4(0), y_3(0) = y_4(0)$$

初期値が一致する。また、

$$\frac{dx_3}{dt}(t) = \frac{dx_1}{dt}(t + t_1) = f(x_1(t + t_1), y_1(t + t_1)) = f(x_3(t), y_3(t))$$

$$\frac{dy_3}{dt}(t) = \frac{dy_1}{dt}(t + t_1) = g(x_1(t + t_1), y_1(t + t_1)) = g(x_3(t), y_3(t))$$

$$\frac{dx_4}{dt}(t) = \frac{dx_2}{dt}(t + t_2) = f(x_2(t + t_2), y_2(t + t_2)) = f(x_4(t), y_4(t))$$

$$\frac{dy_4}{dt}(t) = \frac{dy_2}{dt}(t + t_2) = g(x_2(t + t_2), y_2(t + t_2)) = g(x_4(t), y_4(t))$$

であり、解の一意性から、

$$(x_3(t), y_3(t)) = (x_4(t), y_4(t))$$

$$(x_1(t + t_1), y_1(t + t_1)) = (x_2(t + t_2), y_2(t + t_2))$$

$$(x_1(t), y_1(t)) = (x_2(t + t_2 - t_1), y_2(t + t_2 - t_1))$$

よって $(x_1(t), y_1(t))$ と $(x_2(t), y_2(t))$ の軌道は一致する。

上の証明から、$x(0) > 0$ かつ $y(0) > 0$ ならば、任意の t について $x(t) > 0$ かつ $y(t) > 0$ が成り立つ。

$x, y \neq 0$ のとき、(3.1) の軌道は、

$$\frac{dx}{dt} = ax - bxy, \quad \frac{dy}{dt} = -cy + dxy$$

から

$$x(a - by)\frac{dy}{dt} = (ax - bxy)(-cy + dxy) = y(-c + dx)\frac{dx}{dt}$$

故に

$$\frac{a-by}{y}\frac{dy}{dt} = \frac{-c+dx}{x}\frac{dx}{dt}$$

この両辺を t について積分すると、置換積分の公式から

$$\frac{dy}{dx} = \frac{-cy+dxy}{ax-bxy} = \frac{y(-c+dx)}{x(a-by)}$$

の解曲線である。この方程式を変形すると、

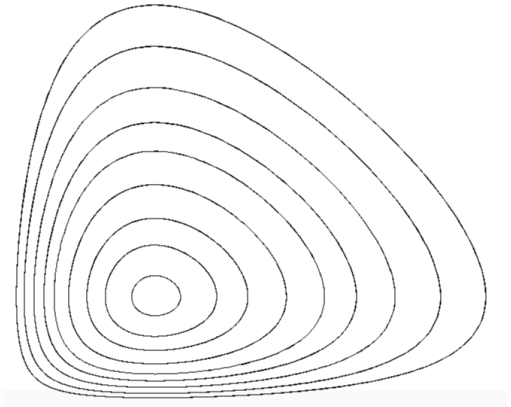
$$\frac{a-by}{y}\frac{dy}{dx} = \frac{-c+dx}{x}$$

の形にできる。これは変数分離系である。したがって、ある定数 k に対して、 $a\log y - by + c\log x - dx = k$ が成り立つ。この方程式の両辺の指数をとると、

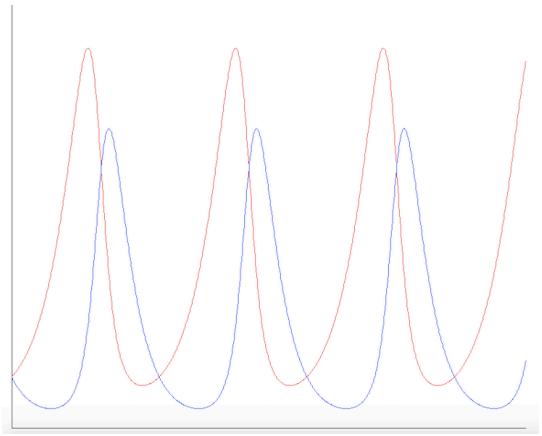
$$\frac{y^a}{e^{by}}\frac{x^c}{e^{dx}} = K \tag{3.2}$$

を得る。ここで、K は定数である。よって、(3.1) の軌道は、(3.2) で定義された曲線群であり、閉じた曲線になる。

下の図は $a=2, b=1, c=3, d=1$ のときの解軌道と初期条件 $x=1, y=1$ のときの解の時間変化をシミュレーションした結果である。



(a) 解軌道



(b) 解の時間変化 (x : 赤, y : 青)

3.4 ダンコナの疑問 ～なぜ第一次世界大戦中、地中海で捕獲されるサメの割合が増えたのか～

1920 年代半ば、イタリアの生物学者ダンコナ (Umberto D'Ancona) は、互いに影響し合う魚類の個体数変動について研究を行っていた。ダンコナは戦争中に食用魚としてあまり適当でないサメやエイなどの軟骨魚類の漁獲量率が大幅に増えていることを発見した。イタリアでは、第一次世界大戦が 1914 年～1918 年にかけて行われており、下の表 3.1 を見ると、ちょうどその時期の軟骨魚類の漁獲量率が高いことがわかる。

表 3.1: イタリアのフューメ港での軟骨魚類の漁獲量率のデータ

1914 年	1915 年	1916 年	1917 年	1918 年	1919 年	1920 年	1921 年	1922 年	1923 年
11..9 %	21.4 %	22.1 %	21.2 %	36.4 %	27.3 %	16.0 %	15.9 %	14.8 %	10.7 %

軟骨魚類と食用魚の関係は、軟骨魚類が捕食者であり、食用魚が被食者である捕食者–被食者関係である。ダンコナは漁業操業が落ち込むことにより、軟骨魚類の食料である食用魚が増え、軟骨魚類が増殖したのではないか、と考えた。しかしこの考えでは、漁業操業の落ち込みにより軟骨魚類が増えることを説明しているだけで、被食者よりも捕食者に有利に働くかを説明できていない。

そこで、軟骨魚類と食用魚の個体数変化を Lotoka-Volterra 方程式を用いてこの問題を分析していく。被食者個体数を $x(t)$ とし、捕食者個体数を $y(t)$ とする。

まずは、漁業操業が行われない時期モデルを考えていく。漁業操業が行われない時期は、

$$\begin{cases} \dfrac{dx}{dt} = ax - bxy \\ \dfrac{dy}{dt} = -cy + dxy \end{cases} \tag{3.1}$$

となる。

ダンコナのデータは、実際には捕食者割合の 1 年毎の平均である。データと (3.1) の予測と比較するために (3.1) の任意の解 $x(t), y(t)$ に対して $x(t), y(t)$ の平均値を計算する。$x(t), y(t)$ を正確に計算できなくても、これらの平均値は次のようにして求めることができる。

(3.1) の解 $x(t), y(t)$ は、平衡点でない限りつねに周期関数であるので、$x(t), y(t)$ を (3.1) の周期解とし、その周期を $T > 0$ をする。$x(t), y(t)$ の平均値を

$$\overline{x} = \frac{1}{T}\int_0^T x(t)dt \ , \ \ \overline{y} = \frac{1}{T}\int_0^T y(t)dt$$

と定義する。(3.1) の第 1 式の両辺を x で割ると、$\dot{x}/x = a - by$ となるので、

$$\frac{1}{T}\int_0^T \frac{\dot{x}(t)}{x(t)}dt = \frac{1}{T}\int_0^T [a - by(t)]dt$$

となる。$\int_0^T \dot{x}(t)/x(t)dt = \log x(T) - \log x(0)$ であり、$x(t)$ は周期 T の周期解であるから、これは 0 になる。したがって、

$$\frac{1}{T}\int_0^T [a - by(t)]dt = 0$$

$$\frac{1}{T}\int_0^T by(t)dt = \frac{1}{T}\int_0^T adt = a$$

$$\frac{1}{T}\int_0^T y(t)dt = \frac{a}{b}$$

となり、$\overline{y} = a/b$ を得る。同様にすると、$\overline{x} = c/d$ を得る。

次に、漁業操業の影響を取り入れたモデルを考える。漁業操業により、被食者である食用魚の個体数は、ある正定数 ϵ を用いて $\epsilon x(t)$ の割合で減少し、捕食者である軟骨魚類の個体数は $\epsilon y(t)$ の割合で減少する。このことをモデルに取り入れると、

$$\begin{cases} \dfrac{dx}{dt} = ax - bxy - \epsilon x = (a - \epsilon)x - bxy \\ \dfrac{dy}{dt} = -cy + dxy - \epsilon y = -(c + \epsilon)y + dxy \end{cases} \tag{3.3}$$

となる。これは、(3.1) の a を $a - \epsilon$ で置き換え、c を $c + \epsilon$ で置き換えたものと同じである (ただし $a - \epsilon > 0$ のとき)。したがって、このときの $x(t)$ と $y(t)$ の平均値は、

$$\overline{x} = \frac{c + \epsilon}{d} \ , \ \overline{y} = \frac{a - \epsilon}{b}$$

となる。

これら 2 つの結果から、適度な漁業操業 ($\epsilon < a$) では、平均して、被食者である食用魚の個体数は増加し、捕食者である軟骨魚類の個体数は減少することがわかる。逆に、漁業操業が落ち込むと、被食者である食用魚の個体数は減少し、捕食者である軟骨魚類の個体数は増加することがわかる。

第4章　3種の生物における個体数のモデル

この章の内容は、松田 [4] による。
最後に、3 種の生物における個体数のモデルをみていく。

4.1　魚種交替

日本近海の漁業で獲れる魚にはマサバ、マイワシ、カタクチイワシがあるが、これらの漁獲高は経時的に上下動し、1980 年代には豊漁だったマイワシが、現在では少ししか獲れなくなり、代わってカタクチイワシが増えている。このように、主に獲れる魚の種類が替わる現象を魚種交替という。 2 種の競争関係は共存しつつ、永久に変動するような魚種交替を再現することができない (マレー [1] 参照) が、3 種の競争関係が 3 すくみの関係にあれば、理論的に共存しつつ、永久に変動することができる。この 3 すくみの関係について詳しくみていく。

3 すくみの関係とは、どのような関係か、マサバ・マイワシ・カタクチイワシを例に挙げて説明すると、
・マサバが優占している時、マサバに弱いカタクチイワシは増えず、マサバに強いマイワシが増える。
・マイワシが優占している時、マイワシに弱いマサバは増えず、マイワシに強いカタクチイワシが増える。
・カタクチイワシが優占している時、カタクチイワシに弱いマイワシは増えず、カタクチイワシに強いマサバが増える。
このようにじゃんけんと同じ関係のことをいう。

4.2　3 すくみの関係のモデル

3 種の個体数をそれぞれ、種 1 のマサバを $N_1(t)$、種 2 のマイワシを $N_2(t)$、種 3 のカタクチイワシを $N_3(t)$ とする。それぞれの個体数の時間変化が次の力学系で表せたとする。

$$\begin{cases} \dfrac{dN_1(t)}{dt} = c_1 + [r_1 - a_{11}N_1 - a_{12}N_2 - a_{13}N_3]N_1 \\ \dfrac{dN_2(t)}{dt} = c_2 + [r_2 - a_{21}N_1 - a_{22}N_2 - a_{23}N_3]N_2 \\ \dfrac{dN_3(t)}{dt} = c_3 + [r_3 - a_{31}N_1 - a_{32}N_2 - a_{33}N_3]N_3 \end{cases} \tag{4.1}$$

c_i ： 微小な正の定数。相手に侵されない聖域からの補給を表す。
r_i ： 内的自然増加率を表す。
a_{ij}： 種 j が種 i に与える負の影響の強さを表す。

連立微分方程式 (4.1) の平衡点を求める。簡単のため、$c_i = 0$ と仮定する。連立方程式

$$\begin{cases} (r_1 - a_{11}N_1 - a_{12}N_2 - a_{13}N_3)N_1 = 0 \\ (r_2 - a_{21}N_1 - a_{22}N_2 - a_{23}N_3)N_2 = 0 \\ (r_3 - a_{31}N_1 - a_{32}N_2 - a_{33}N_3)N_3 = 0 \end{cases}$$

を解くと、共存平衡点 (N_1^*, N_2^*, N_3^*) は

$$\begin{cases} N_1^* = \dfrac{-a_{23}a_{32}r_1 + a_{22}a_{33}r_1 + a_{13}a_{32}r_2 - a_{12}a_{33}r_2 - a_{13}a_{22}r_3 + a_{12}a_{23}r_3}{D} \\ N_2^* = \dfrac{a_{23}a_{31}r_1 - a_{21}a_{33}r_1 - a_{13}a_{31}r_2 + a_{11}a_{33}r_2 + a_{13}a_{21}r_3 - a_{11}a_{23}r_3}{D} \\ N_3^3 = \dfrac{-a_{22}a_{31}r_1 + a_{21}a_{32}r_1 + a_{12}a_{31}r_2 - a_{11}a_{32}r_2 - a_{12}a_{21}r_3 + a_{11}a_{22}r_3}{D} \end{cases}$$

ただし、$D = (-a_{13}a_{22}a_{31} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} + a_{11}a_{22}a_{33})$ となる。

3 種の関係が 3 すくみの関係になり、時間経過により優占種の入れ替わりがおこるるように、r_i や a_{ij} の範囲を絞っていく。

まず、ほとんど種 1 だけがいて、種 2 と種 3 がごくわずかしかいないときを考える。ここでも簡単のため、$c_i = 0$ と仮定する。

種 2 と種 3 はごくわずかしかいないので、$N_2 = N_3 = 0$ として考える。3 すくみの関係になるには、この状態から次に、種 1 に強い種 2 が増加し、種 1 は減少し、種 3 は増加しなければよい。

種 1 の個体数は、増加から減少に変わるので、(4.1) の第 1 式より、$r_1 - a_{11}N_1 \simeq 0$ のとき、$N_1 \simeq r_1/a_{11}$ となる。

種 2 の個体数は増加するので、(4.1) の第 2 式より $r_2 - a_{21}N_1 > 0$ であればよい。$N_1 \simeq r_1/a_{11}$ を代入すると、

$$a_{11}r_2 > a_{21}r_1 \tag{4.2}$$

を得る。

種 3 の個体数は増加しないので、(4.1) の第 3 式より $r_3 - a_{31}N_1 < 0$ であればよい。$N_1 \simeq r_1/a_{11}$ を代入すると、

$$a_{11}r_3 < a_{31}r_1 \tag{4.3}$$

を得る。

同様に次の条件が導ける。

$$a_{22}r_3 > a_{32}r_2 \tag{4.4}$$

$$a_{22}r_1 < a_{12}r_2 \tag{4.5}$$

$$a_{33}r_1 > a_{13}r_3 \tag{4.6}$$

$$a_{33}r_2 < a_{23}r_3 \tag{4.7}$$

また、時間経過により優占種の入れ替わりがおこるには、(4.1) の共存平衡点が不安定である必要がある。(4.1) の共存平衡点が不安定であるには、ヤコビ行列の固有値の実部が少なくとも 1 つ正であればよい。

したがって、3 すくみの関係になり、時間経過により優占種の入れ替わりがおこるには、(4.2)〜(4.7) の不等式を満たし、(4.1) の共存平衡点が不安定である 7 つの条件を満たす必要がある。しかし、これは必要条件であるが、十分条件ではない。例えば、上の 7 つの条件を全て満たすように $(a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}, a_{31}, a_{32}, a_{33}) = (0.2, 0.4, 0.1, 0.1, 0.2, 0.4, 0.4, 0.1, 0.2), c_i = 0.01, (r_1, r_2, r_3) = (0.5, 0.6, 0.7)$ とすると、ヤコビ行列は

$$J(N_1,N_2,N_3)=\begin{bmatrix} 0.5-0.4N_1-0.4N_2-0.1N_3 & -0.4N_1 & -0.1N_1 \\ -0.1N_2 & 0.6-0.1N_1-0.4N_2-0.4N_3 & -0.4N_2 \\ -0.4N_3 & -0.1N_3 & 0.7-0.4N_1-0.1N_2-0.4N_3 \end{bmatrix}$$

固有値は $-0.61027+0.i, 0.0216254+0.204939i, 0.0216254-0.204939i$ の 3 つであるから、共存平衡点は不安定という条件も満たす。(ここでは $c_i=0.01$ として計算している。)

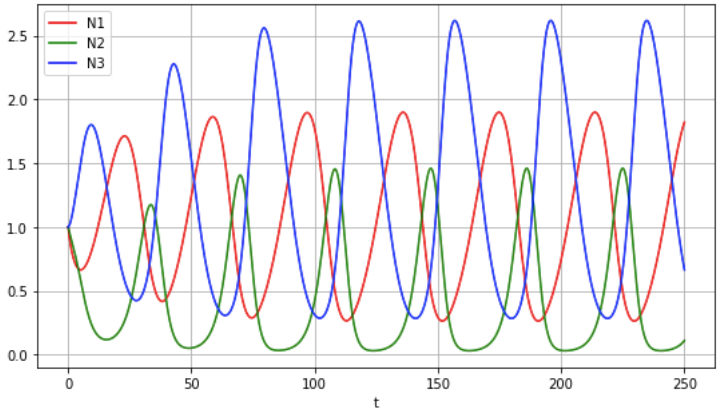


図 4.1: 3 すくみの関係のシミュレーション 1

この上の図 4.1 のようになり、優占種の入れ替わりがおこることもあるが、$(r_1,r_2,r_3)=(0.5,0.5,0.5)$ とし、他の値はそのままにすると、上の 7 つの条件を全て満たしているが、下の図 4.2 のようになり、優占種の入れ替わりは得られない。

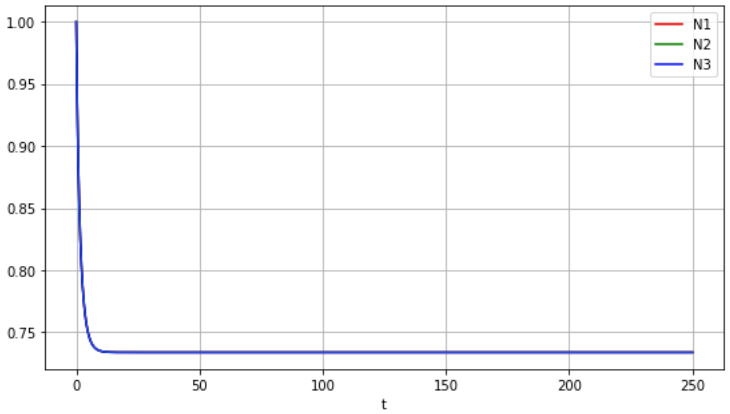


図 4.2: 3 すくみの関係のシミュレーション 2

上の図 4.2 は種 1 である N_1 の曲線と種 2 である N_2 の曲線が見えないが、種 3 である N_3 の曲線と同じ曲線を描いている。

また、$(a_{11},a_{12},a_{13},a_{21},a_{22},a_{23},a_{31},a_{32},a_{33})=(0.2,0.4,0.1,0.1,0.2,0.4,0.4,0.1,0.2), c_i=0, (r_1,r_2,r_3)=(0.5,0.6,0.7)$ のように $c_i=0$ とすると、下の図 4.3 のようになり、絶滅してしまう。

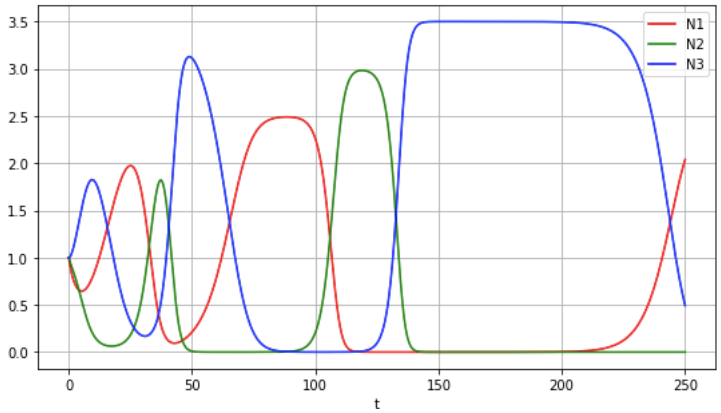


図 4.3: 3 すくみの関係のシミュレーション 3

さらに、$(a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}, a_{31}, a_{32}, a_{33}) = (0.3, 0.8, 0.1, 0.1, 0.2, 0.4, 0.6, 0.1, 0.2), c_i = 0.01, (r_1, r_2, r_3) = (0.5, 0.6, 0.7)$ のようにとすると、上の 7 つの条件を満たしているが、下の図 4.4 のようになり、種 2 が優占しておらず、魚種交替がおこっていない。

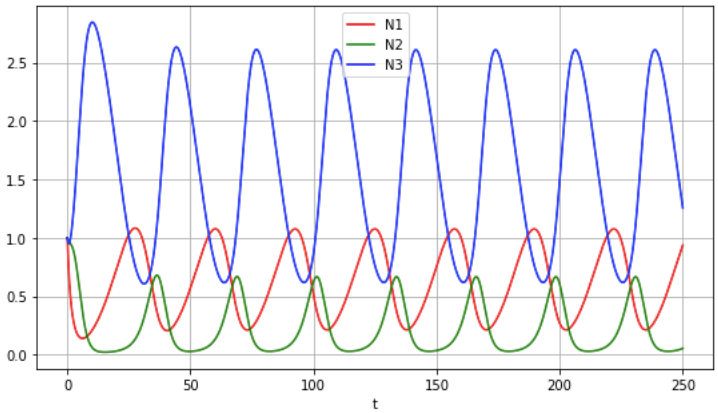


図 4.4: 3 すくみの関係のシミュレーション 3

上の 7 つの条件全てを満たしているのに、優占種の入れ替わりが得られない理由の 1 つとして、条件 (4.2)〜(4.7) の不等式を導いたとき、簡単のため $c_i = 0$ としたが、シミュレーションの際には $c_i = 0.01$ としていることが考えられる。また、実際には環境変化によって個体数変動に影響を受ける。

付 録A　Lotoka-Volterra方程式の解軌道のシミュレーションに用いたプログラム

c 言語の GLSC を用いて書いたプログラム。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <glsc.h>

// ウィンドウに表示する範囲
double xleft, xright, ybottom, ytop;

void draworbit(double, double, double, double);

void g_dump(char *fname, Display *display, Window wid)
{
  char command[256];
  sprintf(command, "import -window %lu %s", wid, fname);
  system(command);
}

double a=2.0, b=1.0, c=3.0, d=1.0;

int main(void)
{
  // 初期値
  double x0, y0;
  // 時間の刻み幅、追跡時間
  double h, tlimit, newh, newT;
  // メニューに対して入力されるコマンドの番号
  int cmd;
  // マウスのボタンの状態
  int but;
  //
  double win_width, win_height, w_margin, h_margin;
  //  ウィンドウ設定に使う変数
  Display *display;
  Window wid; // Window ID
  // ウィンドウに表示する範囲の設定
```

```
printf("範囲 (xleft,ybottom,xright,ytop)?\n");
scanf("%lf%lf%lf%lf", &xleft, &ybottom, &xright, &ytop);
// 時間刻み幅、追跡時間（とりあえず設定）
h = 0.01;
tlimit = 10.0;
// ウィンドウを開く
win_width = 200.0; win_height = 200.0; w_margin = 10.0; h_margin = 10.0;
g_init("GRAPH", win_width  + 2 * w_margin, win_height + 2 * h_margin);
g_device(G_BOTH);
g_def_scale(0, xleft, xright, ybottom, ytop,
            w_margin, h_margin, win_width, win_height);
g_sel_scale(0);
// x 軸、y 軸を描く
g_move(xleft, 0.0); g_plot(xright, 0.0);
g_move(0.0, ybottom); g_plot(0.0, ytop);
// メイン・ループの入口
do {
  // メニューを表示して、何をするか、番号で選択してもらう
  printf(" したいことを番号で選んで下さい。\n");
  printf("  -1:メニュー終了, 0:初期値のキーボード入力, 1:初期値のマウス入力\n");
  printf("   2: 刻み幅, 追跡時間変更 (現在 h=%7.4f, T=%7.4f)\n", h,tlimit);
  scanf("%d", &cmd);
  // 番号 cmd に応じて、指示された仕事をする
  if (cmd == 0) {
    // 初期値の入力
    printf(" 初期値 x0,y0=");
    scanf("%lf%lf", &x0, &y0);
    draworbit(x0,y0,h,tlimit);
  }
  else if (cmd == 1) {
    do {
      printf("マウスの左ボタンで初期値を指定して下さい（右ボタンで中止)。\n");
      g_mouse_sence(&x0, &y0, &but);
      printf("x0=%g, y0=%g, but=%d\n", x0, y0, but);
      if (but == 1) {
        printf("左ボタン, (x0,y0)=%f %f\n", x0,y0);
        draworbit(x0,y0,h,tlimit);
      }
      else if (but == 2) {
        printf("中ボタン, (x0,y0)=%f %f\n", x0,y0);
        draworbit(x0,y0,-h,tlimit);
      }
    }
    while (but != 3);
    printf("右ボタンがクリックされました。マウスによる初期値の入力を打ち切ります。\n");
```

```
    }
    else if (cmd == 2) {
      // 時間刻み幅、追跡時間の変更
      printf("時間刻み幅 h (%g), 追跡時間 T (%g): ", h, tlimit);
      scanf("%lf%lf", &newh, &newT);
      if (newh != 0 && newT != 0) {
        h = newh;
        tlimit = newT;
        printf("新しい時間刻み幅 h = %g, 新しい追跡時間 T = %g\n", h, tlimit);
      }
      else {
        printf("h=%g, T=%g は不適当です。\n", newh, newT);
      }
    }
  }
  while (cmd != -1);

  g_dump("reidai8a.png", g_get_display(), g_get_window());

  printf("GLSC ウィンドウを左ボタンでクリックして下さい\n");
  g_sleep(-1.0);
}

// 指示された初期値に対する解軌道を描く
void draworbit(double x0, double y0, double h, double tlimit)
{
  int in;
  double x, y, fx(double, double), fy(double, double);
  double k1x, k1y, k2x, k2y, k3x, k3y, k4x, k4y, t;
  // 時刻を 0 にセットする
  t = 0.0;
  // 初期値のセット
  x = x0;
  y = y0;
  // 初期点を描く
  if ((in = (xleft <= x && x <= xright && ybottom <= y && y <= ytop)) != 0)
    g_move(x,y);
  // ループの入口
  do {
    // Runge-Kutta 法による計算
    // k1 の計算
    k1x = h * fx(x, y);
    k1y = h * fy(x, y);
    // k2 の計算
    k2x = h * fx(x + k1x / 2.0, y + k1y / 2.0);
```

```
    k2y = h * fy(x + k1x / 2.0, y + k1y / 2.0);
    // k3 の計算
    k3x = h * fx(x + k2x / 2.0, y + k2y / 2.0);
    k3y = h * fy(x + k2x / 2.0, y + k2y / 2.0);
    // k4 の計算
    k4x = h * fx(x + k3x, y + k3y);
    k4y = h * fy(x + k3x, y + k3y);
    // (Xn+1,Yn+1) の計算
    x += (k1x + 2.0 * k2x + 2.0 * k3x + k4x) / 6.0;
    y += (k1y + 2.0 * k2y + 2.0 * k3y + k4y) / 6.0;
     // 解軌道を延ばす
    if (in) {
      if ((in = (xleft <= x && x <= xright && ybottom <= y && y <= ytop)) != 0)
        g_plot(x,y);
    }
    else {
      if ((in = (xleft <= x && x <= xright && ybottom <= y && y <= ytop)) != 0)
        g_move(x,y);
    }
    // 時刻を 1 ステップ分進める
    t += h;
  }
  while (fabs(t) <= fabs(tlimit)); // まだ範囲内かどうかチェック
}

// 微分方程式の右辺のベクトル値関数 f の x 成分
double fx(double x, double y)
{
  return a*x - b*x*y;
}

// 微分方程式の右辺のベクトル値関数 f の y 成分
double fy(double x, double y)
{
  return -c*y + d*x*y;
}
```

付 録B　Lotoka-Volterra方程式の解の時間変化のシミュレーションに用いたプログラム

c 言語の GLSC を用いて書いたプログラム。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <glsc.h>

// ウィンドウに表示する範囲
double xleft, xright, ybottom, ytop;

void draworbit(double, double, double, double);

void g_dump(char *fname, Display *display, Window wid)
{
  char command[256];
  sprintf(command, "import -window %lu %s", wid, fname);
  system(command);
}
//微分方程式の係数
double a=2.0, b=1.0, c=3.0, d=1.0;


int main(void)
{
  double x, y, fx(double, double), fy(double, double);
  double k1x, k1y, k2x, k2y, k3x, k3y, k4x, k4y, t;
  double h, tlimit;
  // 初期値
  double x0=1.0;
  double y0=1.0;
  //  ウィンドウ設定に使う変数
  double win_width, win_height, w_margin, h_margin;
  Display *display;
  Window wid; // Window ID
  // ウィンドウに表示する範囲の設定
  printf("範囲 (xleft,ybottom,xright,ytop)?\n");
  scanf("%lf%lf%lf%lf", &xleft, &ybottom, &xright, &ytop);
```

```
// 時間刻み幅、追跡時間（とりあえず設定）
h = 0.01;
tlimit = 10.0;
// ウィンドウを開く
win_width = 200.0; win_height = 200.0; w_margin = 10.0; h_margin = 10.0;
g_init("GRAPH", win_width  + 2 * w_margin, win_height + 2 * h_margin);
g_device(G_BOTH);
g_def_scale(0, xleft, xright, ybottom, ytop,
            w_margin, h_margin, win_width, win_height);
g_sel_scale(0);
// x 軸、y 軸を描く
g_move(xleft, 0.0); g_plot(xright, 0.0);
g_move(0.0, ybottom); g_plot(0.0, ytop);

// 時刻を 0 にセットする
t = 0.0;
// 初期値のセット
x = x0;
y = y0;
g_line_color(G_RED);
g_move(t, x);

do {
  // Runge-Kutta 法による計算
  // k1 の計算
  k1x = h * fx(x, y);
  k1y = h * fy(x, y);
  // k2 の計算
  k2x = h * fx(x + k1x / 2.0, y + k1y / 2.0);
  k2y = h * fy(x + k1x / 2.0, y + k1y / 2.0);
  // k3 の計算
  k3x = h * fx(x + k2x / 2.0, y + k2y / 2.0);
  k3y = h * fy(x + k2x / 2.0, y + k2y / 2.0);
  // k4 の計算
  k4x = h * fx(x + k3x, y + k3y);
  k4y = h * fy(x + k3x, y + k3y);
  // (Xn+1,Yn+1) の計算
  x += (k1x + 2.0 * k2x + 2.0 * k3x + k4x) / 6.0;
  y += (k1y + 2.0 * k2y + 2.0 * k3y + k4y) / 6.0;

  g_plot(t,x);

  // 時刻を 1 ステップ分進める
  t += h;
}
```

```
  while (fabs(t) <= fabs(tlimit));

  // 初期値に戻す
  t = 0.0;
  x = x0;
  y = y0;

  g_line_color(G_BLUE);
  g_move(t,y);

  do {
    // Runge-Kutta 法による計算
    // k1 の計算
    k1x = h * fx(x, y);
    k1y = h * fy(x, y);
    // k2 の計算
    k2x = h * fx(x + k1x / 2.0, y + k1y / 2.0);
    k2y = h * fy(x + k1x / 2.0, y + k1y / 2.0);
    // k3 の計算
    k3x = h * fx(x + k2x / 2.0, y + k2y / 2.0);
    k3y = h * fy(x + k2x / 2.0, y + k2y / 2.0);
    // k4 の計算
    k4x = h * fx(x + k3x, y + k3y);
    k4y = h * fy(x + k3x, y + k3y);
    // (Xn+1,Yn+1) の計算
    x += (k1x + 2.0 * k2x + 2.0 * k3x + k4x) / 6.0;
    y += (k1y + 2.0 * k2y + 2.0 * k3y + k4y) / 6.0;

    g_plot(t,y);
    // 時刻を 1 ステップ分進める
    t += h;

  }
  while (fabs(t) <= fabs(tlimit));


  //g_dump("reidai8a.png", g_get_display(), g_get_window());

  printf("GLSC ウィンドウを左ボタンでクリックして下さい\n");
  g_sleep(-1.0);
}//main fin


// 微分方程式の右辺のベクトル値関数 f の x 成分
double fx(double x, double y)
```

```
{
  return a*x - b*x*y;
}

// 微分方程式の右辺のベクトル値関数 f の y 成分
double fy(double x, double y)
{
  return -c*y + d*x*y;
}
```

付 録C　3すくみの関係のシミュレーションに用いたプログラム

Python 3.7.1 を用いて書いたプログラム。

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

#連立微分方程式を設定
def func(var, t, c1, c2, c3, r1, r2, r3,
    a11, a12, a13, a21, a22, a23, a31, a32, a33):
    dxdt = c1 + (r1 - a11*var[0] - a12*var[1] - a13*var[2])*var[0]
    dydt = c2 + (r2 - a21*var[0] - a22*var[1] - a23*var[2])*var[1]
    dzdt = c3 + (r3 - a31*var[0] - a32*var[1] - a33*var[2])*var[2]

    return [dxdt, dydt, dzdt]


def plot3d(t_lisst, var_list):
    # Figure を追加
    fig = plt.figure()

    # 3DAxes を追加
    ax = fig.gca(projection='3d')

    # 軸ラベルを設定
    ax.set_xlabel("$x$")
    ax.set_ylabel("$y$")
    ax.set_zlabel("$z$")

    #曲線を描画
    ax.plot(var_list[:, 0], var_list[:, 1], var_list[:, 2])

    plt.show()


if (__name__ == '__main__'):
    t_list = np.linspace(0.0, 250.0, 100000)
```

```
#数値を代入
c1 = 0.01
c2 = 0.01
c3 = 0.01
r1 = 0.5
r2 = 0.6
r3 = 0.7
a11 = 0.2
a12 = 0.4
a13 = 0.1
a21 = 0.1
a22 = 0.2
a23 = 0.4
a31 = 0.4
a32 = 0.1
a33 = 0.2

var_init = [1.0, 1.0, 1.0]
var_list = odeint(func, var_init, t_list, args=(c1, c2, c3, r1, r2, r3,
a11, a12, a13, a21, a22, a23, a31, a32, a33))
print(var_list)

plot3d(t_list, var_list)

plt.figure(figsize=(20,5))

plt.subplot(121)
plt.plot(t_list,var_list[:,0],'r', label='x')
plt.plot(t_list,var_list[:,1],'g', label='y')
plt.plot(t_list,var_list[:,2],'b', label='z')
plt.legend(loc='best')
plt.xlabel('t')
plt.grid()
```

参考文献

[1] James D. Murray, 『マレーの数理生物学入門』, 丸善出版（2017）

[2] M. ブラウン, 『微分方程式 上 その数学と応用』, 丸善出版（2019）

[3] M. ブラウン, 『微分方程式 下 その数学と応用』, 丸善出版（2021）

[4] 松田裕之, 『環境生態学序説』, 共立出版（2000）