シミュレーション レポート

第6回～第10回

提出日　2020年7月15日 1～2コマ目

組番号　408
学籍番号　17406

氏名　金澤雄大

1　目的

2　実験環境

実験環境を表 1 に示す.gcc は Windows 上の WSL(Windows Subsystem for Linux) で動作するものを用いる.

表 1: 実験環境

CPU	AMD Ryzen 5 3600 6-Core Processor
メモリ	16.0GB DDR4
OS	Microsoft Windows 10 Home
gcc	(Ubuntu 9.3.0-17ubuntu1〜20.04) 9.3.0
Make	GNU Make 4.2.1

3　課題 6

本章では課題 6 における課題内容, プログラムの説明, 実行結果, 考察の 4 つについて述べる.

3.1　課題内容

式 (1) の方程式はロトカ・ヴォルテラの方程式と呼ばれる微分方程式である. ロトカ・ヴォルテラの方程式は 2 種類の生物の個体数の変化に関する数理モデルである. 式 (1) は被食者 (食べられる側) の個体数 y_1, 捕食者 (食べる側) の個体数 y_2 としたときの個体数の変化を表している. 本課題では式 (1) をオイラー法で実装し, 正の実数定数 a,b,c,d を変化させたときの解の特徴について考察する.

$$\begin{cases} \frac{dy_1}{dx} = ay_1 - cy_1y_2 \\ \frac{dy_2}{dx} = -by_1 + dy_1y_2 \end{cases} \tag{1}$$

3.2　プログラムの説明

本節では実装のための理論および次に示す 6 つの関数の機能, およびソースコードについて説明する.

1. multipleMatrix 関数
2. addVector 関数
3. scalerVector 関数
4. transformVector 関数
5. setVector 関数
6. main 関数

3.2.1 実装のための理論

本項では, はじめに 2 変数の単純な連立微分方程式について考え, 一般変数に理論を拡張する. はじめに 2 変数の単純な連立方程式について考える. 式 (2) の微分方程式をオイラー法で解く方法を考える. ただし,a に添え字がついているものはすべて定数である. 注目してほしいのは右辺で,y_1,y_2 の一次式になっていることがわかる.

$$\begin{cases} \frac{dy_1}{dx} = a_{11} + a_{12}y_1 + a_{13}y_2 \\ \frac{dy_2}{dx} = a_{21} + a_{22}y_1 + a_{23}y_2 \end{cases} \tag{2}$$

オイラー法を用いると式 (2) は時間 $t+1$ のとき式 (3) のように近似できる. ただし,h は刻み幅とする.

$$\begin{cases} y_1(t+1) = y_1(t) + h(a_{11} + a_{12}y_1(t) + a_{13}y_2(t)) \\ y_2(t+1) = y_2(t) + h(a_{21} + a_{22}y_1(t) + a_{23}y_2(t)) \end{cases} \tag{3}$$

式 (3) を実装すれば 2 変数の場合には問題なく実行できる. しかし, これを 3 変数,4 変数と拡張しようとすると C 言語では不都合が生じる. 例えば式 (2) の右辺を関数として実装した場合, 変数が増えるごとに関数の数が増えてしまい拡張性やメンテナンス性が悪い. このため式 (3) では一般変数に対応できない. そこで行列を用いる. 式 (3) を行列を用いて表すと式 (4) のようになる.

$$\begin{pmatrix} y_1(t+1) \\ y_2(t+1) \end{pmatrix} = \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} + h \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \begin{pmatrix} 1 \\ y_1(t) \\ y_2(t) \end{pmatrix} \tag{4}$$

式 (4) を参考にして, 一般に y_1,y_2...y_n の n 個の連立微分方程式 (式 5) があるとき, オイラー法による数値解は式 (6) の行列を用いることで計算できる. 本課題では式 (6) を実装する.

$$\begin{cases} \frac{dy_1}{dx} = a_{11} + a_{12}y_1 + a_{13}y_2 + \cdots + a_{1n+1}y_n \\ \frac{dy_2}{dx} = a_{21} + a_{22}y_1 + a_{23}y_2 + \cdots + a_{2n+1}y_n \\ \qquad\qquad \vdots \\ \frac{dy_n}{dx} = a_{n1} + a_{n2}y_1 + a_{n3}y_2 + \cdots + a_{nn+1}y_n \end{cases} \tag{5}$$

$$\begin{pmatrix} y_1(t+1) \\ y_2(t+1) \\ \vdots \\ y_n(t+1) \end{pmatrix} = \begin{pmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_n(t) \end{pmatrix} + h \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n+1} \\ a_{21} & a_{22} & \cdots & a_{2n+1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn+1} \end{pmatrix} \begin{pmatrix} 1 \\ y_1(t) \\ y_2(t) \\ \vdots \\ y_n(t) \end{pmatrix} \tag{6}$$

式 (1) のロトカ・ヴォルテラの方程式は y_1y_2 という相互作用項を含んでいるが, 式 (6) の行列によるオイラー法の数値計算はこれに対応していない. 式 (1) の方程式を行列で表すと式 (7) のようになる. これによって相互作用項を対応させることができる. 式 (7) から, 行列の積, ベクトルの変換, ベクトルのスカラー倍, ベクトルの和の 4 つの演算が必要であることがわかる. しかし, 式 (7) では $y_1(t+1)\cdot y_2(t+1) = y_1y_2(t+1)$ が成立していないため,$y_1(t)y_2(t)$ を $y_1(t)\cdot y_2(t)$ に置き換える必要がある.

$$\begin{pmatrix} y_1(t+1) \\ y_2(t+1) \\ y_1y_2(t+1) \end{pmatrix} = \begin{pmatrix} y_1(t) \\ y_2(t) \\ y_1y_2(t) \end{pmatrix} + h \begin{pmatrix} 0 & a & 0 & -c \\ 0 & 0 & -b & d \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ y_1(t) \\ y_2(t) \\ y_1y_2(t) \end{pmatrix} \tag{7}$$

3.2.2 multipleMatrix 関数

multipleMatrix 関数は行列の積を計算する関数である. 表 2 に multipleMatrix 関数の機能, 引数, 返り値の 3 つを示す. multipleMatrix 関数は $n \times (n+1)$ 行列と $(n+1) \times 1$ 行列の積を計算する機能を持つから, 引数として a,b,c の 3 つの行列をとる設計になっている. multipleMatrix 関数は引数 a,b の行列積 ab を c に代入する. ただし,DIM はオブジェクト形式マクロで定義されている微分方程式の変数の数である.

表 2: multipleMatrix 関数の機能, 引数, 返り値

機能	$n \times (n+1)$ 行列と $(n+1) \times 1$ 行列の行列積を計算する
引数	double a[DIM][DIM+1],double b[DIM+1][1],double c[DIM][1]
返り値	なし

リスト 1 に multipleMatrix 関数のソースコードを示す. リスト 1 では入力行列 a,b の積を for 文を用いて計算し, その結果を引数 c に代入する処理を行っている.

リスト 1: multipleMatrix 関数のコード

```
#define DIM 3

void multipleMatrix(double a[DIM][DIM+1],double b[DIM+1][1],double c[DIM][1]){
    int i,j,k;
    double tmp;
    tmp=0;
      for(i=0; i<DIM+1; i++){
        for(j=0; j<DIM; j++){
            c[i][j]=0;
            for(k=0; k<DIM+1; k++){
                c[i][j]+=a[i][k]*b[k][j];
      }
    }
  }
}
```

3.2.3 addVector 関数

3.2.4 scalerVector 関数

3.2.5 transformVector 関数

3.2.6 setVector 関数

3.2.7 main 関数

3.3 実行結果

3.4 考察

参考文献

[1] 国立高専機構長野高専,http://www.nagano-nct.ac.jp/ , 閲覧日 2020 年 8 月 5 日