計算物理演習レポート 2 数値計算

1 概要

2 体問題の数値計算を行った。数値計算の手法としては、Euler 法、Leapfrog 法、古典 4 次 Runge-Kutta 法を用いた。また、各値の計算には C++ 14、プロットには gnuplot を使用した。

2 Euler 法

2.1 内容

2 体問題の数値計算を Euler 法で行った。

2.2 ソースコード

euler/n.cpp

```
#include <string>
#include "../../util.h"

using namespace std;

int main() {
    const double dt = 0.1, G = 1.0, M = 1.0, m = 1.0;
    int n = 2000;
    double x[n], y[n], v_x[n], v_y[n];
    x[0] = 0.5, y[0] = 0.0, v_x[0] = 0.0, v_y[0] = 1.63;
    for (int i = 0; i < n - 1; i++) {
        x[i + 1] = x[i] + v_x[i] * dt;
        y[i + 1] = y[i] + v_y[i] * dt;
        double r = sqrt(pow(x[i], 2) + pow(y[i], 2));
        v_x[i + 1] = v_x[i] - G * M * m * x[i] / (pow(r, 3) * m) * dt;
        v_y[i + 1] = v_y[i] - G * M * m * y[i] / (pow(r, 3) * m) * dt;
    }
    string s;
    for (int i = 0; i < n; i++) {
        s += to_string(x[i]) + " " + to_string(y[i]) + "\n";
    }
    string fpath = "../out/euler/n.dat";
    write_to_file(s, fpath);

// gnuplot で
// ```set size ratio -1
// plot 'n.dat' with points pointtype 0```
// でプロット
    return 0;
}
```

util.cpp

```
//#include <random>
#include "Sample.h"

using namespace std;
```

util.h

```
#pragma once

#include <fstream>
#include <random>

using namespace std;

inline double get_random_0_to_1() {
  // 乱数生成器
  static mt19937_64 mt64(0);

  // [0.0, 1.0) の一様分布実数生成器
  uniform_real_distribution<double> get_rand_uni_real(0.0, 1.0);
  // 乱数を生成
  return get_rand_uni_real(mt64);
}

inline int write_to_file(const string& s, const string& fpath) {
  ofstream f;
  f.open(fpath);
  f << s;
  f.close();
```

```
    return 0;
}
```

2.3 結果

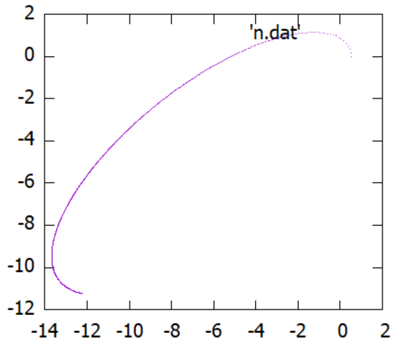

図 1　$dt = 0.1, \mathrm{n} = 1000$ とした場合

dt は 0.1 のまま、n をもう少し大きくしたとき（5000 秒先まで追ったとき）

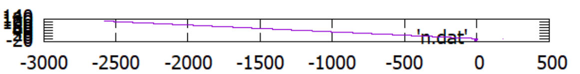

図 2　$dt = 0.1, \mathrm{n} = 50000$ とした場合

dt は 0.1 のまま、n を少しだけ大きくしたとき（200 秒先まで追ったとき）

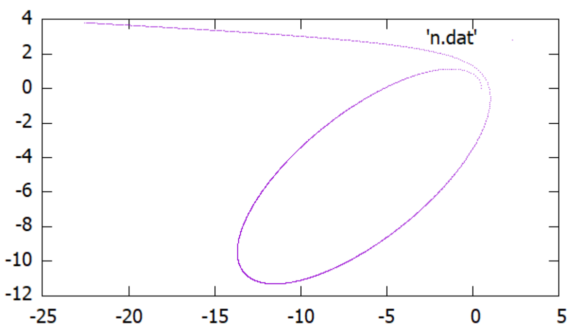

図 3　$dt = 0.1, \mathrm{n} = 2000$ とした場合

dt をもう少し小さくしたとき ($dt = 0.01$ で時刻は同じ 200 秒まで)

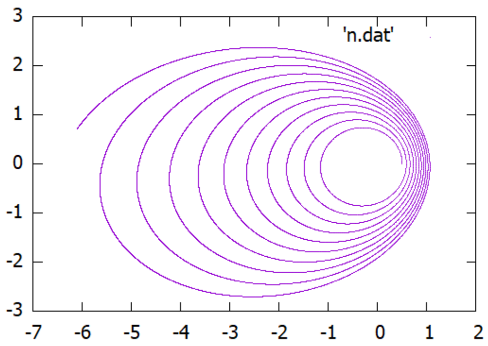

図 4　$dt = 0.01, \mathrm{n} = 20000$ とした場合

2.4 考察

徐々に半径が変わっているが、これは Euler 法がエネルギーを保存しないためと考えられる。また、dt を小さくすればより精度が高くなっているが、これは $\frac{\Delta x}{\Delta t}$ を小さくすればより $\frac{dx}{dt} = \lim_{\Delta t \to 0} \frac{\Delta x}{\Delta t}$ に近づくためと考えられる。

3 Leapfrog 法

3.1 内容

同じ 2 体問題の数値計算を Leapfrog 法でも行った。

3.2 ソースコード

leap-frog/n.cpp

```
#include <string>
#include "../../util.h"

using namespace std;

int main() {
    const int tmax = 100;
    const int n = 1000;
    const double dt = double(tmax) /
        double(n), G = 1.0, M = 1.0, m
        = 1.0;
    double x[2 * n], y[2 * n], v_x[2 *
        n], v_y[2 * n];
    x[0] = 0.5, y[0] = 0.0, v_x[0] =
        0.0, v_y[0] = 1.63;

    double r = sqrt(pow(x[0], 2) + pow(y
        [0], 2));
    v_x[1] = v_x[0] - G * M * m * x[0]
        / (pow(r, 3) * m) * dt * 0.5;
    v_y[1] = v_y[0] - G * M * m * y[0]
        / (pow(r, 3) * m) * dt * 0.5;
    for (long i = 0; i < n - 1; i++) {
        x[2 * (i + 1)] = x[2 * i] + v_x
```

```
            [2 * i + 1] * dt;
        y[2 * (i + 1)] = y[2 * i] + v_y
            [2 * i + 1] * dt;
        r = sqrt(pow(x[2 * (i + 1)], 2)
             + pow(y[2 * (i + 1)], 2));
        v_x[2 * i + 3] = v_x[2 * i + 1]
             - G * M * m * x[2 * (i +
            1)] / (pow(r, 3) * m) * dt;
        v_y[2 * i + 3] = v_y[2 * i + 1]
             - G * M * m * y[2 * (i +
            1)] / (pow(r, 3) * m) * dt;
    }
    string s;
    for (int i = 0; i < n; i++) {
        s += to_string(x[2 * i]) + " "
            + to_string(y[2 * i]) + "\n
            ";
    }
    string fpath = "../out/leap-frog/n.
        dat";
    write_to_file(s, fpath);

// gnuplot で
// ```set size ratio -1
// plot 'n.dat' with points pointtype
    0```
// でプロット
    return 0;
}
```

util.h、util.cpp は 2 と同一である。

3.3 結果

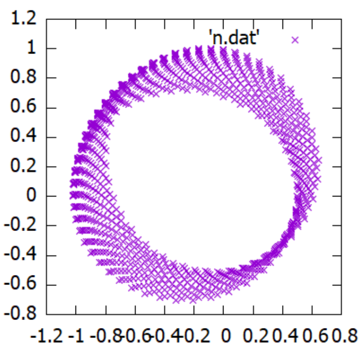

図 5　$dt = 0.1, \mathrm{n} = 1000, v_y = 1.63$ とした場合

dt は 0.1 のまま、n をもう少し大きくしたとき（5000 秒先まで追ったとき）

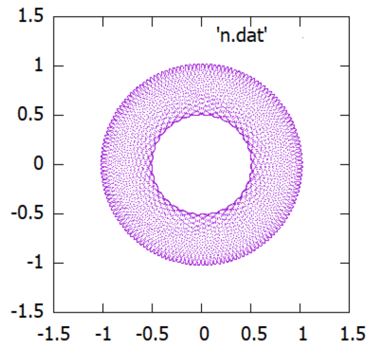

図 6　$\mathrm{d}t = 0.1, \mathrm{n} = 50000, v_y = 1.63$ とした場合

3.4 考察

Euler 法と違い Leapfrog 法はエネルギーを保存するので、円の半径はほぼ一定に保たれているように見える。

4 古典 4 次 Runge-Kutta 法

4.1 内容

テキストにはないが、同じ 2 体問題の数値計算を古典 4 次 Runge-Kutta 法 (RK4) でも行った。

4.2 計算方法

運動方程式

$$m\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -\frac{GMmx}{r^3} \tag{1}$$

$$m\frac{\mathrm{d}^2y}{\mathrm{d}t^2} = -\frac{GMmy}{r^3} \tag{2}$$

は次のような x、y、v_x、v_y についての連立常微分方程式で書ける。

$$\frac{\mathrm{d}x}{\mathrm{d}t} = v_x \tag{3}$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} = v_y \tag{4}$$

$$\frac{\mathrm{d}v_x}{\mathrm{d}t} = -\frac{GMmx}{mr^3} \tag{5}$$

$$\frac{\mathrm{d}v_y}{\mathrm{d}t} = -\frac{GMmy}{mr^3} \tag{6}$$

ここで、各方程式の右辺をそれぞれ

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f\left(t, x, y, v_x, v_y\right) \tag{7}$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} = g\left(t, x, y, v_x, v_y\right) \tag{8}$$

$$\frac{\mathrm{d}v_x}{\mathrm{d}t} = h\left(t, x, y, v_x, v_y\right) \tag{9}$$

$$\frac{\mathrm{d}v_y}{\mathrm{d}t} = i\left(t, x, y, v_x, v_y\right) \tag{10}$$

と置くと、t_{i+1}、x_{i+1}、y_{i+1}、$v_{x_{i+1}}$、$v_{y_{i+1}}$ は t_i、x_i、y_i、v_{x_i}、v_{y_i} を使って次のように求められる。

$$t_{i+1} = t_i + \mathrm{d}t \tag{11}$$

$$x_{i+1} = x_i + \frac{\mathrm{d}t}{6}\left(k_0 + 2k_1 + 2k_2 + k_3\right) \tag{12}$$

$$y_{i+1} = y_i + \frac{\mathrm{d}t}{6}\left(l_0 + 2l_1 + 2l_2 + l_3\right) \tag{13}$$

$$v_{x_{i+1}} = v_{x_i} + \frac{\mathrm{d}t}{6}\left(m_0 + 2m_1 + 2m_2 + m_3\right) \tag{14}$$

$$v_{y_{i+1}} = v_{y_i} + \frac{\mathrm{d}t}{6}\left(n_0 + 2n_1 + 2n_2 + n_3\right) \tag{15}$$

ただし、

$$k_0 = f\left(t_i, x_i, y_i, v_{x_i}, v_{y_i}\right) \tag{16}$$

$$l_0 = g\left(t_i, x_i, y_i, v_{x_i}, v_{y_i}\right) \tag{17}$$

$$m_0 = h\left(t_i, x_i, y_i, v_{x_i}, v_{y_i}\right) \tag{18}$$

$$n_0 = i\left(t_i, x_i, y_i, v_{x_i}, v_{y_i}\right) \tag{19}$$

$$k_1 = f\left(t_i + \frac{\mathrm{d}t}{2}, x_i + \frac{\mathrm{d}tk_0}{2}, y_i + \frac{\mathrm{d}tl_0}{2}, v_{x_i} + \frac{\mathrm{d}tm_0}{2}, v_{y_i} + \frac{\mathrm{d}tn_0}{2}\right) \tag{20}$$

$$l_1 = g\left(t_i + \frac{\mathrm{d}t}{2}, x_i + \frac{\mathrm{d}tk_0}{2}, y_i + \frac{\mathrm{d}tl_0}{2}, v_{x_i} + \frac{\mathrm{d}tm_0}{2}, v_{y_i} + \frac{\mathrm{d}tn_0}{2}\right) \tag{21}$$

$$m_1 = h\left(t_i + \frac{\mathrm{d}t}{2}, x_i + \frac{\mathrm{d}tk_0}{2}, y_i + \frac{\mathrm{d}tl_0}{2}, v_{x_i} + \frac{\mathrm{d}tm_0}{2}, v_{y_i} + \frac{\mathrm{d}tn_0}{2}\right) \tag{22}$$

$$n_1 = i\left(t_i + \frac{\mathrm{d}t}{2}, x_i + \frac{\mathrm{d}tk_0}{2}, y_i + \frac{\mathrm{d}tl_0}{2}, v_{x_i} + \frac{\mathrm{d}tm_0}{2}, v_{y_i} + \frac{\mathrm{d}tn_0}{2}\right) \tag{23}$$

$$k_2 = f\left(t_i + \frac{\mathrm{d}t}{2}, x_i + \frac{\mathrm{d}tk_1}{2}, y_i + \frac{\mathrm{d}tl_1}{2}, v_{x_i} + \frac{\mathrm{d}tm_1}{2}, v_{y_i} + \frac{\mathrm{d}tn_1}{2}\right) \tag{24}$$

$$l_2 = g\left(t_i + \frac{\mathrm{d}t}{2}, x_i + \frac{\mathrm{d}tk_1}{2}, y_i + \frac{\mathrm{d}tl_1}{2}, v_{x_i} + \frac{\mathrm{d}tm_1}{2}, v_{y_i} + \frac{\mathrm{d}tn_1}{2}\right) \tag{25}$$

$$m_2 = h\left(t_i + \frac{\mathrm{d}t}{2}, x_i + \frac{\mathrm{d}tk_1}{2}, y_i + \frac{\mathrm{d}tl_1}{2}, v_{x_i} + \frac{\mathrm{d}tm_1}{2}, v_{y_i} + \frac{\mathrm{d}tn_1}{2}\right) \tag{26}$$

$$n_2 = i\left(t_i + \frac{\mathrm{d}t}{2}, x_i + \frac{\mathrm{d}tk_1}{2}, y_i + \frac{\mathrm{d}tl_1}{2}, v_{x_i} + \frac{\mathrm{d}tm_1}{2}, v_{y_i} + \frac{\mathrm{d}tn_1}{2}\right) \tag{27}$$

$$k_3 = f\left(t_i + \mathrm{d}t, x_i + \mathrm{d}tk_2, y_i + \mathrm{d}tl_2, v_{x_i} + \mathrm{d}tm_2, v_{y_i} + \mathrm{d}tn_2\right) \tag{28}$$

$$l_3 = g\left(t_i + \mathrm{d}t, x_i + \mathrm{d}tk_2, y_i + \mathrm{d}tl_2, v_{x_i} + \mathrm{d}tm_2, v_{y_i} + \mathrm{d}tn_2\right) \tag{29}$$

$$m_3 = h\left(t_i + \mathrm{d}t, x_i + \mathrm{d}tk_2, y_i + \mathrm{d}tl_2, v_{x_i} + \mathrm{d}tm_2, v_{y_i} + \mathrm{d}tn_2\right) \tag{30}$$

$$n_3 = i\left(t_i + \mathrm{d}t, x_i + \mathrm{d}tk_2, y_i + \mathrm{d}tl_2, v_{x_i} + \mathrm{d}tm_2, v_{y_i} + \mathrm{d}tn_2\right) \tag{31}$$

である。

4.3 ソースコード

runge-kutta/n.cpp

```
#include <string>
#include "../../util.h"

using namespace std;

#define tmax 100
#define n 1000
#define dt double(tmax) / double(n)
#define G 1.0
#define M 1.0
#define m 1.0

double f_dx_dt(double t, double _x, double _y, double _v_x, double _v_y) {
    return _v_x;
}

double g_dy_dt(double t, double _x, double _y, double _v_x, double _v_y) {
    return _v_y;
}

double h_dv_x_dt(double t, double _x, double _y, double _v_x, double _v_y) {
    double r = sqrt(pow(_x, 2) + pow(_y, 2));
    return -G * M * m * _x / (pow(r, 3) * m);
}

double i_dv_y_dt(double t, double _x, double _y, double _v_x, double _v_y) {
    double r = sqrt(pow(_x, 2) + pow(_y, 2));
    return -G * M * m * _y / (pow(r, 3) * m);
}

int main() {
    double t[n], x[n], y[n], v_x[n], v_y[n];
    t[0] = 0.0, x[0] = 0.5, y[0] = 0.0, v_x[0] = 0.0, v_y[0] = 1.63;

    for (int i = 0; i < n - 1; i++) {
        double k0 = f_dx_dt(t[i], x[i], y[i], v_x[i], v_y[i]);
        double l0 = g_dy_dt(t[i], x[i], y[i], v_x[i], v_y[i]);
        double m0 = h_dv_x_dt(t[i], x[i], y[i], v_x[i], v_y[i]);
        double n0 = i_dv_y_dt(t[i], x[i], y[i], v_x[i], v_y[i]);
        double k1 = f_dx_dt(t[i] + dt / 2.0, x[i] + dt / 2.0 * k0, y[i] + dt / 2.0 * l0, v_x[i] + dt / 2.0 * m0, v_y[i] + dt / 2.0 * n0);
```

```
            double l1 = g_dy_dt(t[i] + dt / 2.0, x[i] + dt / 2.0 * k0, y[i] + dt / 2.0 * l0, v_x[i] + dt / 2.0 * m0, v_y[i] + dt / 2.0 * n0);
            double m1 = h_dv_x_dt(t[i] + dt / 2.0, x[i] + dt / 2.0 * k0, y[i] + dt / 2.0 * l0, v_x[i] + dt / 2.0 * m0, v_y[i] + dt / 2.0 * n0);
            double n1 = i_dv_y_dt(t[i] + dt / 2.0, x[i] + dt / 2.0 * k0, y[i] + dt / 2.0 * l0, v_x[i] + dt / 2.0 * m0, v_y[i] + dt / 2.0 * n0);
            double k2 = f_dx_dt(t[i] + dt / 2.0, x[i] + dt / 2.0 * k1, y[i] + dt / 2.0 * l1, v_x[i] + dt / 2.0 * m1, v_y[i] + dt / 2.0 * n1);
            double l2 = g_dy_dt(t[i] + dt / 2.0, x[i] + dt / 2.0 * k1, y[i] + dt / 2.0 * l1, v_x[i] + dt / 2.0 * m1, v_y[i] + dt / 2.0 * n1);
            double m2 = h_dv_x_dt(t[i] + dt / 2.0, x[i] + dt / 2.0 * k1, y[i] + dt / 2.0 * l1, v_x[i] + dt / 2.0 * m1, v_y[i] + dt / 2.0 * n1);
            double n2 = i_dv_y_dt(t[i] + dt / 2.0, x[i] + dt / 2.0 * k1, y[i] + dt / 2.0 * l1, v_x[i] + dt / 2.0 * m1, v_y[i] + dt / 2.0 * n1);
            double k3 = f_dx_dt(t[i] + dt, x[i] + dt * k2, y[i] + dt * l2, v_x[i] + dt * m2, v_y[i] + dt * n2);
            double l3 = g_dy_dt(t[i] + dt, x[i] + dt * k2, y[i] + dt * l2, v_x[i] + dt * m2, v_y[i] + dt * n2);
            double m3 = h_dv_x_dt(t[i] + dt, x[i] + dt * k2, y[i] + dt * l2, v_x[i] + dt * m2, v_y[i] + dt * n2);
            double n3 = i_dv_y_dt(t[i] + dt, x[i] + dt * k2, y[i] + dt * l2, v_x[i] + dt * m2, v_y[i] + dt * n2);
            t[i + 1] = t[i] + dt;
            x[i + 1] = x[i] + (dt / 6.0) * (k0 + 2.0 * k1 + 2.0 * k2 + k3);
            y[i + 1] = y[i] + (dt / 6.0) * (l0 + 2.0 * l1 + 2.0 * l2 + l3);
            v_x[i + 1] = v_x[i] + (dt / 6.0) * (m0 + 2.0 * m1 + 2.0 * m2 + m3);
            v_y[i + 1] = v_y[i] + (dt / 6.0) * (n0 + 2.0 * n1 + 2.0 * n2 + n3);
        }
        string s;
        for (int i = 0; i < n; i++) {
            s += to_string(x[i]) + " " + to_string(y[i]) + "\n";
        }
        string fpath = "../out/runge-kutta/n.dat";
        write_to_file(s, fpath);

    // gnuplot で
    // ‘‘‘set size ratio -1
    // plot 'n.dat' with points pointtype 0‘‘‘
    // でプロット
        return 0;
    }
```

util.h、util.cpp は 2 と同一である。

4.4 結果

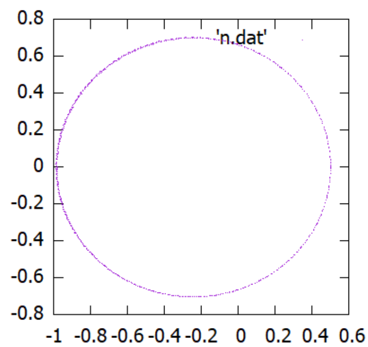

図 7　$dt = 0.1, n = 1000, v_y = 1.63$ とした場合

dt は 0.1 のまま、n をもう少し大きくしたとき（5000 秒先まで追ったとき）

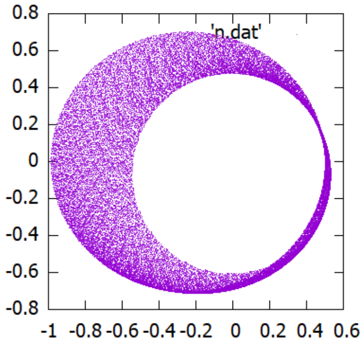

図 8　dt = 0.1, n = 50000, v_y = 1.63 とした場合

4.5　考察

古典 4 次 Runkge-Kutta 法は 4 次精度なので、Euler 法や Leapfrog 法に比べ同じ dt = 0.1 でもかなり精度がよくなっており、軌道がほとんどずれていない。

ただし、5000 秒先まで追うとどんどん軌道半径が小さくなってしまった。これは、ただ軌道半径がほぼ一定に保たれて位置がずれるだけの Leapfrog 法とは違って、古典 4 次 Runkge-Kutta 法がエネルギーを保存しないためと思われる。

参考文献

[1] ルンゲクッタ法による常微分方程式の数値解法　埼玉工業大学 https://www.sit.ac.jp/user/konishi/JPN/L_Support/SupportPDF/Runge-KuttaMethod.pdf
[2] ルンゲ = クッタ法 - Wikipedia https://ja.wikipedia.org/wiki/ルンゲ = クッタ法
[3] N 体シミュレーションの基礎 道越秀吾 国立天文台 CfCA https://www.cfca.nao.ac.jp/ cfca/hpc/mu-v/text/michikoshi_12.pdf
[4] suti-sekibun-bibun.pdf https://www.sci.kagoshima-u.ac.jp/fujii/data_kougi/suti-sekibun-bibun.pdf
[5] 古典 4 次 Runge-Kutta 法の精度確認 - Qiita https://qiita.com/kaityo256/items/e3428deb394b3ad1e739