数理生物学演習

第4回　ロトカ-ボルテラ モデル

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
- Python環境の導入

ロトカ-ボルテラ モデル（２種系）

被食-捕食系

$$\begin{cases} \text{被食者} & \dfrac{dx}{dt} = (a - by)x \\ \text{捕食者} & \dfrac{dy}{dt} = (cx - d)y \end{cases}$$

- 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

競争系

$$\begin{cases} \dfrac{dx}{dt} = (a - bx - cy)x \\ \dfrac{dy}{dt} = (d - ex - fy)y \end{cases}$$

- 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

共生系

$$\begin{cases} \dfrac{dx}{dt} = (a - bx + cy)x \\ \dfrac{dy}{dt} = (d + ex - fy)y \end{cases}$$

- 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

固有値による力学系の局所安定性解析

目的

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

の平衡点まわりの局所安定性を調べる

これを平衡点 $\boldsymbol{x}^*$のまわりで線形化して

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

と記述すれば，

n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．

ただし，$\mathbf{n}$，$\mathbf{M}$はそれぞれ

$$\mathbf{n} = \begin{pmatrix} n_1 \\ n_2 \\ \vdots \\ n_n \end{pmatrix} \quad \mathbf{M} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}^*) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}^*) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}^*) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}^*) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}^*) & \cdots & \frac{\partial f_2}{\partial x_n}(\mathbf{x}^*) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}^*) & \frac{\partial f_n}{\partial x_2}(\mathbf{x}^*) & \cdots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}^*) \end{pmatrix}$$

- 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

詳しくは，
数理生物学入門（巌佐）の
第２章を参照

被食-捕食系について平衡点の安定性を調べてみよう

実際にプログラムを組んでみよう！

配列

同じ型を持つ変数の集まり

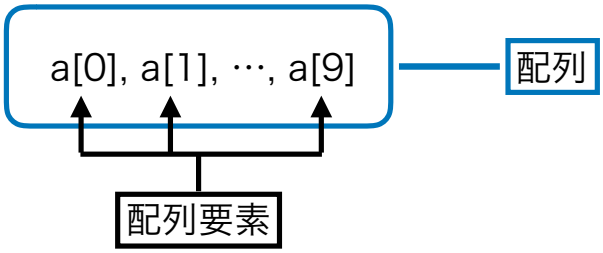
型　配列名[配列サイズ];
型　配列1[サイズ], 配列2[サイズ],…, 配列n[サイズ];

たくさんの変数を個別に宣言するのは面倒！

```
4-1. 配列
#include <stdio.h>

int main(void){
  int i;
  int a[10];

  for(i=0;i<10;i++){
    a[i]=i;
  }

  for(i=0;i<10;i++){
    printf("%d\n",a[i]);
  }

  return 0;
}
```

i番目の要素にiを代入

- 配列のなかのそれぞれの変数を配列要素という



- 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
int a[10];　で定義したならば，
a[0]～a[9]までの要素が存在する

4-2. i番目までの総和

```
#include <stdio.h>

int main(void){
  int i,j;
  int a[10],b[10];
    int temp;

  for(i=0;i<10;i++){
    a[i]=i;
  }

  for(i=0;i<10;i++){
    b[i]=0;
    for(j=0;j<10;j++){
      if(j<=i){
        temp=b[i]+a[j];
    b[i]=temp;
      }
    }
  }

  for(i=0;i<10;i++){
    printf("%d, %d\n",a[i],b[i]);
  }
  return 0;
}
```

初期化

$$b_i = \sum_{j=0}^{i} a_j$$

出力

```
0, 0
1, 1
2, 3
3, 6
 . . .
8, 36
9, 45
```

繰り返し処理 ループ その２ while

- 以前はforループを扱ったが，状況によってはより適切なループが存在する．
- そんなわけで今回はwhileループを紹介する

```
while (継続判定) {
  文1;
  文2;
  …
  文n;
  };
```

1. 継続判定が真ならば2へ．
偽ならばループ終了
2.文1～文nを実行．その後1へ

4-3. whileループ

```
#include <stdio.h>

int main(void){
  int i,j;

  for(i=0;i<10;i++){
    j=0;
    while (j<i){
      printf("True, ");
      j++;
    }
    printf("\n");
  }
  return 0;
}
```

forループのと一番の違いは
「継続判定を最初におこなう」点
→ 一度も処理を実行しないケースを書きやすい

型変換

暗黙の型変換　代入式により型が自動的に調整される

```
4-4. 代入式による型変換
#include <stdio.h>

int main(void){
  int i;
  double d;

  //int -> double
  i=5;
  d=i;
  printf("%d, %f\n",i,d);

  //double ->int
  d=3.6;
  i=d;
  printf("%d, %f\n",i,d);

  return 0;
}
```

double型になる

int型になる

それぞれ値や変数の型を変えて試してみよう

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

(型) 変数

変数の型を型として計算を実行する

```
4-5. キャスト
#include <stdio.h>

int main(void){
  int i1,i2;
  double d1,d2;

  i1=5;
  d1=6.0;
  i2=(int)d1%i1;
  d2=(double)i1/d1;

  printf("%d, %f\n",i2,d2);

  return 0;
}
```

d1をint型として使う
i1をdouble型として使う

被食-捕食系

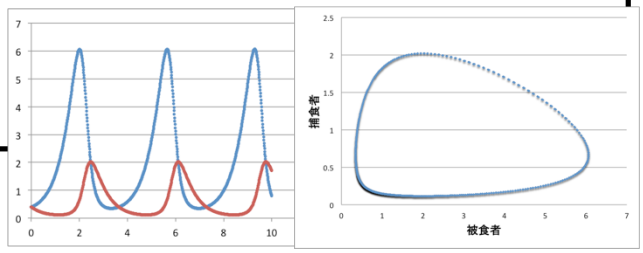
```
4-6. 被食-捕食系
#include <stdio.h>

int main(void){
  int i,j,k;
  double t,dt;
  double x[2];
  double xx[2];
  double a,b,c,d;

  FILE *fp;
  fp=fopen(“output_4-6.csv","w");

  dt=0.0001;
  a=2.0;
  b=3.0;
  c=1.0;
  d=2.0;

  t=0.0;
  x[0]=0.4;
  x[1]=0.4;
  fprintf(fp,"%f, %f, %f\n",t,x[0],x[1]);

  for(i=1;i<=100000;i++){
    t=dt*i;
    xx[0]=x[0]+dt*(a-b*x[1])*x[0];
    xx[1]=x[1]+dt*(c*x[0]-d)*x[1];
    x[0]=xx[0];
    x[1]=xx[1];

    if(i%100==0){
      fprintf(fp,"%f, %f,
%f\n",t,x[0],x[1]);
    }
  }

  fclose(fp);
  return 0;
}
```

被食者 $\frac{dx}{dt} = (a - by)x$

捕食者 $\frac{dy}{dt} = (cx - d)y$

パラメータ設定

初期値設定



競争系

$$\begin{cases} \frac{dx}{dt} = (a - bx - cy)x \\ \frac{dy}{dt} = (d - ex - fy)y \end{cases}$$

共生系

$$\begin{cases} \frac{dx}{dt} = (a - bx + cy)x \\ \frac{dy}{dt} = (d + ex - fy)y \end{cases}$$

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
4-7. 競争系，4-8. 共生系

Pythonの環境構築

https://koji.noshita.net/page/compbio/env/envpython/

Pythonを用いた可視化

https://koji.noshita.net/materials/compbio/example/04/plot_Lotka-Volterra.ipynb

1. リンク先のファイルを保存
2. 演習の作業用ディレクトリへドラッグ&ドロップ
3. Jupyter Notebookのダッシュボードから作業用ディレクトリへ移動し，plot_Lotka-Volterra.ipynbを開く
4. 記載に従って実行

本日の課題

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.

課題をPDFファイルにまとめて，
Google フォームにて提出すること

次回予告

第5回：突然変異固定までの待ち時間

5月14日

復習推奨

- 遺伝的浮動
- ライト-フィッシャー モデル