シミュレーション
—定積分と常微分方程式—

学籍番号：16426
4 年 電子情報工学科 23 番
福澤 大地

提出日：2019 年 12 月 12 日

1　目的

コンピュータを用いて、数学的な問題の近似解を求める手法を学習する。

今回は、定積分と常微分方程式について数値積分を行う。いくつかの手法を実装し、それぞれを比較することで、各手法の特徴や精度を確認する。

2　開発環境

プログラムの開発、実行を行った環境を表 1 に示す。

表 1　開発環境

CPU	Intel Core i5–7400 @ 3.0GHz
メモリ	8GB
OS	Microsoft Windows 10 Home
システム	64bit
コンパイラ	GCC 7.4.0

3　課題 1：台形公式

3.1　課題内容

式 1 について、台形公式を用いて数値積分を行う。

$$\int_0^{\frac{\pi}{6}} \frac{dx}{\cos x} \tag{1}$$

3.2　プログラムリスト

課題 1 のプログラムリストをリスト 1 に示す。

分割数を n, 分割幅を h とすると、台形公式は式 2 のように表せる。この計算を `trapezoid` 関数内で行うことで、定積分の近似解を導出している。

$$\int_a^b f(x)dx \simeq \frac{h}{2}\left[f(a) + 2\sum_{j=1}^{n-1} f(jh) + f(b)\right] \tag{2}$$

リスト 1　課題 1 のプログラム

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

#define a 0.0
#define b (M_PI / 6)
#define SOLUTION (log(3) / 2)

double f(double);
double trapezoid(int);
```

```c

int main(void) {
    int i;

    for (i = 1; i <= 32; i *= 2) {
        double S = trapezoid(i);
        printf("n = %2d, S = %.16f, e = %.16f\n", i, S, S - SOLUTION);
    }

    return 0;
}

// f(x)
double f(double x) {
    return 1 / cos(x);
}

// 台形公式
double trapezoid(int n) {
    int i;
    double x;
    double h = (b - a) / n;
    double S = f(a) + f(b);

    for (i = 1; i <= n - 1; i++) {
        x = a + i * h;
        S += f(x) * 2;
    }

    S = S * h / 2;
    return S;
}
```

3.3 実行結果

課題 1 の実行結果をリスト 2 に示す。

リスト 2　課題 1 の実行結果

```
n =  1, S = 3.0000000000000000, e = -0.1415926535897931
n =  2, S = 3.1000000000000001, e = -0.0415926535897930
n =  4, S = 3.1311764705882354, e = -0.0104161830015577
n =  8, S = 3.1389884944910889, e = -0.0026041590987043
n = 16, S = 3.1409416120413889, e = -0.0006510415484042
n = 32, S = 3.1414298931749753, e = -0.0001627604148178
```

3.4 考察

式 1 の解析解は、式 3 のようになる。

$$
\begin{aligned}
\int_0^{\frac{\pi}{6}} \frac{dx}{\cos x} &= \left[\frac{1}{2}\log\left(\frac{1+\sin x}{1-\sin x}\right)\right]_0^{\frac{\pi}{6}} \\
&= \frac{1}{2}\log 3
\end{aligned} \tag{3}
$$

リスト 2 を見ると、n が大きくなるにつれ、数値解と解析解の差が少なくなっていることが分かる。具体的には、分割数が 2 倍になると、誤差は約 1/4 になっていることが分かる。

4　課題 2：シンプソンの公式

4.1　課題内容

式 4 について、シンプソン公式を用いて数値積分を行う。

$$
\int_0^{\frac{\pi}{2}} \sin x dx \tag{4}
$$

4.2　プログラムリスト

課題 2 のプログラムリストをリスト 3 に示す。

分割数を n, 分割幅を h とすると、シンプソン公式は式 5 のように表せる。この計算を `simpson` 関数内で行うことで、定積分の近似解を導出している。

$$
\int_a^b f(x)dx \simeq \frac{h}{3}\left[f(a) + 4\sum_{j=1}^{\frac{n}{2}} f(2jh-1) + 2\sum_{j=1}^{\frac{n}{2}-1} f(2jh) + f(b)\right] \tag{5}
$$

リスト 3　課題 2 のプログラム

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

#define a 0.0
#define b (M_PI / 2)
#define SOLUTION 1

double f(double);
double simpson(int);

int main(void) {
    int i;

    for (i = 2; i <= 32; i *= 2) {
        double S = simpson(i);
        printf("n = %2d, S = %.16f, e = %.16f\n", i, S, S - SOLUTION);
    }

    return 0;
}

// f(x)
double f(double x) {
```

```c
    return sin(x);
}

// シンプソンの公式
double simpson(int n) {
    int i;
    double x;
    double h = (b - a) / n;
    double S = f(a) + f(b);

    for (i = 1; i <= n - 1; i += 2) {
        x = a + i * h;
        S += f(x) * 4;
    }

    for (i = 2; i <= n - 2; i += 2) {
        x = a + i * h;
        S += f(x) * 2;
    }

    S = S * h / 3;
    return S;
}
```

4.3 実行結果

double 型の場合の実行結果をリスト 4, float 型の場合の実行結果をリスト 5 に示す。

リスト 4　課題 2 の実行結果（double 型の場合）

```
n =  2, S = 1.0022798774922104, e = 0.0022798774922104
n =  4, S = 1.0001345849741938, e = 0.0001345849741938
n =  8, S = 1.0000082955239677, e = 0.0000082955239677
n = 16, S = 1.0000005166847066, e = 0.0000005166847066
n = 32, S = 1.0000000322650009, e = 0.0000000322650009
```

リスト 5　課題 2 の実行結果（float 型の場合）

```
n =  2, S = 1.0022798776626587, e = 0.0022798776626587
n =  4, S = 1.0001345872879028, e = 0.0001345872879028
n =  8, S = 1.0000083446502686, e = 0.0000083446502686
n = 16, S = 1.0000005960464478, e = 0.0000005960464478
n = 32, S = 1.0000001192092896, e = 0.0000001192092896
```

4.4 考察

式 4 の解析解は、式 6 のようになる。

$$\begin{aligned}\int_0^{\frac{\pi}{2}} \sin x dx &= [-\cos(x)]_0^{\frac{\pi}{2}} \\ &= 1 \end{aligned} \tag{6}$$

リスト 4 を見ると、n が大きくなるにつれ、数値解と解析解の差が少なくなっていることが分かる。具体的には、分割数が 2 倍になると、誤差は約 1/16 になっていることが分かる。

また、リスト 4 とリスト 5 比べると、float 型で計算を行ったときの方が誤差が大きくなっていることが分かる。特に、$n = 32$ の場合を比較すると、float 型の誤差は double 型に比べて約 4 倍と、非常に大きな差が生じている。

5　課題 3–5：常微分方程式

5.1　課題内容

式 7 をオイラー法、ホイン法、ルンゲ・クッタ法を用いて解く。ただし、$t = 0$ のとき $u = 1$ とする。

$$\frac{du}{dt} = u \tag{7}$$

5.2　プログラムリスト

課題 3 のプログラムリストをリスト 6 に示す。

オイラー法は式 8, ホイン法は式 9, ルンゲ・クッタ法は式 10 のように表せる。これらの計算を、それぞれ `euler`, `heun`, `runge_kutta` 関数内で行うことで、解を導出している。

$$\begin{aligned} x_{i+1} &= x_i + h \\ y_{i+1} &= y_i + hf(x_i, y_i) \end{aligned} \tag{8}$$

$$\begin{aligned} x_{i+1} &= x_i + h \\ k_1 &= hf(x_i, y_i) \\ k_2 &= hf(x_i + h, y_i + k_1) \\ y_{i+1} &= y_i + \frac{1}{2}(k_1 + k_2) \end{aligned} \tag{9}$$

$$\begin{aligned} x_{i+1} &= x_i + h \\ k_1 &= hf(x_i, y_i) \\ k_2 &= hf(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}) \\ k_3 &= hf(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}) \\ k_4 &= hf(x_i + h, y_i + k_3) \\ y_{i+1} &= y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \tag{10}$$

リスト 6　課題 3–5 のプログラム

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

#define t0 0.0
#define u0 1.0
#define H 0.1
#define STEP 10

double up(double, double);
```

```c
double us(double);
void euler(double *, double *, double, int);
void heun(double *, double *, double, int);
void runge_kutta(double *, double *, double, int);

int main(void) {
    int i;
    double t[STEP + 1];
    double u[STEP + 1];

    euler(t, u, H, STEP);

    printf("オイラー法\n");
    for (i = 0; i <= STEP; i++)
        printf("i = %2d, t = %.16f, u = %.16f, e = %.16f\n", i, t[i], u[i], u[i] - us(t[i]));
    printf("\n");

    heun(t, u, H, STEP);

    printf("ホイン法\n");
    for (i = 0; i <= STEP; i++)
        printf("i = %2d, t = %.16f, u = %.16f, e = %.16f\n", i, t[i], u[i], u[i] - us(t[i]));
    printf("\n");

    runge_kutta(t, u, H, STEP);

    printf("ルンゲ・クッタ法\n");
    for (i = 0; i <= STEP; i++)
        printf("i = %2d, t = %.16f, u = %.16f, e = %.16f\n", i, t[i], u[i], u[i] - us(t[i]));

    return 0;
}

// u'(t, u)
double up(double t, double u) {
    return u;
}

// uの解析解
double us(double t) {
    return exp(t);
}

// オイラー法
void euler(double *t, double *u, double h, int step) {
    int i;

    t[0] = t0;
    u[0] = u0;

    for (i = 0; i <= step - 1; i++) {
        t[i + 1] = t[i] + h;
        u[i + 1] = u[i] + h * up(t[i], u[i]);
```

```c
    }
}

// ホイン法
void heun(double *t, double *u, double h, int step) {
    int i;

    t[0] = t0;
    u[0] = u0;

    for (i = 0; i <= step - 1; i++) {
        double k1 = h * up(t[i], u[i]);
        double k2 = h * up(t[i] + h, u[i] + k1);

        t[i + 1] = t[i] + h;
        u[i + 1] = u[i] + (k1 + k2) / 2;
    }
}

// ルンゲ・クッタ法
void runge_kutta(double *t, double *u, double h, int step) {
    int i;

    t[0] = t0;
    u[0] = u0;

    for (i = 0; i <= step - 1; i++) {
        double k1 = h * up(t[i], u[i]);
        double k2 = h * up(t[i] + h / 2, u[i] + k1 / 2);
        double k3 = h * up(t[i] + h / 2, u[i] + k2 / 2);
        double k4 = h * up(t[i] + h, u[i] + k3);

        t[i + 1] = t[i] + h;
        u[i + 1] = u[i] + (k1 + k2 * 2 + k3 * 2 + k4) / 6;
    }
}
```

5.3 実行結果

リスト 7　課題 3–5 の実行結果

```
オイラー法
i =  0, t = 0.0000000000000000, u = 1.0000000000000000, e = 0.0000000000000000
i =  1, t = 0.1000000000000000, u = 1.1000000000000001, e = -0.0051709180756476
i =  2, t = 0.2000000000000000, u = 1.2100000000000002, e = -0.0114027581601697
i =  3, t = 0.3000000000000000, u = 1.3310000000000002, e = -0.0188588075760030
i =  4, t = 0.4000000000000000, u = 1.4641000000000002, e = -0.0277246976412702
i =  5, t = 0.5000000000000000, u = 1.6105100000000001, e = -0.0382112707001281
i =  6, t = 0.6000000000000000, u = 1.7715610000000002, e = -0.0505578003905087
i =  7, t = 0.7000000000000000, u = 1.9487171000000001, e = -0.0650356074704765
i =  8, t = 0.7999999999999999, u = 2.1435888100000002, e = -0.0819521184924672
i =  9, t = 0.8999999999999999, u = 2.3579476910000001, e = -0.1016554201569493
```

```
i = 10, t = 0.9999999999999999, u = 2.5937424601000001, e = -0.1245393683590450

ホイン法
i =  0, t = 0.0000000000000000, u = 1.0000000000000000, e = 0.0000000000000000
i =  1, t = 0.1000000000000000, u = 1.1050000000000000, e = -0.0001709180756477
i =  2, t = 0.2000000000000000, u = 1.2210250000000000, e = -0.0003777581601698
i =  3, t = 0.3000000000000000, u = 1.3492326250000000, e = -0.0006261825760032
i =  4, t = 0.4000000000000000, u = 1.4909020506249999, e = -0.0009226470162704
i =  5, t = 0.5000000000000000, u = 1.6474467659406249, e = -0.0012745047595033
i =  6, t = 0.6000000000000000, u = 1.8204286763643904, e = -0.0016901240261185
i =  7, t = 0.7000000000000000, u = 2.0115736873826515, e = -0.0021790200878251
i =  8, t = 0.7999999999999999, u = 2.2227889245578298, e = -0.0027520039346376
i =  9, t = 0.8999999999999999, u = 2.4561817616364019, e = -0.0034213495205475
i = 10, t = 0.9999999999999999, u = 2.7140808466082240, e = -0.0042009818508211

ルンゲ・クッタ法
i =  0, t = 0.0000000000000000, u = 1.0000000000000000, e = 0.0000000000000000
i =  1, t = 0.1000000000000000, u = 1.1051708333333334, e = -0.0000000847423143
i =  2, t = 0.2000000000000000, u = 1.2214025708506946, e = -0.0000001873094753
i =  3, t = 0.3000000000000000, u = 1.3498584970625378, e = -0.0000003105134654
i =  4, t = 0.4000000000000000, u = 1.4918242400806858, e = -0.0000004575605845
i =  5, t = 0.5000000000000000, u = 1.6487206385968383, e = -0.0000006321032899
i =  6, t = 0.6000000000000000, u = 1.8221179620919332, e = -0.0000008382985757
i =  7, t = 0.7000000000000000, u = 2.0137516265967768, e = -0.0000010808736999
i =  8, t = 0.7999999999999999, u = 2.2255395632923154, e = -0.0000013652001520
i =  9, t = 0.8999999999999999, u = 2.4596014137800708, e = -0.0000016973768786
i = 10, t = 0.9999999999999999, u = 2.7182797441351658, e = -0.0000020843238793
```

5.4 考察

リスト 6 の 3 つの手法の実行結果を見比べると、オイラー法よりホイン法、ホイン法よりルンゲ・クッタ法の方が誤差が少ないことが分かる。これは i が大きくなるにつれて顕著に現れる。

$h = 0.1$, $h = 0.05$ とし、オイラー法で計算した数値解と、解析解を比較したグラフを図 1 に示す。

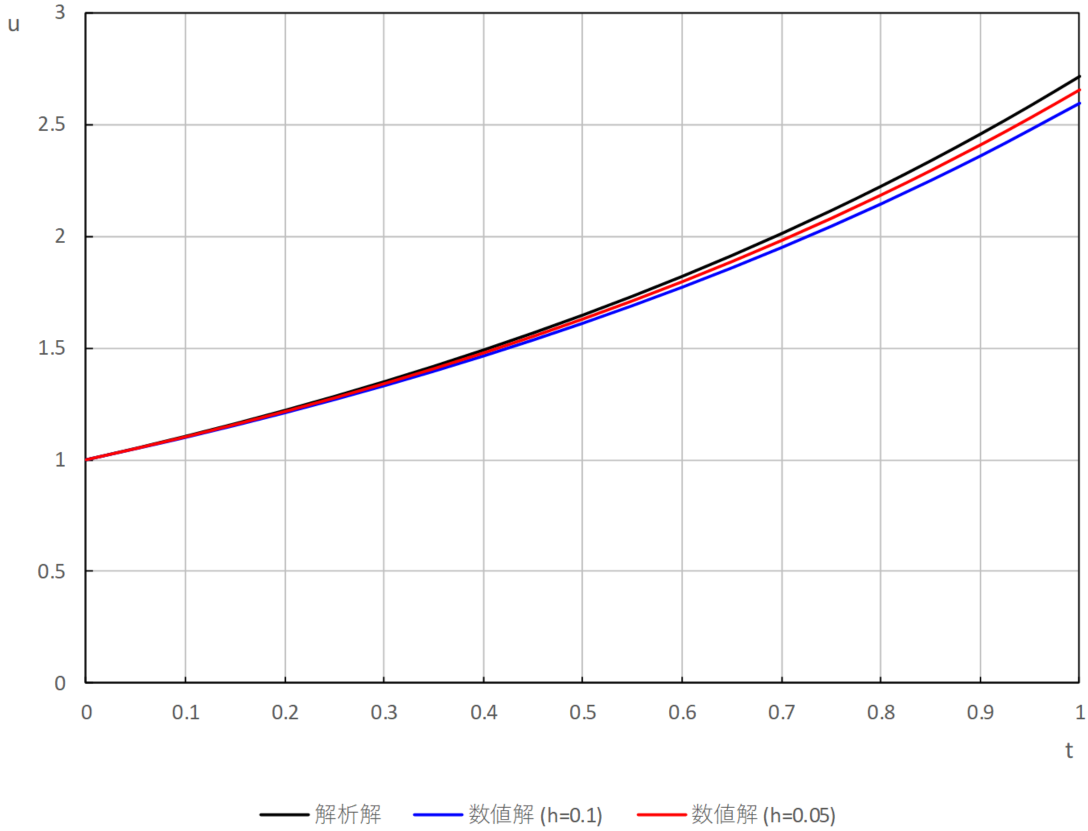

図 1　オイラー法の数値解と解析解