シミュレーション

4 年電子情報工学科
34 番 横前洸佑

提出日：2019/1/23（木）
提出期限：2019/1/23（木）17:00

1　課題 6

課題 6 では、オイラー法を用いて生物の生存競争モデルの連立微分方程式、式 (1) を解くプログラムを作成する。ここではパラメータ a,b,c,d の値が全て 1、$y_1(x_0) = 10,\ y_2(x_0) = 10,\ dx = 0.1$ とする。

オイラー法

$$x_{i+1} = x_i + h$$
$$y_{i+1} = y_i + hf(x_i, y_i)$$

$$\begin{cases} \frac{dy_1}{dx} = ay_1 - cy_1y_2 \\ \frac{dy_2}{dx} = -by_2 + dy_1y_2 \end{cases} \tag{1}$$

1.1　作成したプログラム

今回作成したプログラムをソースコード 1 に示す。

ソースコード 1　課題 6 のプログラム

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

double diff_equa_1 (double, double);
double diff_equa_2 (double, double);
double euler_method (double x, double y_1, double y_2, double h, int step);

double a = 1.0;
double b = 1.0;
double c = 1.0;
double d = 1.0;

double new_x = 0.0;
double new_y_1 = 0.0;
double new_y_2 = 0.0;
int main (void){
    double x = 0.0;
    double y_1 = 10.0;
    double y_2 = 10.0;
    double h = 0.1;
    int step = 1;

    printf ("i, x, y_1, y_2\n");
    printf ("0, %f, %f, %f\n", x, y_1, y_2);
    euler_method(x, y_1, y_2, h, step);
}

//微分方程式
//式1
//dy_1 / dx = ay_1 - cy_1y_2
double diff_equa_1 (double y_1, double y_2){
    double result = a * y_1 - c * y_1 * y_2;
    return result;
}

//微分方程式
//式2
```

```
//dy_2 / dx = -by_2 + dy_1y_2
double diff_equa_2 (double y_1, double y_2){
    double result = -b * y_2 + d * y_1 * y_2;
    return result;
}

//オイラーの公式
double euler_method (double x, double y_1, double y_2, double h, int step){
    double old_x = x;
    double old_y_1 = y_1;
    double old_y_2 = y_2;

    for(int i = 0; i < step; i++){

        new_x = old_x + h;
        new_y_1 = old_y_1 + (h * diff_equa_1(old_y_1, old_y_2));
        new_y_2 = old_y_2 + (h * diff_equa_2(old_y_1, old_y_2));
        old_x = new_x;
        old_y_1 = new_y_1;
        old_y_2 = new_y_2;
        printf("%d, %f, %f, %f\n", i + 1, new_x, new_y_1, new_y_2);
    }

    return 0;
}
```

このプログラムはオイラー法で刻み幅 0.1 で 1 ステップ実行している。オイラー法の演算部及び微分方程式部は関数化して計算しやすいようにしてある。

1.2 プログラムの実行結果

実行結果を以下に示す。

```
i, x, y_1, y_2
0, 0.000000, 10.000000, 10.000000
1, 0.100000, 1.000000, 19.000000
```

1.3 考察

最初に 1 ステップ後の y_1, y_2 の値について考察する。x_1 の値は

$$x_1 = x_i + h$$

$$= 0 + 0.1$$

$$= 0.1$$

より 0.1 となる。

$y_1(x_1)$ の値は

$$
\begin{aligned}
y_1(x_1) &= y_0 + hf(x_0, y_0(x_1)) \\
&= 10 + 0.1 \times (-90) \\
&= 1
\end{aligned}
$$

よって 1 となる。$y_2(x_1)$ の値は

$$
\begin{aligned}
y_2(x_1) &= y_0 + hf(x_0, y_0(x_0)) \\
&= 10 + 0.1 \times 90 \\
&= 19
\end{aligned}
$$

よって 19 となる。この結果よりプログラムは正しく動作していると言える。

次にパラメータ a,b,c,d についていろいろな場合について計算し特徴を考察する。なお、ここではプログラムの計算ステップ数を 30 回とする。$y_2 > a/c$ の場合に変数 y_1, y_2 を縦軸, 時間 x を横軸としてグラフとすると図 1 のようになる。

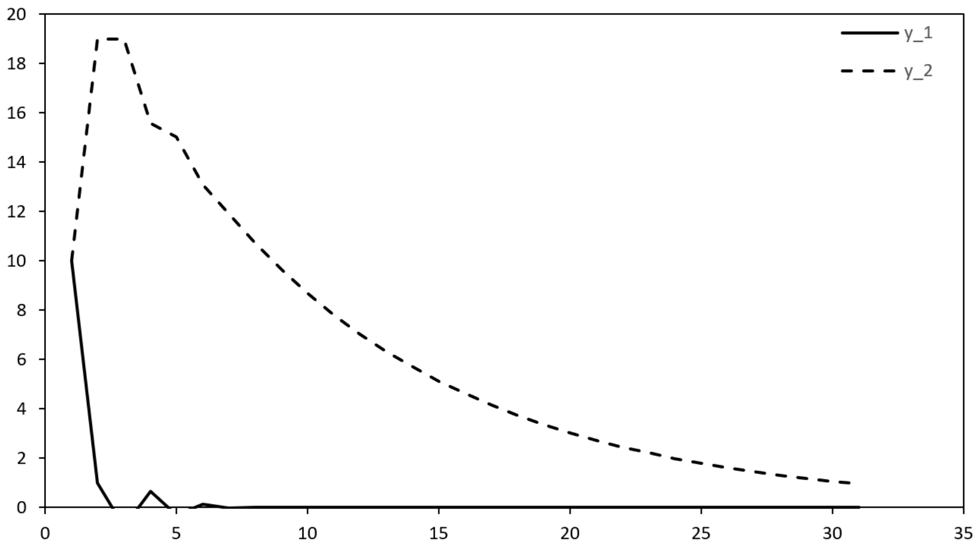

図 1　$y_2 > a/c$ の場合

$y_2 < a/c$ の場合は図 2 のようになる。

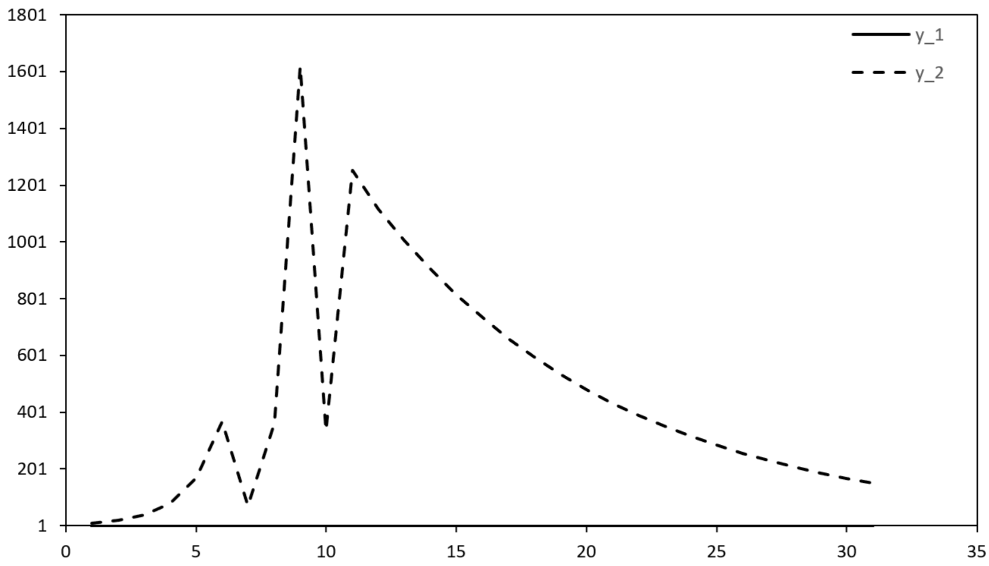

図 2　$y_2 < a/c$ の場合

$y_1 > b/d$ の場合は図 3 のようになる。

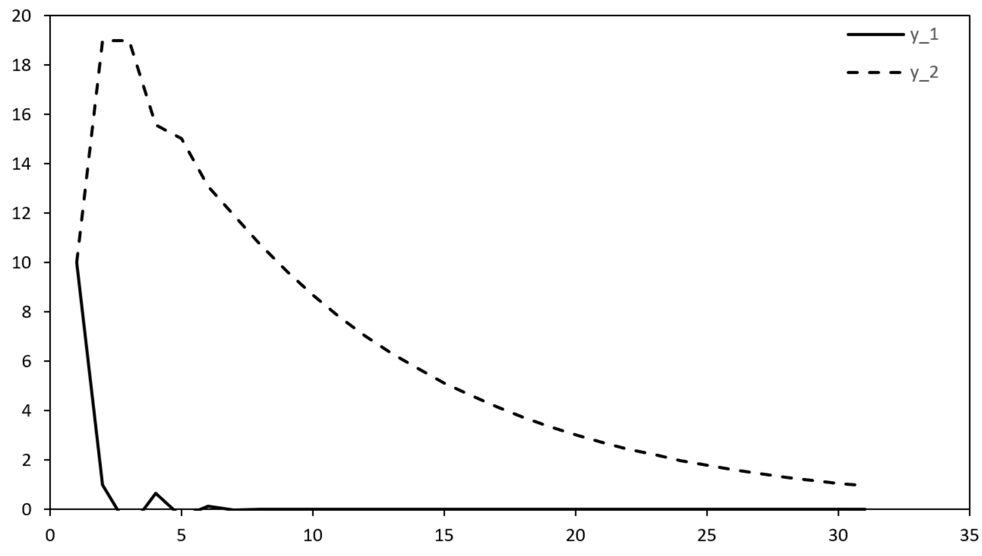

図 3　$y_1 > b/d$ の場合

$y_1 < b/d$ の場合は図 4 のようになる。

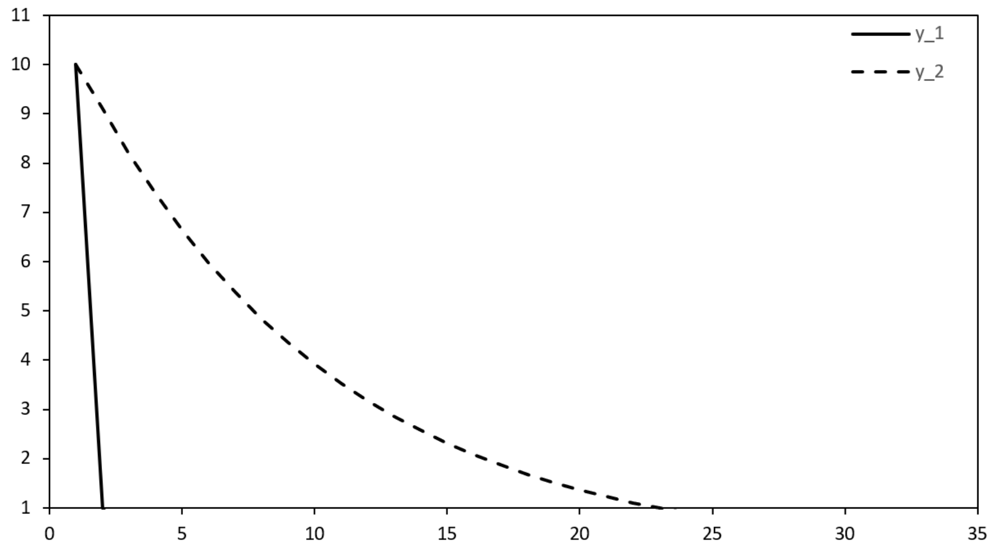

図 4　$y_1 < b/d$ の場合

2　課題 7

課題 7 では、式 (2) に示す高階微分方程式のニュートンの運動方程式をオイラー法で解くプログラムを作成する。そして、$l = 0$ の場合に単振動することを確認する。今回は、$t = 0, y = 0, y' = 0, m = 1$ とする。

$$y'' = \frac{-kx - ly'}{m} \tag{2}$$

ここで、式 (2) において、速度を表す従属変数 v を導入して、$y' = v$ とすれば

$$\begin{cases} v' = \frac{-kx-lv}{m} \\ y' = v \end{cases} \tag{3}$$

という連立 1 階微分方程式が得られる。この式 (3) は課題 6 と同じように解くことができる。

2.1　作成したプログラム

今回作成したプログラムをソースコード 2 に示す。

ソースコード 2　課題 7 のプログラム

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

double diff_equa_1 (double, double);
double diff_equa_2 (double);
double euler_method (double t, double y, double v, double h, int step);

```

```c
double new_t = 0.0;
double new_y = 0.0;
double new_v = 0.0;
double m = 1.0;
double k = 2.0;
double l = 0.0;


int main (void){
    double t = 0.0;
    double y = 10.0;
    double v = 0.0;
    double h = 0.005;
    int step = 2000;

    printf ("i, t, y, v\n");
    printf ("0, %f, %f, %f\n", t, y, v);
    euler_method(t, y, v, h, step);
}

//微分方程式
//式1
//v' = -k * y - l * v
double diff_equa_1 (double y, double v){
    double result = (-k * y) - (l * v);
    return result;
}

//微分方程式
//式2
//y' = v
double diff_equa_2 (double v){
    double result = v;
    return result;
}

//オイラーの公式
double euler_method (double t, double y, double v, double h, int step){
    double old_t = t;
    double old_y = y;
    double old_v = v;

    for(int i = 0; i < step; i++){

        new_t = old_t + h;
        new_v = old_v + (h * diff_equa_1(old_y, old_v));
        new_y = old_y + (h * diff_equa_2(old_v));
        old_t = new_t;
        old_y = new_y;
        old_v = new_v;
        printf("%d, %f, %f, %f\n", i + 1, new_t, new_y, new_v);
    }

    return 0;
}
```

このプログラムでは、刻み幅 h は 0.005 とし、計算ステップ数を 2000 回に設定した。

2.2 プログラムの実行結果

実行結果を以下に示す。なお、出力される実行結果が多いため一部のみを掲載する。

```
i, t, y, v
```

```
0, 0.000000, 10.000000, 0.000000
1, 0.005000, 10.000000, -0.100000
2, 0.010000, 9.999500, -0.200000
3, 0.015000, 9.998500, -0.299995
4, 0.020000, 9.997000, -0.399980
5, 0.025000, 9.995000, -0.499950
6, 0.030000, 9.992500, -0.599900
7, 0.035000, 9.989501, -0.699825
8, 0.040000, 9.986002, -0.799720
9, 0.045000, 9.982003, -0.899580
10, 0.050000, 9.977505, -0.999400
```

2.3 考察

出力された結果をグラフにすると図 5 のようになる。これより、単振動していることがわかる。

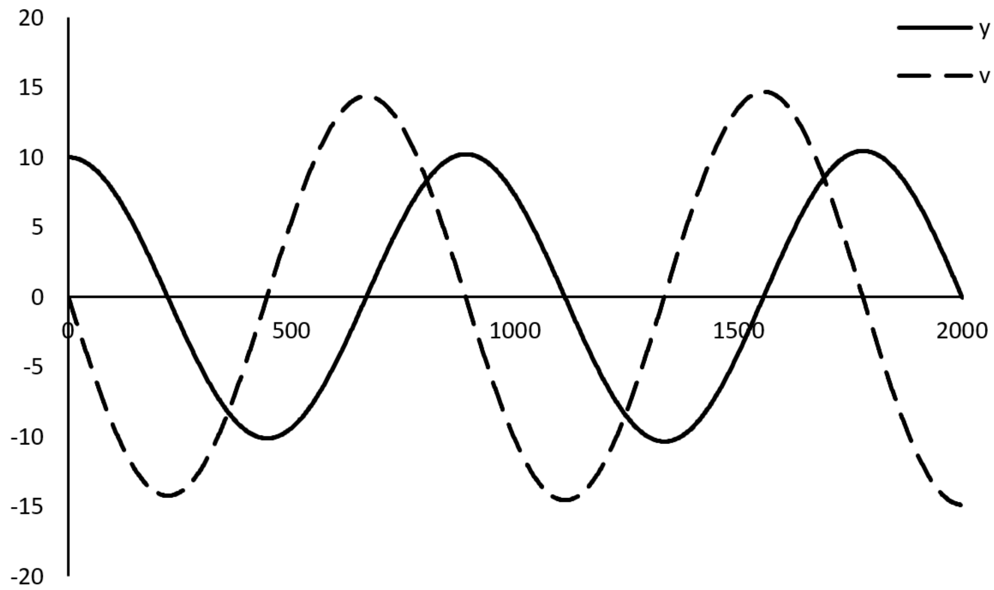

図 5　課題 7 の単振動の様子

また、k の値を 4 にした場合を図 6、l の値を 4 にした場合を図 7 にそれぞれ示す。

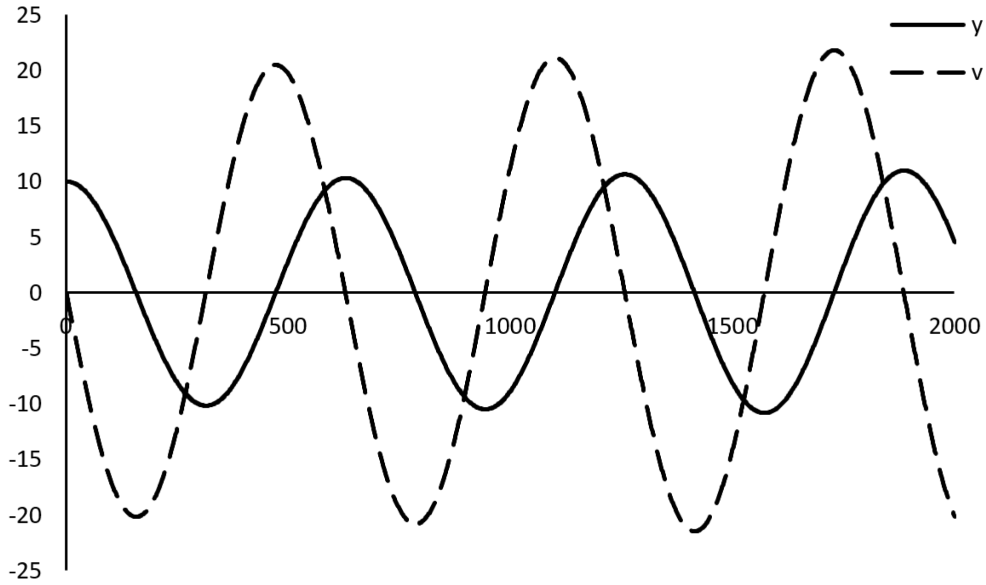

図 6　k の値が 4 の場合

これより k の値は単振動の周期を決める事がわかる。

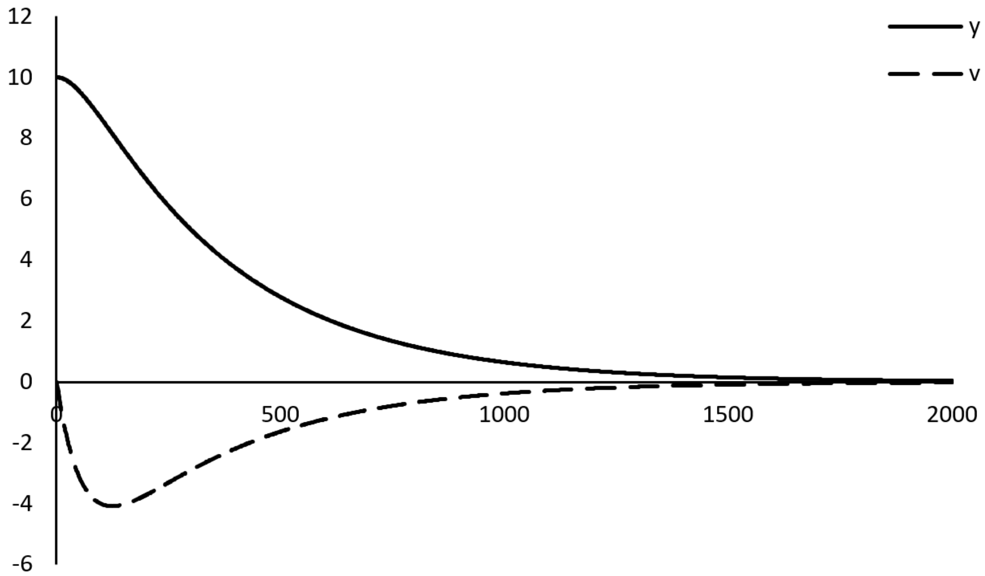

図 7　l の値が 4 の場合

l の値を大きくすると、単振動の振れ幅が小さくなることがわかる。

3　課題 8

RLC 共振回路の微分方程式

$$L\frac{d^2Q}{dt^2} + R\frac{dQ}{dt} + \frac{Q}{C} = 0 \tag{4}$$

の式 (4) をホイン法によって解くプログラムを作成する。そして、R=0 の場合に、単振動することを確認する。ここで、初期条件は $t = 0, Q = Q_0, dQ/dt = 0$ とする。また、パラメータは $R = 1[k\Omega], C = 0.3[nF], L = 10[mH], Q_0 = 10[pC]$ 程度の値を用いる。

ホイン法

$$\begin{aligned} x_{i+1} &= x_i + h \\ k_1 &= hf(x_i, y_i) \\ k_2 &= hf(x_i + h, y_i + k_1) \\ y_{i+1} &= y_i + \tfrac{1}{2}(k_1 + k_2) \end{aligned}$$

3.1　作成したプログラム

作成したプログラムをソースコード 3 に示す。

ソースコード 3　課題 8 のプログラム

```c
#include <stdio.h>
#include <math.h>

double diff_equa_1 (double, double);
double diff_equa_2 (double);
double heun_method (double t, double I, double Q,double dt, int step);

double R = 0.0;
double C = 0.3;
double L = 10.0;

int main (void){
    double t = 0.0;
    double Q = 10.0;
    double I = 0.0;

    double dt = 0.01;
    int step = 2000;

    printf ("i, t, I, Q\n");
    printf ("0, %f, %f, %f\n", t, I, Q);
    heun_method(t, I, Q, dt, step);
}


//微分方程式
//式1
// I’ = (-R * I - (Q / C)) / L
double diff_equa_1 (double Q, double I){
    double result = ((-R * I) -(Q / C)) / L ;
    return result;
}


```

```
//微分方程式
//式2
// Q' = I
double diff_equa_2 (double I){
    double result = I;
    return result;
}

//ホイン法
double heun_method (double t, double I, double Q, double dt, int step){
    double old_t = t;
    double old_Q = Q;
    double old_I = I;


    for(int i = 0; i < step; i++){
        double new_t = old_t + dt;
        double k1_eq1 = (dt * diff_equa_1(old_Q, old_I));
        double k2_eq1 = (dt * diff_equa_1(old_Q + dt, old_I + k1_eq1));
        double k1_eq2 = (dt * diff_equa_2(old_I));
        double k2_eq2 = (dt * diff_equa_2(old_I + k1_eq2));

        double new_I = old_I + (0.5 * (k1_eq1 + k2_eq1));
        double new_Q = old_Q + (0.5 * (k1_eq2 + k2_eq2));
        old_t = new_t;
        old_I = new_I;
        old_Q = new_Q;
        printf("%d, %f, %f, %f\n", i + 1, old_t, old_I, old_Q);
    }

    return 0;
}
```

ここで、ホイン法の刻み幅 h は 0.01 で、計算ステップ数を 2000 回とする。

3.2 プログラムの実行結果

実行結果を以下に示す。なお、出力される結果が多いため一部のみを掲載する。

```
i, t, I, Q
0, 0.000000, 0.000000, 10.000000
1, 0.010000, -0.033350, 10.000000
2, 0.020000, -0.066700, 9.999665
3, 0.030000, -0.100049, 9.998994
4, 0.040000, -0.133396, 9.997989
5, 0.050000, -0.166739, 9.996648
6, 0.060000, -0.200078, 9.994973
7, 0.070000, -0.233411, 9.992962
8, 0.080000, -0.266737, 9.990616
9, 0.090000, -0.300056, 9.987935
10, 0.100000, -0.333366, 9.984920
```

3.3　考察

出力された結果をグラフにすると図 8 のようになる。これより、単振動していることがわかる。

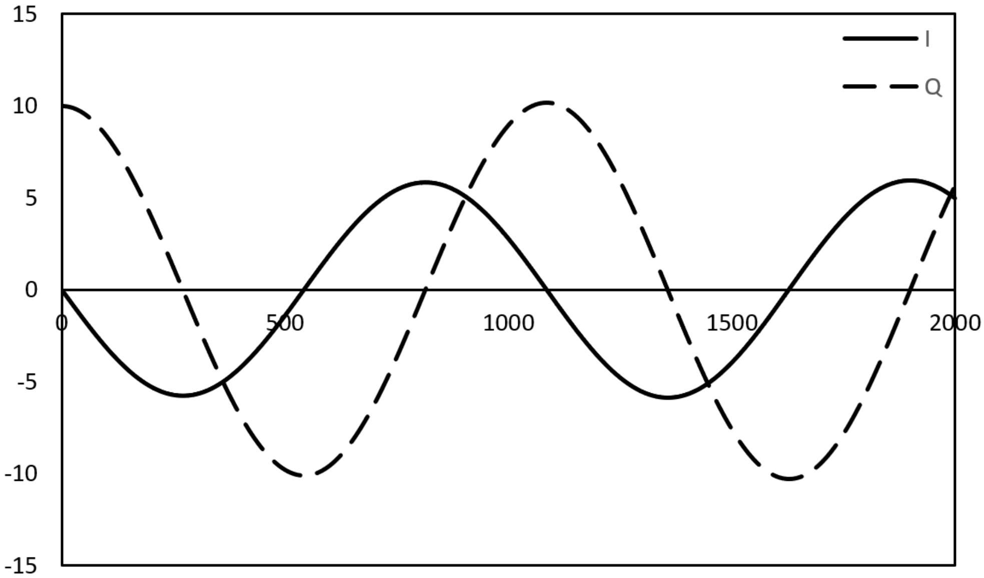

図 8　課題 8 の単振動の様子

次に R の値を $R = 2\sqrt{L/C}$ にしたときの様子を図 9 に示す。

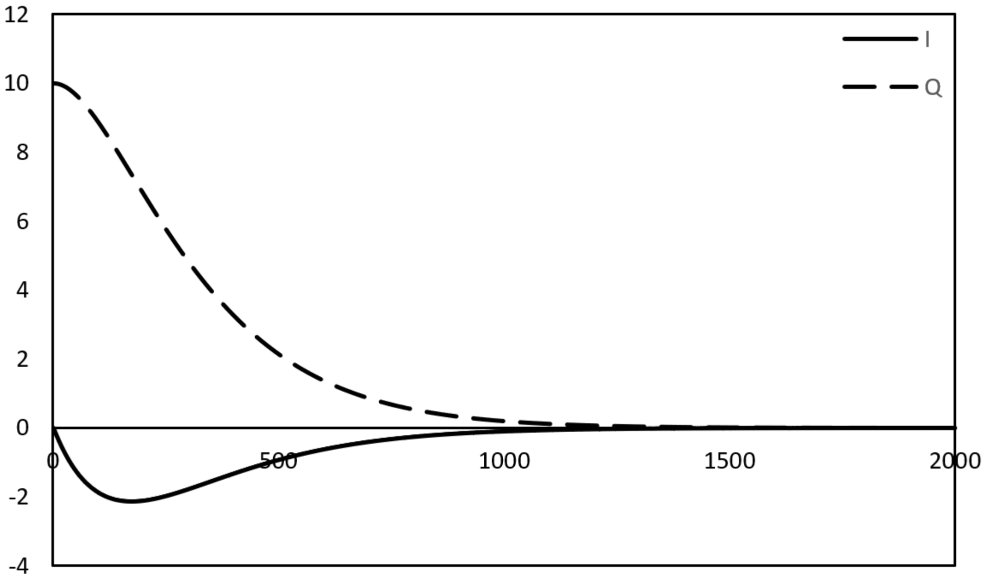

図 9　$R = 2\sqrt{L/C}$ の場合

これより、振動しなくなり、値が 0 に近づいていることがわかる。

4　課題 9

課題 9 では式 (5)(5) をホイン法で数値的に解くプログラムを作成する。また、エネルギー保存則の式 (7) が成り立つように刻み幅を設定する。なお、初期条件として、$t = 0, x = 0.1, y = 0.0, v_x = 0.0, v_y = 0.1$ とし、$q = m = 1.0$ とする。

$$\begin{cases} v'_x = \frac{q}{m} v_y B_0 \\ x' = v_x \end{cases} \tag{5}$$

$$\begin{cases} v'_y = -\frac{q}{m} v_x B_0 \\ y' = v_y \end{cases} \tag{6}$$

$$v_x^2 + v_y^2 = constant \tag{7}$$

4.1　作成したプログラム

作成したプログラムをソースコード 4 に示す。

ソースコード 4　課題 9 のプログラム

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

double diff_equa_1 (double);
double diff_equa_2 (double);
double diff_equa_3 (double);
double diff_equa_4 (double);
double heun_method (double, double, double, double, double, double, int, double);

const double q = 1.0;
const double m = 1.0;
double B = 10.0;


int main (void){
    double t = 0.0;
    double x = 0.1;
    double y = 0.0;
    double v_x = 0.0;
    double v_y = 0.1;
    double v_0 = pow(v_x, 2.0) + pow(v_y, 2.0);

    double dt = 0.005;
    int step = 500;

    printf ("i, t, x, y, v_x, v_y\n");
    printf ("0, %f, %f, %f, %f, %f\n", t, x, y, v_x, v_y);
    heun_method(t, x, y, v_x, v_y, dt, step, v_0);
}


//微分方程式
```

```
//式(2)
// v_y' = (q / m) * B * v_y
double diff_equa_1 (double v_y){
    double result = (q / m) * B * v_y;
    return result;
}


//微分方程式
//式(2)
// x' = v_x
double diff_equa_2 (double v_x){
    double result = v_x;
    return result;
}

//微分方程式
//式(3)
// v_y' = -(q / m) * B * v_x
double diff_equa_3 (double v_x){
    double result = -(q / m) * B * v_x;
    return result;
}

//微分方程式
//式(3)
//y' = v_y
double diff_equa_4 (double v_y){
    double result = v_y;
    return result;
}

//ホイン法
double heun_method (double t, double x, double y, double v_x,double v_y, double dt, int
     step, double v_0){
    double old_t = t;
    double old_x = x;
    double old_y = y;
    double old_v_y = v_y;
    double old_v_x = v_x;

    for(int i = 0; i < step; i++){
        double new_t = old_t + dt;
        double k1_eq1 = (dt * diff_equa_1(old_v_y));
        double k2_eq1 = (dt * diff_equa_1(old_v_y + k1_eq1));
        double k1_eq2 = (dt * diff_equa_2(old_v_y));
        double k2_eq2 = (dt * diff_equa_2(old_v_y + k1_eq2));

        double k1_eq3 = (dt * diff_equa_3(old_v_x));
        double k2_eq3 = (dt * diff_equa_3(old_v_x + k1_eq3));
        double k1_eq4 = (dt * diff_equa_4(old_v_y));
        double k2_eq4 = (dt * diff_equa_4(old_v_y + k1_eq4));

        double new_x = old_x + (0.5 * (k1_eq2 + k2_eq2));
        double new_y = old_y + (0.5 * (k1_eq4 + k2_eq4));
        double new_v_x = old_v_x + (0.5 * (k1_eq1 + k2_eq1));
        double new_v_y = old_v_y + (0.5 * (k1_eq3 + k2_eq3));

        old_t = new_t;
        old_x = new_x;
        old_y = new_y;
        old_v_x = new_v_x;
        old_v_y = new_v_y;

        printf("%d, %f, %f, %f, %f, %f\n", i + 1, old_t, old_x, old_y, old_v_x, old_v_y);

        double v_n = pow(new_v_x, 2.0) + pow(new_v_y, 2.0);
        printf("誤差:%f\n", (v_n - v_0) / v_0);
    }

    return 0;
}
```

ここでは、刻み幅を 0.005、計算ステップ数を 500 回としている。

4.2　プログラムの実行結果

実行結果を以下に示す。なお、出力される結果が多いため一部のみを掲載する。

```
i, t, x, y, v_x, v_y
0, 0.000000, 0.100000, 0.000000, 0.000000, 0.100000
1, 0.005000, 0.100501, 0.000501, 0.000501, 0.100000
誤差:0.000025
2, 0.010000, 0.101002, 0.001002, 0.001002, 0.099998
誤差:0.000051
3, 0.015000, 0.101504, 0.001504, 0.001504, 0.099993
誤差:0.000076
4, 0.020000, 0.102005, 0.002005, 0.002005, 0.099985
誤差:0.000102
5, 0.025000, 0.102506, 0.002506, 0.002506, 0.099975
誤差:0.000128
6, 0.030000, 0.103007, 0.003007, 0.003007, 0.099963
誤差:0.000155
7, 0.035000, 0.103508, 0.003508, 0.003508, 0.099948
誤差:0.000181
8, 0.040000, 0.104009, 0.004009, 0.004009, 0.099930
誤差:0.000208
9, 0.045000, 0.104510, 0.004510, 0.004510, 0.099910
誤差:0.000235
10, 0.050000, 0.105011, 0.005011, 0.005011, 0.099888
誤差:0.000263
```

結果より、誤差はほとんど発生していないことがわかる。

4.3　考察

ここでは、磁場の値 B を 1,5,10 と変化させて粒子の運動がどのように変化するか考察する。図 10,11,12 に変化させた時の様子を示す。

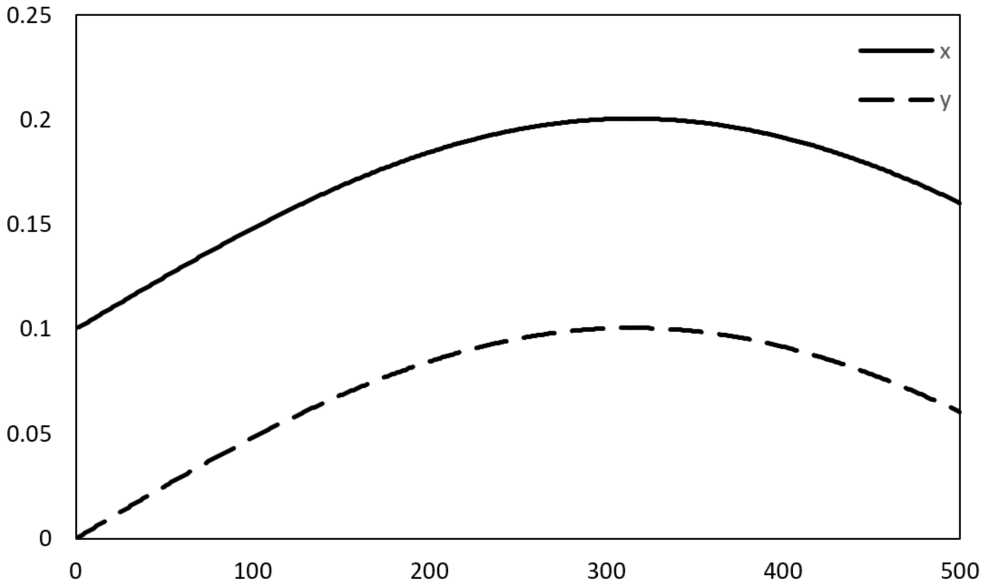

図 10　$B = 1$ の場合

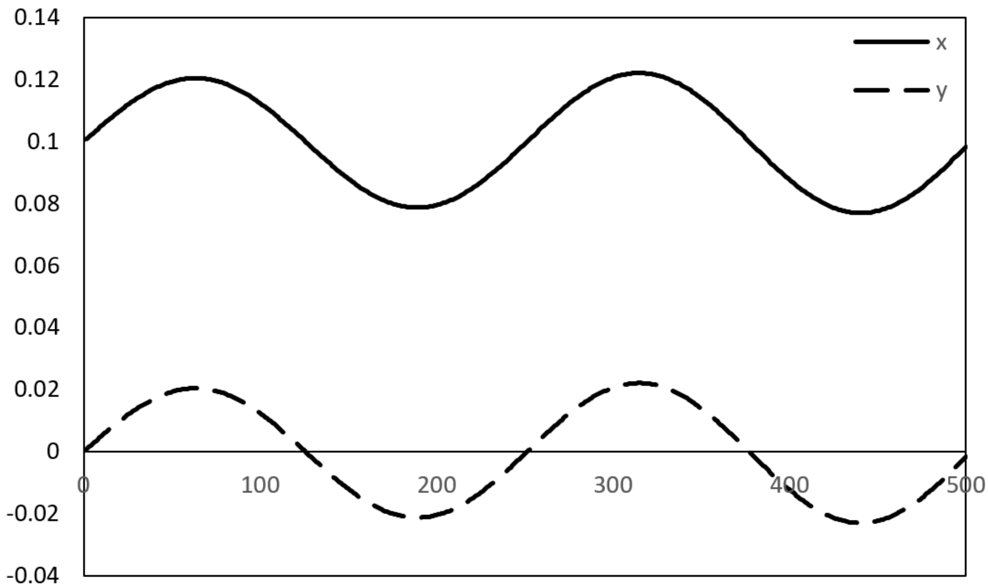

図 11　$B = 5$ の場合

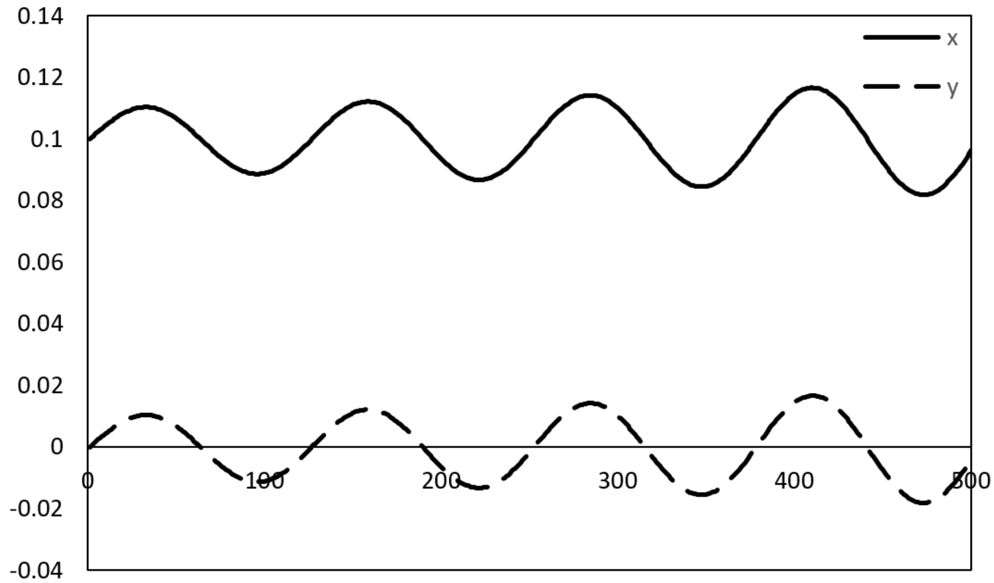

図 12　$B = 10$ の場合

B の値を大きくすると振動の周期が大きくなることがわかる。

5　課題 10

課題 10 では、課題 9 の式に式 (8) が示す、電場から受ける力を追加し、粒子の運動がどうなるかを報告する。ここでは、$B = E = 1.0$ とし、初期条件は、$t = 0.0$ のとき，$x = 0.1$，$y = 0.0$ ，$z = 0, 0, v_x = 0.0$，$v_y = 0.1, v_z = 0.0$ とする。また、$q = m = 1.0$ とする。

$$\begin{cases} v_z' = \frac{q}{m}z \\ z' = v_z \end{cases} \tag{8}$$

5.1　作成したプログラム

作成したプログラムをソースコード 5 に示す。

ソースコード 5　課題 10 のプログラム

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

double diff_equa_1 (double, double);
double diff_equa_2 (double);
double diff_equa_3 (double, double);
double diff_equa_4 (double);
double diff_equa_5 (double, double);
double diff_equa_6 (double);
double heun_method (double, double, double, double, double, double, double, double, int);

const double q = 1.0;
const double m = 1.0;
```

```c
const double B = 1.0;
const double E = 1.0;


int main (void){
    double t = 0.0;
    double x = 0.1;
    double y = 0.0;
    double z = 0.0;
    double v_x = 0.0;
    double v_y = 0.1;
    double v_z = 0.0;

    double dt = 0.05;
    int step = 4000;

    //printf ("i, t, x, y, v_x, v_y\n");
    //printf ("0 %f %f %f %f %f %f %f\n", t, x, y, z, v_x, v_y, v_z);
    printf("x, y, z\n");
    printf("%f %f %f\n", x, y, z);
    heun_method(t, x, y, z, v_x, v_y, v_z, dt, step);
}


//微分方程式
//式(x)
// v_y' = (q / m) * (x + v_y * B)
double diff_equa_1 (double x, double v_y){
    double result = (q / m) * (x + v_y * B);
    return result;
}


//微分方程式
//式(x)
// x' = v_x
double diff_equa_2 (double v_x){
    double result = v_x;
    return result;
}

//微分方程式
//式(y)
// v_y' = (q / m) * (y - v_x * B)
double diff_equa_3 (double y, double v_x){
    double result = (q / m) * (y - v_x * B);
    return result;
}

//微分方程式
//式(y)
//y' = v_y
double diff_equa_4 (double v_y){
    double result = v_y;
    return result;
}

//微分方程式
//式(z)
// v_z' = (q / m) * z
double diff_equa_5 (double z, double v_z){
    double result = (q / m) * z;
    return result;
}

//微分方程式
//式(z)
//z' = v_z
double diff_equa_6 (double v_z){
    double result = v_z;
    return result;
}

```

```
//ホイン法
double heun_method (double t, double x, double y, double z, double v_x,double v_y,double
    v_z, double dt, int step){
    double old_t = t;
    double old_x = x;
    double old_y = y;
    double old_z = z;
    double old_v_y = v_y;
    double old_v_x = v_x;
    double old_v_z = v_z;

    for(int i = 0; i < step; i++){
        double new_t = old_t + dt;
        double k1_eq1 = (dt * diff_equa_1(old_x, old_v_y));
        double k2_eq1 = (dt * diff_equa_1(old_x + dt, old_v_y + k1_eq1));
        double k1_eq2 = (dt * diff_equa_2(old_v_y));
        double k2_eq2 = (dt * diff_equa_2(old_v_y + k1_eq2));

        double k1_eq3 = (dt * diff_equa_3(old_y,old_v_x));
        double k2_eq3 = (dt * diff_equa_3(old_y + dt, old_v_x + k1_eq3));
        double k1_eq4 = (dt * diff_equa_4(old_v_y));
        double k2_eq4 = (dt * diff_equa_4(old_v_y + k1_eq4));

        double k1_eq5 = (dt * diff_equa_5(old_z, old_v_z));
        double k2_eq5 = (dt * diff_equa_5(old_z + dt, old_v_z + k1_eq5));
        double k1_eq6 = (dt * diff_equa_6(old_v_z));
        double k2_eq6 = (dt * diff_equa_6(old_v_z + k1_eq6));

        double new_x = old_x + (0.5 * (k1_eq2 + k2_eq2));
        double new_y = old_y + (0.5 * (k1_eq4 + k2_eq4));
        double new_z = old_z + (0.5 * (k1_eq6 + k2_eq6));
        double new_v_x = old_v_x + (0.5 * (k1_eq1 + k2_eq1));
        double new_v_y = old_v_y + (0.5 * (k1_eq3 + k2_eq3));
        double new_v_z = old_v_z + (0.5 * (k1_eq5 + k2_eq5));

        old_t = new_t;
        old_x = new_x;
        old_y = new_y;
        old_z = new_z;
        old_v_x = new_v_x;
        old_v_y = new_v_y;
        old_v_z = new_v_z;

        printf("%f %f %f\n", old_x, old_y, old_z);
        //printf("%d %f %f %f %f %f %f %f\n", i + 1, old_t, old_x, old_y, old_z, old_v_x,
            old_v_y, old_v_z);

    }

    return 0;
}
```

ここでは、刻み幅を 0.05 とし、計算ステップ数を 4000 回としている。

5.2 プログラムの実行結果

実行結果を以下に示す。なお、出力される結果が多いため一部のみを掲載する。

```
x, y, z
0.100000 0.000000 0.000000
```

```
0.105125 0.005125 0.000000
0.110314 0.010314 0.000064
0.115551 0.015551 0.000192
0.120820 0.020820 0.000385
0.126103 0.026103 0.000641
0.131382 0.031382 0.000963
0.136640 0.036640 0.001351
0.141855 0.041855 0.001805
0.147010 0.047010 0.002327
0.152082 0.052082 0.002917
```

5.3 考察

出力された結果を 3 次元プロットしたものを図 13 に示す。

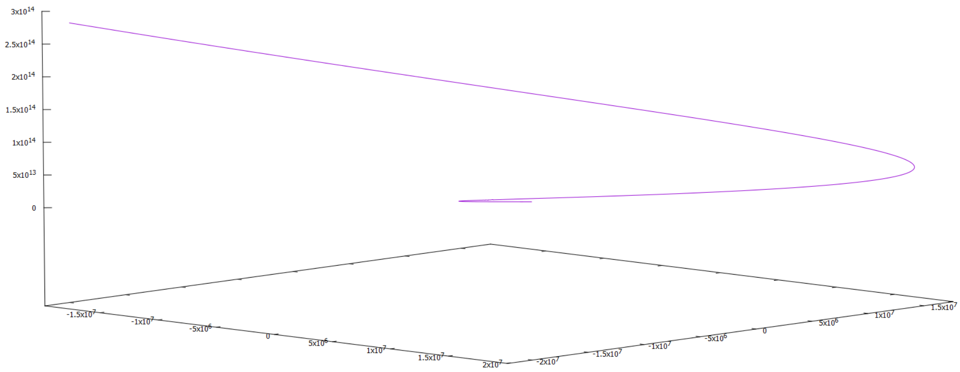

図 13　課題 10 の 3 次元プロット

これより、運動の振れ幅が大きく増加することがわかる。