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：2019/12/12（木）
提出期限：2019/12/12（木）17:00

1　課題 1

課題 1 では、台形公式、式 (1) を用いて式 (2) について数値積分を行う。さらに、台形公式を使用する際に分割数を 1,2,4,... のように 1/2 ずつ細かくしていき、台形公式で求めた積分値の結果と解析解との関係を報告する。

$$\int_a^b f(x)dx \approx \frac{h}{2}\left[y_0 + 2\sum_{j=1}^{n-1} y_j + y_n\right] \tag{1}$$

$$\int_0^{\frac{\pi}{6}} \frac{dx}{\cos x} \tag{2}$$

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 1 のプログラム

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

double integration_func(double x);
double trapezoidal_rule(double a, double b, int N);

int main (void){
    int x = 3;
    int N = 1;

    for (; N <= 512; N *= 2){
        double result = trapezoidal_rule(0.0, M_PI / 6, N);
        printf("分割数 N = %d\n計算結果 = %f\n計算誤差 = %f\n\n", N, result, fabs(0.549306144 - result));
    }
    return 0;
}

//積分される関数
double integration_func (double x){
    double result = 1.0 / cos(x);
    return result;
}

//台形公式
//積分範囲:a -> b
//分割数N
double trapezoidal_rule (double a,double b, int N){
    double h = (b - a) / N;

    double y_0 = integration_func(a);
    double y_n = integration_func(b);
    double tmp = a;
    double y_j = 0.0;
    double res_tmp = 0.0;

    for (int i = 0; i < N-1; i++){
        tmp = tmp + h;
        y_j = integration_func(tmp);
        res_tmp += y_j;
    }

    res_tmp *= 2.0;

    double result = (h / 2.0) * (y_0 + res_tmp + y_n);

    return result;
```

```
}
```

このプログラムでは、分割数 N を 1 から 512 まで計算している。そして、計算結果と式 2 の解析解の $\frac{1}{2}\log_e 3 = 0.549306144$ との差を表示する。なお、積分される関数及び台形公式の計算部分は使いやすくするためにそれぞれ個別の関数にしている。

1.2　プログラムの実行結果

実行結果を以下に示す。

```
分割数 N = 1
計算結果 = 0.564099
計算誤差 = 0.014793

分割数 N = 2
計算結果 = 0.553084
計算誤差 = 0.003778

分割数 N = 4
計算結果 = 0.550256
計算誤差 = 0.000950

分割数 N = 8
計算結果 = 0.549544
計算誤差 = 0.000238

分割数 N = 16
計算結果 = 0.549366
計算誤差 = 0.000059

分割数 N = 32
計算結果 = 0.549321
計算誤差 = 0.000015

分割数 N = 64
計算結果 = 0.549310
計算誤差 = 0.000004

分割数 N = 128
```

```
計算結果 = 0.549307
計算誤差 = 0.000001

分割数 N = 256
計算結果 = 0.549306
計算誤差 = 0.000000

分割数 N = 512
計算結果 = 0.549306
計算誤差 = 0.000000
```

1.3 考察

計算結果と解析解との差の様子を図 1 に示す。

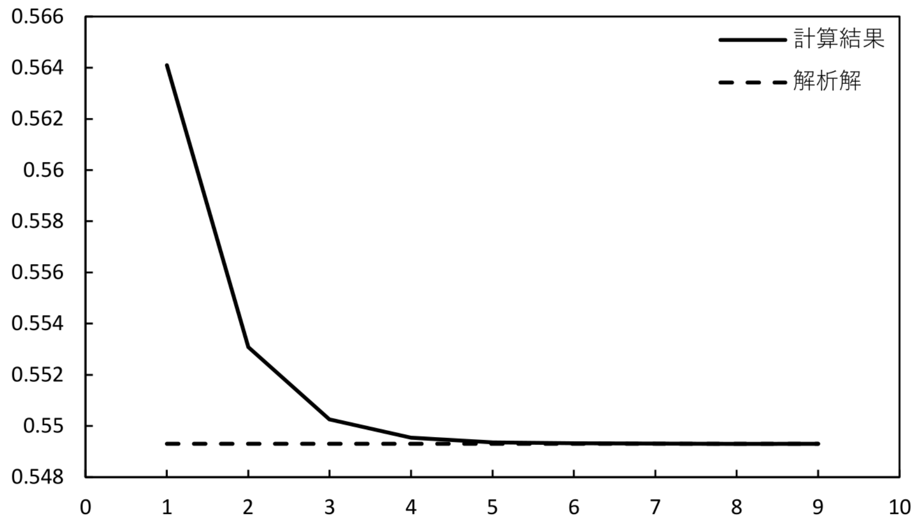

図 1　計算結果と解析解の差の様子

実行結果より、分割数 N が 1/2 になるごとに計算誤差が 1/4 ずつ減っていることがわかる。また、図 1 より、分割数 N が増えるたびに計算結果と解析解の差が減って誤差が減って行くことが確認できる。

2　課題 2

課題 2 では、シンプソンの公式、式 (3) を用いて式 (4) について数値積分を行う。さらに `float` 型と `double` 型で実行し、丸め誤差が現れる刻み幅を調べる。また、刻み幅を 1/2 にした時の誤差の減り方について報告する。

$$\int_a^b f(x)dx \approx \frac{h}{3}[y_0 + 4(y_1 + y_3 + \cdots + y_{n-1}) + 2(y_2 + y_4 + \cdots + y_{n-2}) + y_n] \tag{3}$$

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

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 2 のプログラム

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

double integration_func_double(double x);
double simpson_rule_double(double a, double b, int N);

float integration_func_float(float x);
float simpson_rule_float(float a, float b, int N);


int main (void){
    int N = 50;

    printf("double\n刻み幅, 計算結果, 誤差\n");
    for(;N >= 2; N -= 2){
    double result_d = simpson_rule_double(0.0, M_PI / 2, N);
    printf("%d,%lf,%lf,\n", N, result_d, fabs(1.0-result_d));
    }

    N = 50;
    printf("\nfloat\n刻み幅, 計算結果, 誤差\n");
    for(;N >= 2; N -= 2){
    float result_f = simpson_rule_float(0.0, M_PI / 2, N);
    printf("%d,%f,%f,\n", N, result_f, fabsf(1.0-result_f));
    }
    return 0;
}

//積分される関数
//double
double integration_func_double (double x){
    double result = sin(x);
    return result;
}

//シンプソンの公式
//double
double simpson_rule_double (double a,double b, int N){
    double h = (b - a) / N;

    double y0 = integration_func_double(a);
    double Yn = integration_func_double(b);
    //奇数
    double tmp_odd = a - h;
    double Y_odd = 0.0;
    double odd_res_tmp = 0.0;
    //偶数
    double tmp_even = a;
    double Y_even = 0.0;
    double even_res_tmp = 0.0;

    for (int i = 1; i < N; i += 2){
        tmp_odd = tmp_odd + 2 * h;
        Y_odd = integration_func_double(tmp_odd);
        odd_res_tmp += Y_odd;
    }

    for (int i = 2; i < N; i += 2){
        tmp_even = tmp_even + 2 * h;
```

```
        Y_even = integration_func_double(tmp_even);
        even_res_tmp += Y_even;
    }

    odd_res_tmp *= 4.0;
    even_res_tmp *= 2.0;

    double result = (h / 3.0) * (y0 + odd_res_tmp  + even_res_tmp + Yn);

    return result;
}

//積分される関数
//float
float integration_func_float (float x){
    float result = sin(x);
    return result;
}

//シンプソンの公式
//float
float simpson_rule_float (float a,float b, int N){
    float h = (b - a) / N;

    float y0 = integration_func_float(a);
    float Yn = integration_func_float(b);
    //奇数
    float tmp_odd = a - h;
    float Y_odd = 0.0;
    float odd_res_tmp = 0.0;
    //偶数
    float tmp_even = a;
    float Y_even = 0.0;
    float even_res_tmp = 0.0;

    for (int i = 1; i < N; i += 2){
        tmp_odd = tmp_odd + 2 * h;
        Y_odd = integration_func_float(tmp_odd);
        odd_res_tmp += Y_odd;
    }

    for (int i = 2; i < N; i += 2){
        tmp_even = tmp_even + 2 * h;
        Y_even = integration_func_float(tmp_even);
        even_res_tmp += Y_even;
    }

    odd_res_tmp *= 4.0;
    even_res_tmp *= 2.0;

    float result = (h / 3.0) * (y0 + odd_res_tmp  + even_res_tmp + Yn);

    return result;
}
```

このプログラムでは、`float` 型と `double` 型でシンプソンの公式を実行する。そして、(計算結果ー真値)の絶対値を表示している。なお、式 (4) の真値は 1.0 である。また、刻み幅 N は 2 から 50 まで計算している。

2.2 プログラムの実行結果

実行結果を以下に示す。

```
$ ./kadai2
double
刻み幅, 計算結果, 誤差
50,1.000000,0.000000,
```

```
48,1.000000,0.000000,
46,1.000000,0.000000,
44,1.000000,0.000000,
42,1.000000,0.000000,
40,1.000000,0.000000,
38,1.000000,0.000000,
36,1.000000,0.000000,
34,1.000000,0.000000,
32,1.000000,0.000000,
30,1.000000,0.000000,
28,1.000000,0.000000,
26,1.000000,0.000000,
24,1.000000,0.000000,
22,1.000000,0.000000,
20,1.000000,0.000000,
18,1.000000,0.000000,
16,1.000001,0.000001,
14,1.000001,0.000001,
12,1.000002,0.000002,
10,1.000003,0.000003,
8,1.000008,0.000008,
6,1.000026,0.000026,
4,1.000135,0.000135,
2,1.002280,0.002280,

float
刻み幅, 計算結果, 誤差
50,1.000000,0.000000,
48,1.000000,0.000000,
46,1.000000,0.000000,
44,1.000000,0.000000,
42,1.000000,0.000000,
40,1.000000,0.000000,
38,1.000000,0.000000,
36,1.000000,0.000000,
34,1.000000,0.000000,
32,1.000000,0.000000,
30,1.000000,0.000000,
```

```
28,1.000000,0.000000,
26,1.000000,0.000000,
24,1.000000,0.000000,
22,1.000000,0.000000,
20,1.000000,0.000000,
18,1.000000,0.000000,
16,1.000001,0.000001,
14,1.000001,0.000001,
12,1.000002,0.000002,
10,1.000003,0.000003,
8,1.000008,0.000008,
6,1.000026,0.000026,
4,1.000135,0.000135,
2,1.002280,0.002280,
```

2.3　考察

実行結果より、丸め誤差が現れる刻み幅は 16 であることがわかる。また、刻み幅を 1/2 にしていった時の誤差の減り方の様子を図 2 に示す。

図 2 は片対数グラフであるため、誤差は指数関数的に減って行くことが確認できる。

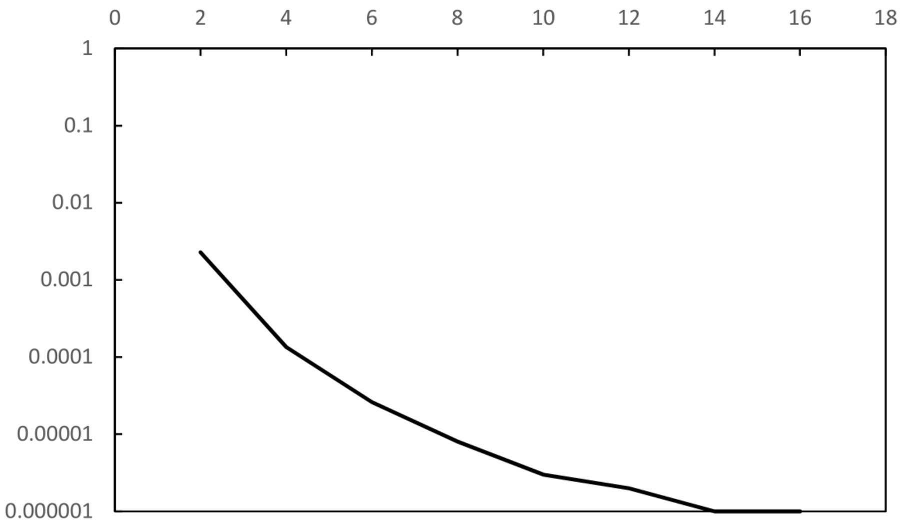

図 2　誤差の減少の様子

3　課題 3,4,5

課題 3 では、オイラー法を用いて式 (5) の微分方程式を解く。そして、解析解と数値解を同じグラフにプロットし、オイラー法がどの程度正しいかを報告する。

課題 4 では、課題 3 をホイン法を用いて同様に行う。また、誤差の特徴についても同様に調べる。

課題 5 では、課題 3 をルンゲクッタ法を用いて同様に行う。また、誤差の特徴についても同様に調べる。

$$\frac{du}{dt} = u \,(\text{ただし、}t = 0 \text{ のとき } u = 1) \tag{5}$$

オイラー法

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

ホイン法

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

ルンゲ・クッタ法

$$x_{i+1} = x_i + h$$
$$k_1 = hf(x_i, y_i)$$
$$k_2 = hf\left(x_i + \tfrac{h}{2}, y_i + \tfrac{k_1}{2}\right)$$
$$k_3 = hf\left(x_i + \tfrac{h}{2}, y_i + \tfrac{k_2}{2}\right)$$
$$k_4 = hf\left(x_i + h, y_i + k_3\right)$$
$$y_{i+1} = y_i + \tfrac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$

式 (5) の解析解は、

$$\frac{du}{dt} = u$$
$$\int \frac{1}{u} du = \int dt$$
$$\log |u| = t + c$$
$$u(t) = \pm e^{t+c} = \pm e^c e^t$$

ここで、$C = \pm e^c$ とすると

$$u(t) = Ce^t$$

初期条件より、

$$u_0 = Ce^0$$

$$C = 1$$

よって

$$u(t) = e^t$$

3.1 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 3 に示す。

ソースコード 3 課題 3、4、5 のプログラム

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

double diff_equa (double t, double u);
double euler_rule (double t, double u, double h, int step);
double heun_method (double t, double u, double h, int step);
double RK_method (double t, double u, double h, int step);


double calculated_t = 0.0;
double calculated_u = 0.0;
int main (void){
    double t = 0.0;
    double u = 1.0;
    double h = 0.025;
    int step = 40;

    //printf("刻み幅 = %f\n", h);

    //printf("オイラーの公式\n");
    printf("i, t, u\n");
    euler_rule(t, u, h, step);

    /*printf("ホイン法\n");
    printf("i, t, u\n");
    heun_method(t, u, h, step);

    printf("RK法\n");
    printf("i, t, u\n");
    RK_method(t, u, h, step);*/

    //printf ("t0 = %f, u0 = %f\n", t, u);
   // printf ("t1 = %f, u1 = %f\n", new_t, new_u);

}

//微分方程式
double diff_equa (double t, double u){
    double result = u;
    return result;
}

//オイラーの公式
double euler_rule (double t, double u, double h, int step){
    double old_t = t;
    double old_u = u;
    double new_t = 0;
    double new_u = 0;
    for(int i = 0; i < step; i++){

        new_t = old_t + h;
        new_u = old_u + (h * diff_equa(old_t, old_u));
        old_t = new_t;
        old_u = new_u;
        printf("%d, %f, %f\n", i + 1, new_t, new_u);
    }


    double result = new_u;
    return result;
}
```

```c

//ホイン法
double heun_method (double t, double u, double h, int step){
    double t_i = t;
    double u_i = u;

    for(int i = 0; i < step; i++){
        double t_i1 = t_i + h;
        double k1 = (h * diff_equa(t_i, u_i));
        double k2 = (h * diff_equa(t_i + h, u_i + k1));

        double u_i1 = u_i + (0.5 * (k1 + k2));
        t_i = t_i1;
        u_i = u_i1;
        printf("%d, %f, %f\n", i + 1, t_i, u_i);
    }
    calculated_t = t_i;
    calculated_u = u_i;

    return 0;
}

//ルンゲ・クッタ法
double RK_method (double t, double u, double h, int step){
    double t_i = t;
    double u_i = u;

    for(int i = 0; i < step; i++){
        double t_i1 = t_i + h;
        double k1 = h * diff_equa(t_i, u_i);
        double k2 = h * diff_equa((t_i + h * 0.5), (u_i + k1 * 0.5));
        double k3 = h * diff_equa((t_i + h * 0.5), (u_i + k2 * 0.5));
        double k4 = h * diff_equa((t_i + h), (u_i + k3));

        double u_i1 = u_i + ((k1 + 2 * k2 + 2 * k3 + k4) / 6);
        t_i = t_i1;
        u_i = u_i1;
        printf("%d, %f, %f\n", i + 1, t_i, u_i);
    }
    calculated_t = t_i;
    calculated_u = u_i;

    return 0;
}
```

3.2 プログラムの実行結果

実行結果を以下に示す。なお、出力される数値データが多いため、実行結果を一部省略し、t の値が 0.10 と 1.00 の時のみを示している。出力された全データは付録 A に示す。

```
刻み幅 = 0.100000
オイラーの公式
i, t, u
1, 0.100000, 1.100000
...
10, 1.000000, 2.593742

ホイン法
i, t, u
1, 0.100000, 1.105000
```

```
...
10, 1.000000, 2.714081

RK 法
i, t, u
1, 0.100000, 1.105171
...
10, 1.000000, 2.718280


刻み幅 = 0.050000
オイラーの公式
i, t, u
1, 0.050000, 1.050000
...
20, 1.000000, 2.653298

ホイン法
i, t, u
1, 0.050000, 1.051250
...
20, 1.000000, 2.717191

RK 法
i, t, u
1, 0.050000, 1.051271
...
20, 1.000000, 2.718282


刻み幅 = 0.025000
オイラーの公式
i, t, u
1, 0.025000, 1.025000
...
40, 1.000000, 2.685064

ホイン法
i, t, u
```

```
1, 0.025000, 1.025313
...
40, 1.000000, 2.718004

RK 法
i, t, u
1, 0.025000, 1.025315
...
40, 1.000000, 2.718282
```

3.3 考察

図 3 はオイラー法で式 (5) を解いて、刻み幅を $h = 0.1, h = 0.05, h = 0.025$ に変更していき同じ t の値をグラフにプロットしたものである。

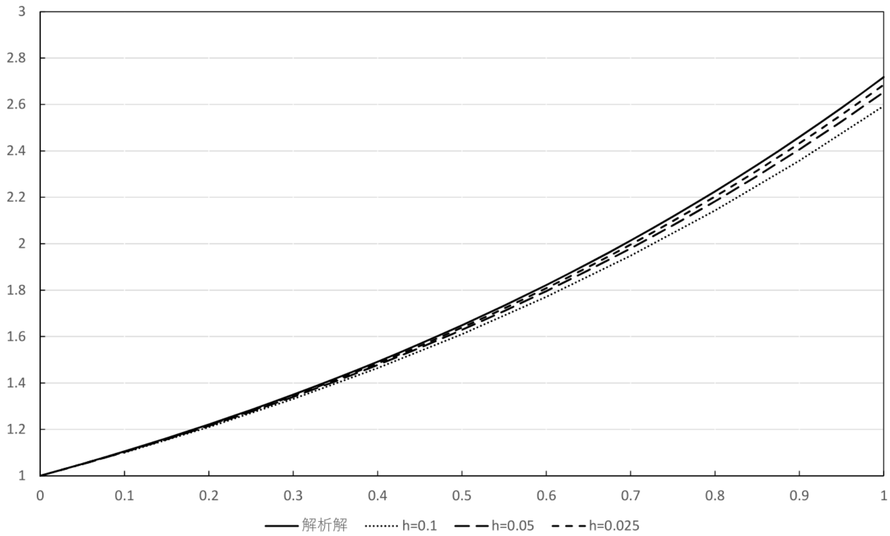

図 3　オイラー法 解析解との差

刻み幅を小さくするほど実線で表された解析解に近づく事がわかる。しかし、t の値が大きくなるにつれて差が大きくなってしまう。

図 4 はオイラー法で式 (5) を解いて、刻み幅を $h = 0.1, h = 0.05, h = 0.025$ に変更していき同じ t の値をグラフにプロットしたものである。

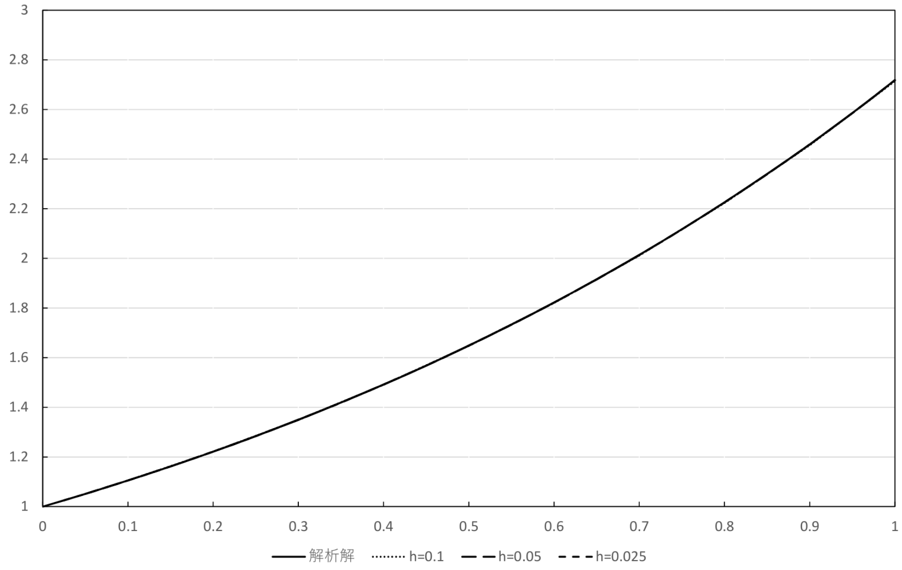

図 4　ホイン法 解析解との差

数値解は解析解とほぼ一致していることがわかる。

図 5 はオイラー法で式 (5) を解いて、刻み幅を $h = 0.1, h = 0.05, h = 0.025$ に変更していき同じ t の値をグラフにプロットしたものである。

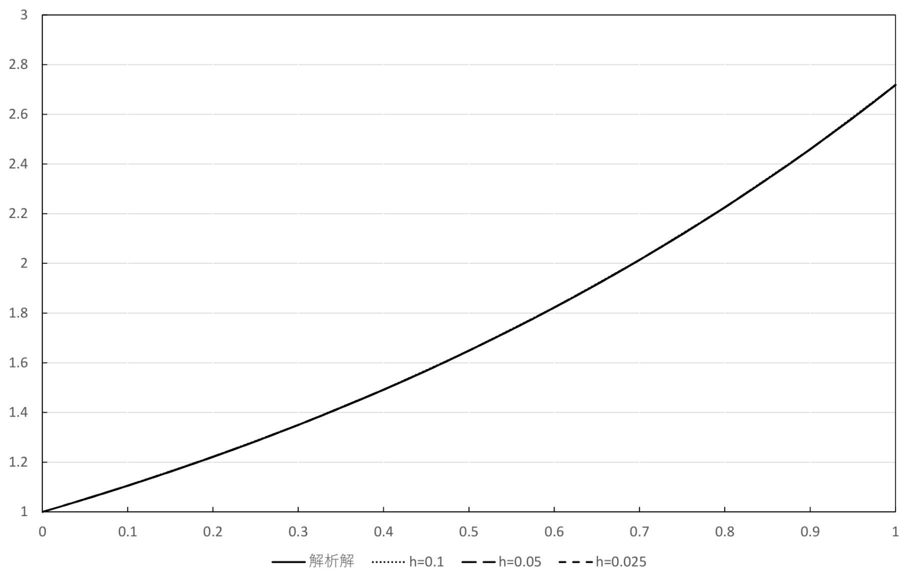

図 5　ルンゲ・クッタ法 解析解との差

数値解は解析解とほぼ一致していることがわかる。

付録 A　課題 3、4、5 実行結果

課題 3、4、5 実行結果を以下に示す。

```
刻み幅 = 0.100000
オイラーの公式
i, t, u
1, 0.100000, 1.100000
2, 0.200000, 1.210000
3, 0.300000, 1.331000
4, 0.400000, 1.464100
5, 0.500000, 1.610510
6, 0.600000, 1.771561
7, 0.700000, 1.948717
8, 0.800000, 2.143589
9, 0.900000, 2.357948
10, 1.000000, 2.593742
ホイン法
i, t, u
1, 0.100000, 1.105000
2, 0.200000, 1.221025
3, 0.300000, 1.349233
4, 0.400000, 1.490902
5, 0.500000, 1.647447
6, 0.600000, 1.820429
7, 0.700000, 2.011574
8, 0.800000, 2.222789
9, 0.900000, 2.456182
10, 1.000000, 2.714081
RK 法
i, t, u
1, 0.100000, 1.105171
2, 0.200000, 1.221403
3, 0.300000, 1.349858
4, 0.400000, 1.491824
5, 0.500000, 1.648721
6, 0.600000, 1.822118
7, 0.700000, 2.013752
8, 0.800000, 2.225540
9, 0.900000, 2.459601
10, 1.000000, 2.718280
```

```
刻み幅 = 0.050000
オイラーの公式
i, t, u
1, 0.050000, 1.050000
2, 0.100000, 1.102500
3, 0.150000, 1.157625
4, 0.200000, 1.215506
5, 0.250000, 1.276282
6, 0.300000, 1.340096
7, 0.350000, 1.407100
8, 0.400000, 1.477455
9, 0.450000, 1.551328
10, 0.500000, 1.628895
11, 0.550000, 1.710339
12, 0.600000, 1.795856
13, 0.650000, 1.885649
14, 0.700000, 1.979932
15, 0.750000, 2.078928
16, 0.800000, 2.182875
17, 0.850000, 2.292018
18, 0.900000, 2.406619
19, 0.950000, 2.526950
20, 1.000000, 2.653298
ホイン法
i, t, u
1, 0.050000, 1.051250
2, 0.100000, 1.105127
3, 0.150000, 1.161764
4, 0.200000, 1.221305
5, 0.250000, 1.283897
6, 0.300000, 1.349696
7, 0.350000, 1.418868
8, 0.400000, 1.491585
9, 0.450000, 1.568029
10, 0.500000, 1.648390
11, 0.550000, 1.732870
12, 0.600000, 1.821680
13, 0.650000, 1.915041
```

```
14, 0.700000, 2.013187
15, 0.750000, 2.116363
16, 0.800000, 2.224826
17, 0.850000, 2.338849
18, 0.900000, 2.458715
19, 0.950000, 2.584724
20, 1.000000, 2.717191
RK 法
i, t, u
1, 0.050000, 1.051271
2, 0.100000, 1.105171
3, 0.150000, 1.161834
4, 0.200000, 1.221403
5, 0.250000, 1.284025
6, 0.300000, 1.349859
7, 0.350000, 1.419068
8, 0.400000, 1.491825
9, 0.450000, 1.568312
10, 0.500000, 1.648721
11, 0.550000, 1.733253
12, 0.600000, 1.822119
13, 0.650000, 1.915541
14, 0.700000, 2.013753
15, 0.750000, 2.117000
16, 0.800000, 2.225541
17, 0.850000, 2.339647
18, 0.900000, 2.459603
19, 0.950000, 2.585710
20, 1.000000, 2.718282


刻み幅 = 0.025000
オイラーの公式
i, t, u
1, 0.025000, 1.025000
2, 0.050000, 1.050625
3, 0.075000, 1.076891
4, 0.100000, 1.103813
```

```
5, 0.125000, 1.131408
6, 0.150000, 1.159693
7, 0.175000, 1.188686
8, 0.200000, 1.218403
9, 0.225000, 1.248863
10, 0.250000, 1.280085
11, 0.275000, 1.312087
12, 0.300000, 1.344889
13, 0.325000, 1.378511
14, 0.350000, 1.412974
15, 0.375000, 1.448298
16, 0.400000, 1.484506
17, 0.425000, 1.521618
18, 0.450000, 1.559659
19, 0.475000, 1.598650
20, 0.500000, 1.638616
21, 0.525000, 1.679582
22, 0.550000, 1.721571
23, 0.575000, 1.764611
24, 0.600000, 1.808726
25, 0.625000, 1.853944
26, 0.650000, 1.900293
27, 0.675000, 1.947800
28, 0.700000, 1.996495
29, 0.725000, 2.046407
30, 0.750000, 2.097568
31, 0.775000, 2.150007
32, 0.800000, 2.203757
33, 0.825000, 2.258851
34, 0.850000, 2.315322
35, 0.875000, 2.373205
36, 0.900000, 2.432535
37, 0.925000, 2.493349
38, 0.950000, 2.555682
39, 0.975000, 2.619574
40, 1.000000, 2.685064
ホイン法
i, t, u
```

```
1, 0.025000, 1.025313
2, 0.050000, 1.051266
3, 0.075000, 1.077876
4, 0.100000, 1.105160
5, 0.125000, 1.133134
6, 0.150000, 1.161816
7, 0.175000, 1.191225
8, 0.200000, 1.221378
9, 0.225000, 1.252294
10, 0.250000, 1.283993
11, 0.275000, 1.316494
12, 0.300000, 1.349817
13, 0.325000, 1.383985
14, 0.350000, 1.419017
15, 0.375000, 1.454936
16, 0.400000, 1.491764
17, 0.425000, 1.529524
18, 0.450000, 1.568240
19, 0.475000, 1.607936
20, 0.500000, 1.648637
21, 0.525000, 1.690368
22, 0.550000, 1.733156
23, 0.575000, 1.777026
24, 0.600000, 1.822007
25, 0.625000, 1.868127
26, 0.650000, 1.915414
27, 0.675000, 1.963897
28, 0.700000, 2.013609
29, 0.725000, 2.064578
30, 0.750000, 2.116838
31, 0.775000, 2.170420
32, 0.800000, 2.225359
33, 0.825000, 2.281688
34, 0.850000, 2.339444
35, 0.875000, 2.398661
36, 0.900000, 2.459377
37, 0.925000, 2.521630
38, 0.950000, 2.585459
```

```
39, 0.975000, 2.650903
40, 1.000000, 2.718004
RK 法
i, t, u
1, 0.025000, 1.025315
2, 0.050000, 1.051271
3, 0.075000, 1.077884
4, 0.100000, 1.105171
5, 0.125000, 1.133148
6, 0.150000, 1.161834
7, 0.175000, 1.191246
8, 0.200000, 1.221403
9, 0.225000, 1.252323
10, 0.250000, 1.284025
11, 0.275000, 1.316531
12, 0.300000, 1.349859
13, 0.325000, 1.384031
14, 0.350000, 1.419068
15, 0.375000, 1.454991
16, 0.400000, 1.491825
17, 0.425000, 1.529590
18, 0.450000, 1.568312
19, 0.475000, 1.608014
20, 0.500000, 1.648721
21, 0.525000, 1.690459
22, 0.550000, 1.733253
23, 0.575000, 1.777131
24, 0.600000, 1.822119
25, 0.625000, 1.868246
26, 0.650000, 1.915541
27, 0.675000, 1.964033
28, 0.700000, 2.013753
29, 0.725000, 2.064731
30, 0.750000, 2.117000
31, 0.775000, 2.170592
32, 0.800000, 2.225541
33, 0.825000, 2.281881
34, 0.850000, 2.339647
```

```
35, 0.875000, 2.398875
36, 0.900000, 2.459603
37, 0.925000, 2.521868
38, 0.950000, 2.585710
39, 0.975000, 2.651167
40, 1.000000, 2.718282
```