令和四年度後期 数理工学実験 課題レポート

熱方程式の差分法

情報学科 2 回生 平田蓮

学生番号: 1029342830

実験日: 10 月 25, 31 日, 11 月 1, 7 日

実験場所: 京都大学工学部総合校舎数理計算機室

11 月 14 日 提出

目次

1 目的

本実験では、熱方程式 (拡散方程式) の差分法について学ぶ。

2 原理

本節では、3 で行う課題で用いるアルゴリズムについて述べる。

2.1 拡散方程式

微粒子などが空間上で自発的に広がる現象を拡散といい、拡散方程式はその振る舞いを記す。ここでは 1 次元空間上の拡散を考える。微粒子の濃度を $C(x,t)$, 流れを $J(x,t)$ とする。微粒子の保存を考えると、

$$\frac{\partial}{\partial t}C(x,t) = -\frac{\partial}{\partial x}J(x,t) \tag{2.1}$$

を得る。また、J は濃度勾配に比例すると考えると、比例定数 D を用いて、

$$J(x,t) = -D\frac{\partial}{\partial x}C(x,t) \tag{2.2}$$

と書ける。これらを合わせて、拡散方程式

$$\frac{\partial}{\partial t}C(x,t) = D\frac{\partial^2}{\partial x^2}C(x,t)$$

を得る。

$D = 1$ として、方程式

$$\frac{\partial}{\partial t}u(x,t) = \frac{\partial^2}{\partial x^2}u(x,t) \tag{2.3}$$

を初期条件 $u(x,0) = u_0(x)$、範囲 $x \in [0,L]$ のもとで考える。これを解くには、初期条件の他に、$x = 0, L$ での境界条件が必要である。本実験では、主に以下の境界条件を用いる。

■Dirichlet **境界条件**　境界上の u の値を与える。

$$u(0,t) = u_L$$
$$u(L,t) = u_R$$

■Neumann **境界条件**　境界上の u の傾きを与える。

$$\frac{\partial u}{\partial x}(0,t) = J_L$$
$$\frac{\partial u}{\partial x}(L,t) = J_R$$

2.2 数値解法

時間と空間を離散化して式 (2.3) を解くことを考える。時間と空間の刻み幅をそれぞれ $\Delta t, \Delta x$、空間の刻み数を N とする。$(\because L = N\Delta x)$ 離散化された時空間上での u の平均値を、

$$u_j^n \coloneqq \frac{1}{\Delta x}\int_{(j-1)\Delta x}^{j\Delta x} u(x, n\Delta t)\mathrm{d}x \ (1 \leq j \leq N, 0 \leq n)$$

とする。u_0^n, u_{N+1}^n を用いて境界条件を表す。

■Dirichlet 境界条件

$$u_0^n = u_L$$
$$u_{N+1}^n = u_R$$

■Neumann 境界条件

$$u_1^n - u_0^n = J_L\Delta x \rightarrow u_0^n = u_1^n - J_L\Delta x$$
$$u_{N+1}^n - u_N^n = J_R\Delta x \rightarrow u_{N+1}^n = u_N^n + J_R\Delta x$$

2.2.1 オイラー陽解法

式 (2.1) を t について $[n\Delta t, (n+1)\Delta t]$ の範囲で積分すると、

$$\int_{n\Delta t}^{(n+1)\Delta t} \frac{\partial}{\partial t}u(x,t)\mathrm{d}t = -\int_{n\Delta t}^{(n+1)\Delta t} \frac{\partial}{\partial x}J(x,t)\mathrm{d}t$$
$$\rightarrow u(x,(n+1)\Delta t) - u(x,n\Delta t) = -\int_{n\Delta t}^{(n+1)\Delta t} \frac{\partial}{\partial x}J(x,t)\mathrm{d}t$$

となり、右辺を近似すると、

$$\begin{aligned} u(x,(n+1)\Delta t) - u(x,n\Delta t) &= -\int_{n\Delta t}^{(n+1)\Delta t} \frac{\partial}{\partial x}J(x,t)\mathrm{d}t \\ &\approx -\Delta t\frac{\partial}{\partial x}J(x,n\Delta t) \end{aligned} \tag{2.4}$$

を得る。次にこれを x について $[(j-1)\Delta x, j\Delta x]$ の範囲で積分すると、

$$\int_{(j-1)\Delta x}^{j\Delta x}\{u(x,(n+1)\Delta t) - u(x,n\Delta t)\}\mathrm{d}x \approx -\int_{(j-1)\Delta x}^{j\Delta x}\Delta t\frac{\partial}{\partial x}J(x,n\Delta t)\mathrm{d}x$$
$$\rightarrow \frac{1}{\Delta x}\int_{(j-1)\Delta x}^{j\Delta x}\{u(x,(n+1)\Delta t) - u(x,n\Delta t)\}\mathrm{d}x \approx -\frac{1}{\Delta x}\int_{(j-1)\Delta x}^{j\Delta x}\Delta t\frac{\partial}{\partial x}J(x,n\Delta t)\mathrm{d}x$$
$$\rightarrow u_j^{n+1} - u_j^n \approx -\frac{1}{\Delta x}\int_{(j-1)\Delta x}^{j\Delta x}\Delta t\frac{\partial}{\partial x}J(x,n\Delta t)\mathrm{d}x$$
$$= -\frac{\Delta t}{\Delta x}\{J(j\Delta x, n\Delta t) - J((j-1)\Delta x, n\Delta t)\} \tag{2.5}$$

を得る。

式 (2.2) に $D = 1$ を代入すると、

$$J(j\Delta x, n\Delta t) = -\frac{\partial}{\partial x}u(j\Delta x, n\Delta t)$$
$$J((j-1)\Delta x, n\Delta t) = -\frac{\partial}{\partial x}u((j-1)\Delta x, n\Delta t)$$

を得る。これらの右辺は次のように近似できる。

$$J(j\Delta x, n\Delta t) \approx -\frac{u((j+1)\Delta x, n\Delta t) - u(j\Delta x, n\Delta t)}{\Delta x} \approx -\frac{u^n_{j+1} - u^n_j}{\Delta x}$$
$$J((j-1)\Delta x, n\Delta t) \approx -\frac{u(j\Delta x, n\Delta t) - u((j-1)\Delta x, n\Delta t)}{\Delta x} \approx -\frac{u^n_j - u^n_{j-1}}{\Delta x}$$

これを式 (2.5) に代入すると、

$$\begin{aligned} u^{n+1}_j - u^n_j &\approx -\frac{\Delta t}{\Delta x}\{J(j\Delta x, n\Delta t) - J((j-1)\Delta x, n\Delta t)\} \\ &\approx \frac{\Delta t}{\Delta x^2}(u^n_{j-1} - 2u^n_j + u^n_{j+1}) \\ \rightarrow u^{n+1}_j &\approx u^n_j + \frac{\Delta t}{\Delta x^2}(u^n_{j-1} - 2u^n_j + u^n_{j+1}) \end{aligned} \tag{2.6}$$

を得る。式 (2.6) をオイラー陽解法という。この式に従って値を n について更新することで拡散方程式を解くことができる。

2.2.2 クランク-ニコルソン法

式 (2.4) での近似をより高精度にすると、

$$u(x, (n+1)\Delta t) - u(x, n\Delta t) \approx -\frac{\Delta t}{2}\left\{\frac{\partial}{\partial x}J(x, (n+1)\Delta t) + \frac{\partial}{\partial x}J(x, n\Delta t)\right\}$$

と書ける。これを同様に x について積分すると、

$$\begin{aligned} u^{n+1}_j - u^n_j &\approx -\frac{1}{\Delta x}\int_{(j-1)\Delta x}^{j\Delta x}\frac{\Delta t}{2}\left\{\frac{\partial}{\partial x}J(x, (n+1)\Delta t) + \frac{\partial}{\partial x}J(x, n\Delta t)\right\}\mathrm{d}x \\ &= -\frac{\Delta t}{2\Delta x}[\{J(j\Delta x, (n+1)\Delta t) - J((j-1)\Delta x, (n+1)\Delta t)\} + \{J((j-1)\Delta x, n\Delta t) - J(j\Delta x, n\Delta t)\}] \\ &\approx \frac{\Delta t}{2\Delta x^2}\left\{(u^{n+1}_{j-1} - 2u^{n+1}_j + u^{n+1}_{j+1}) + (u^n_{j-1} - 2u^n_j + u^n_{j+1})\right\} \end{aligned} \tag{2.7}$$

を得る。式 (2.7) をクランク-ニコルソン法という。

2.3 LU 分解

オイラー陽解法は u^{n+1}_j が計算済みの $u^n_{\{j-1,j,j+1\}}$ を用いて表されているため、簡単に計算することができる。一方、クランク-ニコルソン法では、u^{n+1}_j が u^{n+1}_j を用いて表されているため、計算するには工夫が必要である。本実験では、LU 分解を用いて計算する。

式 (2.7) は、以下に定義する $A, \boldsymbol{x}, \boldsymbol{z}$ を用いて $A\boldsymbol{x} = \boldsymbol{z}$ と書ける。

$$A = \begin{cases} \begin{pmatrix} 1+c & -\frac{c}{2} & 0 & \cdots & 0 \\ -\frac{c}{2} & 1+c & -\frac{c}{2} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -\frac{c}{2} & 1+c & -\frac{c}{2} \\ 0 & \cdots & 0 & -\frac{c}{2} & 1+c \end{pmatrix} & \text{(Dirichlet 境界条件)} \\ \begin{pmatrix} 1+\frac{c}{2} & -\frac{c}{2} & 0 & \cdots & 0 \\ -\frac{c}{2} & 1+c & -\frac{c}{2} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -\frac{c}{2} & 1+c & -\frac{c}{2} \\ 0 & \cdots & 0 & -\frac{c}{2} & 1+\frac{c}{2} \end{pmatrix} & \text{(Neumann 境界条件)} \end{cases}$$

$$\boldsymbol{x} = \begin{pmatrix} u_1^{n+1} \\ u_2^{n+1} \\ \vdots \\ u_N^{n+1} \end{pmatrix}$$

$$\boldsymbol{z} = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{pmatrix} \quad z_j = \begin{cases} \begin{cases} (1-c)u_1^n + cu_L + \frac{c}{2}u_2^n & (j=1) \\ \frac{c}{2}u_{j-1}^n + (1-c)u_j^n + \frac{c}{2}u_{j+1}^n & (2 \le j \le N-1) \\ (1-c)u_N^n + cu_R + \frac{c}{2}u_{N-1}^n & (j=N) \end{cases} & \text{(Dirichlet 境界条件)} \\ \begin{cases} (1-\frac{c}{2})u_1^n - cJ_L\Delta x + \frac{c}{2}u_2^n & (j=1) \\ \frac{c}{2}u_{j-1}^n + (1-c)u_j^n + \frac{c}{2}u_{j+1}^n & (2 \le j \le N-1) \\ (1-\frac{c}{2})u_N^n + cJ_R\Delta x + \frac{c}{2}u_{N-1}^n & (j=N) \end{cases} & \text{(Neumann 境界条件)} \end{cases}$$

$$\left(c \coloneqq \frac{\Delta t}{\Delta x^2} \right)$$

A を下三角行列 L と上三角行列 U の積に分解することを考える。($\therefore A = LU$)

$$A = \begin{pmatrix} a_1 & b_1 & 0 & \cdots & 0 \\ c_1 & a_2 & b_2 & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & c_{N-2} & a_{N-1} & b_{N-1} \\ 0 & \cdots & 0 & c_{N-1} & a_N \end{pmatrix} \tag{2.8}$$

として、

$$L = \begin{pmatrix} \alpha_1 & 0 & 0 & \cdots & 0 \\ c_1 & \alpha_2 & 0 & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & c_{N-2} & \alpha_{N-1} & 0 \\ 0 & \cdots & 0 & c_{N-1} & \alpha_N \end{pmatrix}, U = \begin{pmatrix} 1 & \beta_1 & 0 & \cdots & 0 \\ 0 & 1 & \beta_2 & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 0 & 1 & \beta_{N-1} \\ 0 & \cdots & 0 & 0 & 1 \end{pmatrix}$$

と置くと、

$$LU = \begin{pmatrix} \alpha_1 & \alpha_1\beta_1 & 0 & \cdots & 0 \\ c_1 & \alpha_2 + c_1\beta_1 & \alpha_2\beta_2 & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & c_{N-2} & \alpha_{N-1} + c_{N-2}\beta_{N-2} & \alpha N - 1\beta N - 1 \\ 0 & \cdots & 0 & c_{N-1} & \alpha_N + c_{N-1}\beta_{N-1} \end{pmatrix} \tag{2.9}$$

となり、式 (2.8) と (2.9) を比較することで、α_j, β_j を j について昇順に求められることがわかる。

$A = LU$ と分解できたため、式 (2.7) は、

$$\begin{aligned} A\boldsymbol{x} &= \boldsymbol{z} \\ \rightarrow LU\boldsymbol{x} &= \boldsymbol{z} \\ \rightarrow L\boldsymbol{y} &= \boldsymbol{z} \quad \left(\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} \coloneqq U\boldsymbol{x} \right) \end{aligned}$$

と書き換えられる。$L\boldsymbol{y} = \boldsymbol{z}$ は、

$$\begin{pmatrix} \alpha_1 & 0 & 0 & \cdots & 0 \\ c_1 & \alpha_2 & 0 & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & c_{N-2} & \alpha_{N-1} & 0 \\ 0 & \cdots & 0 & c_{N-1} & \alpha_N \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} \alpha_1 y_1 \\ c_1 y_1 + \alpha_2 y_2 \\ \vdots \\ c_{N-1} y_{N-1} + \alpha_N y_N \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{pmatrix}$$

であるため、y_j は c_j, z_j, α_j を用いて j について昇順に求められる。次に、$U\boldsymbol{x} = \boldsymbol{y}$ は、

$$\begin{pmatrix} 1 & \beta_1 & 0 & \cdots & 0 \\ 0 & 1 & \beta_2 & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 0 & 1 & \beta_{N-1} \\ 0 & \cdots & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_1^{n+1} \\ u_2^{n+1} \\ \vdots \\ u_N^{n+1} \end{pmatrix} = \begin{pmatrix} u_1^{n+1} + \beta_1 u_2^{n+1} \\ u_2^{n+1} + \beta_2 u_3^{n+1} \\ \vdots \\ u_N^{n+1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

であるため、u_j^{n+1} は y_j, β_j を用いて j について降順に求められる。

3 課題

3.1 課題 1 拡散方程式の数値解

式 (2.3) を $x \in [0, L = 10]$ について、初期条件 $u_0(x) = \dfrac{1}{\sqrt{2\pi}} e^{\frac{1}{2}(x-5)^2}$ のもとで解く。以下の 4 つの場合を考える。

- オイラー陽解法、Dirichlet 境界条件
- オイラー陽解法、Neumann 境界条件
- クランク-ニコルソン法、Dirichlet 境界条件
- クランク-ニコルソン法、Neumann 境界条件

作成したソースコードの中で、全ての場合において用いた部分をコード 1, 2 に示す。

コード 1　solver.h

```
#include <iostream>
#include <vector>

class Solver {
protected:
    double dt, dx;
    int N;

    std::vector<double> u;

    void print_u(
        const double t
    ) {
        std::cout << t;
        for (int i = 0; i <= this->N + 1; i++) {
            std::cout << "␣" << this->u[i];
        }
        std::cout << std::endl;
    }

    void set_initial_values(
        const double (*u0)(const double)
    ) {
        for (int i = 1; i <= Solver::N; i++) {
            Solver::u[i] = (u0((i - 1) * Solver::dx) + u0(i * Solver::dx)) / 2.0;
        }
        this->set_condition_values();

        this->print_u(0);
    }

    virtual void set_condition_values() = 0;

public:
    Solver(
        const double dt,
        const double dx,
        const int N
    ): u(N + 2) {
        this->dt = dt;
        this->dx = dx;
        this->N = N;
    }

    virtual ~Solver() {}

    virtual void solve(
        const double t_end
    ) = 0;
};
```

コード 2　$u_0(x)$ の実装

```
#include <cmath>
```

```
2
3 const double u0(const double x) {
4     return 1.0 / sqrt(2.0 * M_PI) * exp(-(x - 5.0) * (x - 5.0) / 2.0);
5 }
```

3.1.1 オイラー陽解法、Dirichlet 境界条件

オイラー陽解法を用いて、Dirichlet 境界条件 $u_L = u_R = 0$ のもとで解く。定数は、$\Delta t = 0.01, \Delta x = 0.5, N = 20$ とした。作成したソースコードをコード 3 に示す。

コード 3　オイラー陽解法を用いて Dirichlet 境界条件のもとで解くソースコード

```
1  #include <iomanip>
2  #include <iterator>
3
4  #include "solver.h"
5
6
7  class EulerSolverWithDirichletCondition: Solver {
8  private:
9      double uL, uR;
10
11     void set_condition_values() override {
12         Solver::u[0] = this->uL;
13         Solver::u[Solver::N + 1] = this->uR;
14     }
15
16 public:
17     EulerSolverWithDirichletCondition(
18         const double dt,
19         const double dx,
20         const double N,
21
22         const double (*u0)(const double),
23         const double uL,
24         const double uR
25     ): Solver(dt, dx, N) {
26         this->uL = uL;
27         this->uR = uR;
28
29         this->set_initial_values(u0);
30     }
31
32     void solve(const double t_end) override {
33         for (int n = 1; n * Solver::dt <= t_end; n++) {
34             std::vector<double> u_;
35             std::copy(Solver::u.begin(), Solver::u.end(), std::back_inserter(u_));
36
37             for (int i = 1; i <= Solver::N; i++) {
38                 Solver::u[i] += (u_[i - 1] - 2 * u_[i] + u_[i + 1]) * Solver::dt / Solver::
                        dx / Solver::dx;
39             }
40             this->set_condition_values();
41
42             Solver::print_u(n * Solver::dt);
43         }
44     }
```

```
45 };
46
47
48 int main() {
49     std::cout << std::fixed << std::setprecision(16);
50
51     EulerSolverWithDirichletCondition solver = EulerSolverWithDirichletCondition(0.01, 0.5,
           20, u0, 0, 0);
52     solver.solve(5);
53 }
```

$t = 1, 2, 3, 4, 5$ における数値解を表 1, 図 1 にまとめる。

表 1　オイラー陽解法を用いて Dirichlet 境界条件のもとで解いた解

x	$u(x,1)$	$u(x,2)$	$u(x,3)$	$u(x,4)$	$u(x,5)$
0.25	0.004731	0.012185	0.015810	0.016703	0.016305
0.75	0.011554	0.025763	0.032106	0.033423	0.032422
1.25	0.022665	0.041850	0.049208	0.050098	0.048135
1.75	0.040201	0.061026	0.067131	0.066534	0.063170
2.25	0.065673	0.083109	0.085503	0.082365	0.077192
2.75	0.099036	0.107029	0.103559	0.097067	0.089810
3.25	0.137785	0.130870	0.120218	0.109999	0.100600
3.75	0.176680	0.152135	0.134230	0.120478	0.109143
4.25	0.208638	0.168194	0.144387	0.127875	0.115071
4.75	0.226757	0.176850	0.149730	0.131705	0.118108
5.25	0.226757	0.176850	0.149730	0.131705	0.118108
5.75	0.208638	0.168194	0.144387	0.127875	0.115071
6.25	0.176680	0.152135	0.134230	0.120478	0.109143
6.75	0.137785	0.130870	0.120218	0.109999	0.100600
7.25	0.099036	0.107029	0.103559	0.097067	0.089810
7.75	0.065673	0.083109	0.085503	0.082365	0.077192
8.25	0.040201	0.061026	0.067131	0.066534	0.063170
8.75	0.022665	0.041850	0.049208	0.050098	0.048135
9.25	0.011554	0.025763	0.032106	0.033423	0.032422
9.75	0.004731	0.012185	0.015810	0.016703	0.016305

3.1.2　オイラー陽解法、Neumann 境界条件

オイラー陽解法を用いて、Neumann 境界条件 $J_L = J_R = 0$ のもとで解く。定数は、$\Delta t = 0.01, \Delta x = 0.5, N = 20$ とした。作成したソースコードをコード 4 に示す。

コード 4　オイラー陽解法を用いて Neumann 境界条件のもとで解くソースコード

```
1 #include <iomanip>
2 #include <iterator>
```

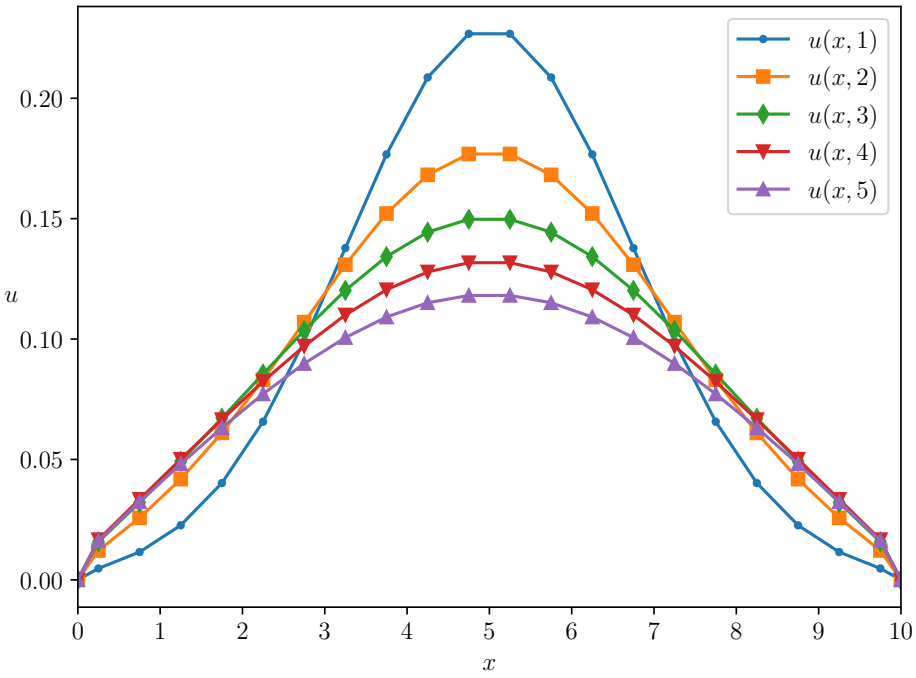

図 1　オイラー陽解法を用いて Dirichlet 境界条件のもとで解いた解

```
3
4  #include "solver.h"
5
6
7  class EulerSolverWithNeumannCondition: Solver {
8  private:
9      double JL, JR;
10
11     void set_condition_values() override {
12         Solver::u[0] = Solver::u[1] - this->JL * Solver::dx;
13         Solver::u[Solver::N + 1] = Solver::u[Solver::N] + this->JR * Solver::dx;
14     }
15
16 public:
17     EulerSolverWithNeumannCondition(
18         const double dt,
19         const double dx,
20         const double N,
21
22         const double (*u0)(const double),
23         const double JL,
24         const double JR
25     ): Solver(dt, dx, N) {
26         this->JL = JL;
27         this->JR = JR;
28
29         this->set_initial_values(u0);
30     }
```

```
31
32      void solve(const double t_end) override {
33          for (int n = 1; n * Solver::dt <= t_end; n++) {
34              std::vector<double> u_;
35              std::copy(Solver::u.begin(), Solver::u.end(), std::back_inserter(u_));
36
37              for (int i = 1; i <= Solver::N; i++) {
38                  Solver::u[i] += (u_[i - 1] - 2 * u_[i] + u_[i + 1]) * Solver::dt / Solver::
                        dx / Solver::dx;
39              }
40              this->set_condition_values();
41
42              Solver::print_u(n * Solver::dt);
43          }
44      }
45  };
46
47
48  int main() {
49      std::cout << std::fixed << std::setprecision(16);
50
51      EulerSolverWithNeumannCondition solver = EulerSolverWithNeumannCondition(0.01, 0.5, 20,
            u0, 0, 0);
52      solver.solve(5);
53  }
```

$t = 1, 2, 3, 4, 5$ における数値解を表 2, 図 2 にまとめる。

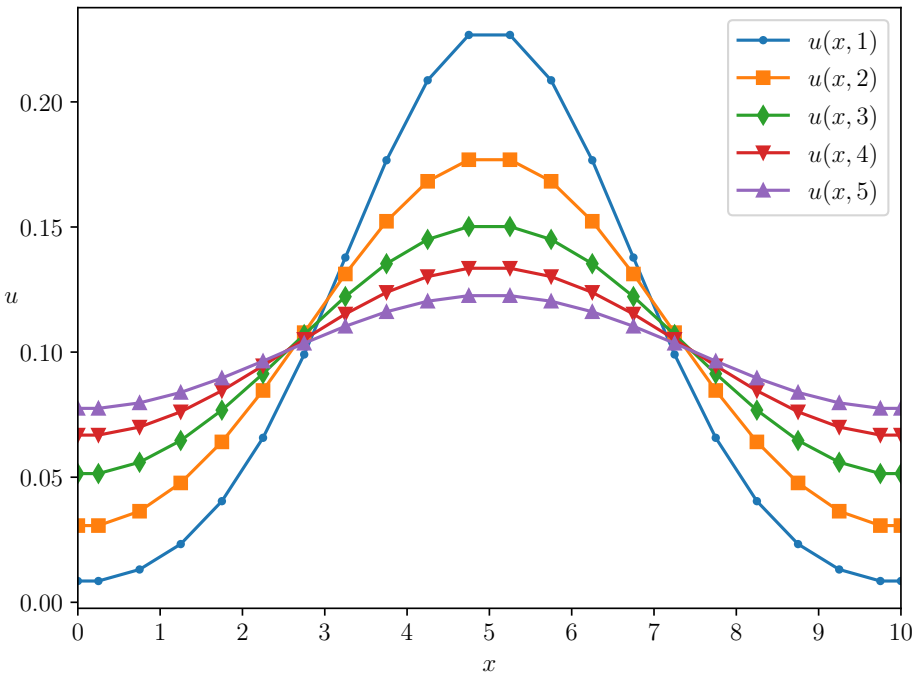

図 2　オイラー陽解法を用いて Neumann 境界条件のもとで解いた解

表 2　オイラー陽解法を用いて Neumann 境界条件のもとで解いた解

x	$u(x,1)$	$u(x,2)$	$u(x,3)$	$u(x,4)$	$u(x,5)$
0.25	0.008510	0.030667	0.051446	0.066829	0.077513
0.75	0.013120	0.036405	0.055969	0.070027	0.079704
1.25	0.023271	0.047708	0.064661	0.076128	0.083875
1.75	0.040421	0.064112	0.076811	0.084566	0.089624
2.25	0.065748	0.084665	0.091372	0.094545	0.096395
2.75	0.099060	0.107780	0.107008	0.105107	0.103530
3.25	0.137792	0.131219	0.122188	0.115219	0.110330
3.75	0.176682	0.152291	0.135339	0.123872	0.116125
4.25	0.208638	0.168264	0.145032	0.130188	0.120342
4.75	0.226757	0.176888	0.150174	0.133518	0.122561
5.25	0.226757	0.176888	0.150174	0.133518	0.122561
5.75	0.208638	0.168264	0.145032	0.130188	0.120342
6.25	0.176682	0.152291	0.135339	0.123872	0.116125
6.75	0.137792	0.131219	0.122188	0.115219	0.110330
7.25	0.099060	0.107780	0.107008	0.105107	0.103530
7.75	0.065748	0.084665	0.091372	0.094545	0.096395
8.25	0.040421	0.064112	0.076811	0.084566	0.089624
8.75	0.023271	0.047708	0.064661	0.076128	0.083875
9.25	0.013120	0.036405	0.055969	0.070027	0.079704
9.75	0.008510	0.030667	0.051446	0.066829	0.077513

3.1.3　クランク-ニコルソン法、Dirichlet 境界条件

オイラー陽解法を用いて、Dirichlet 境界条件 $u_L = u_R = 0$ のもとで解く。定数は、$\Delta t = 0.01, \Delta x = 0.05, N = 200$ とした。作成したソースコードをコード 5 に示す。

コード 5　クランク-ニコルソン法を用いて Dirichlet 境界条件のもとで解くソースコード

```
#include <iomanip>

#include "solver.h"


class CrankSolverWithDirichletCondition: Solver {
private:
    double c, uL, uR;

    void set_condition_values() override {
        Solver::u[0] = this->uL;
        Solver::u[Solver::N + 1] = this->uR;
    }

    const std::vector<std::vector<double>> calculate_A() {
```

```
16          std::vector<std::vector<double>> A(Solver::N + 2, std::vector<double>(Solver::N +
                2, 0));
17          for (int i = 1; i <= Solver::N; i++) {
18              A[i][i - 1] = -this->c / 2.0;
19              A[i][i] = 1 + this->c;
20              A[i][i + 1] = -this->c / 2.0;
21          }
22
23          return A;
24      }
25
26      const std::pair<std::vector<double>, std::vector<double>> LU_decomposition(
27          const std::vector<std::vector<double>> A
28      ) {
29          std::vector<double> alpha(N + 1, 0), beta(N + 1, 0);
30          for (int i = 1; i <= Solver::N; i++) {
31              alpha[i] = A[i][i] - A[i][i - 1] * beta[i - 1];
32              beta[i] = A[i][i + 1] / alpha[i];
33          }
34
35          return std::make_pair(alpha, beta);
36      }
37
38      const std::vector<double> calculate_z(
39          const std::vector<double> alpha,
40          const std::vector<double> beta
41      ) {
42          std::vector<double> z(N + 1);
43          z[1] = (1 - this->c) * Solver::u[1] + this->c * this->uL + this->c / 2.0 * Solver::
                u[2];
44          for (int i = 2; i <= Solver::N - 1; i++) {
45              z[i] = (1 - this->c) * Solver::u[i] + this->c / 2.0 * (Solver::u[i - 1] +
                    Solver::u[i + 1]);
46          }
47          z[N] = (1 - this->c) * Solver::u[N] + this->c * this->uR + this->c / 2.0 * Solver::
                u[N - 1];
48
49          return z;
50      }
51
52      const std::vector<double> calculate_y(
53          const std::vector<double> alpha,
54          const std::vector<std::vector<double>> A,
55          const std::vector<double> z
56      ) {
57          std::vector<double> y(N + 1, 0);
58          for (int i = 1; i <= Solver::N; i++) {
59              y[i] = (z[i] - A[i][i - 1] * y[i - 1]) / alpha[i];
60          }
61
62          return y;
63      }
64
65      const std::vector<double> calculate_x(
66          const std::vector<double> beta,
67          const std::vector<double> y
68      ) {
```

```
69          std::vector<double> x(N + 2, 0);
70          for (int i = N; i >= 1; i--) {
71              x[i] = y[i] - beta[i] * x[i + 1];
72          }
73
74          return x;
75      }
76
77  public:
78      CrankSolverWithDirichletCondition(
79          const double dt,
80          const double dx,
81          const double N,
82
83          const double (*u0)(const double),
84          const double uL,
85          const double uR
86      ): Solver(dt, dx, N) {
87          c = Solver::dt / Solver::dx / Solver::dx;
88
89          this->uL = uL;
90          this->uR = uR;
91
92          this->set_initial_values(u0);
93      }
94
95      void solve(const double t_end) override {
96          const std::vector<std::vector<double>> A = this->calculate_A();
97
98          std::vector<double> alpha, beta;
99          std::tie(alpha, beta) = this->LU_decomposition(A);
100
101         for (int n = 1; n * Solver::dt <= t_end; n++) {
102             const std::vector<double> z = this->calculate_z(alpha, beta);
103             const std::vector<double> y = this->calculate_y(alpha, A, z);
104             const std::vector<double> x = this->calculate_x(beta, y);
105
106             for (int i = 1; i <= N; i++) {
107                 Solver::u[i] = x[i];
108             }
109
110             this->set_condition_values();
111
112             Solver::print_u(n * Solver::dt);
113         }
114     }
115 };
116
117
118 int main() {
119     std::cout << std::fixed << std::setprecision(16);
120
121     CrankSolverWithDirichletCondition solver = CrankSolverWithDirichletCondition(0.01,
            0.05, 200, u0, 0, 0);
122     solver.solve(5);
123 }
```

$t = 1, 2, 3, 4, 5$ における数値解を表 3, 図 3 にまとめる。

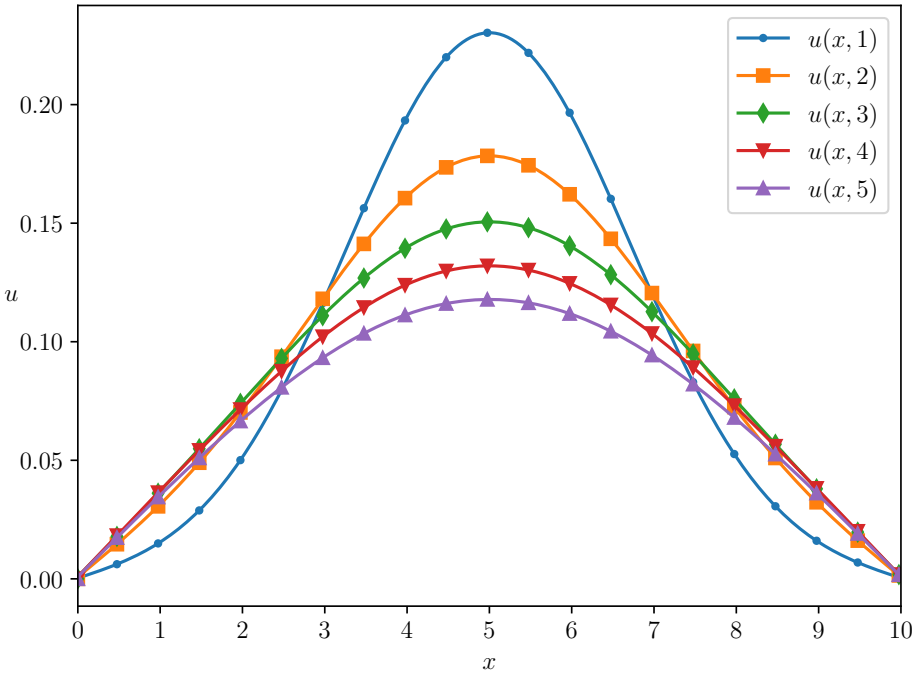

図 3　クランク-ニコルソン法を用いて Dirichlet 境界条件のもとで解いた解

3.1.4　クランク-ニコルソン法、Neumann 境界条件

オイラー陽解法を用いて、Neumann 境界条件 $J_L = J_R = 0$ のもとで解く。定数は、$\Delta t = 0.01, \Delta x = 0.05, N = 200$ とした。作成したソースコードをコード 6 に示す。

コード 6　クランク-ニコルソン法を用いて Neumann 境界条件のもとで解くソースコード

```
#include <iomanip>

#include "solver.h"


class CrankSolverWithNeumannCondition: Solver {
private:
    double c, JL, JR;

    void set_condition_values() override {
        Solver::u[0] = Solver::u[1] - this->JL * Solver::dx;
        Solver::u[Solver::N + 1] = Solver::u[Solver::N] + this->JR * Solver::dx;
    }

    const std::vector<std::vector<double>> calculate_A() {
        std::vector<std::vector<double>> A(Solver::N + 2, std::vector<double>(Solver::N +
            2, 0));
        for (int i = 1; i <= Solver::N; i++) {
            A[i][i - 1] = -this->c / 2.0;
```

```
19              A[i][i] = 1 + this->c;
20              A[i][i + 1] = -this->c / 2.0;
21          }
22          A[1][1] -= this->c / 2.0;
23          A[N][N] -= this->c / 2.0;
24
25          return A;
26      }
27
28      const std::pair<std::vector<double>, std::vector<double>> LU_decomposition(
29          const std::vector<std::vector<double>> A
30      ) {
31          std::vector<double> alpha(N + 1, 0), beta(N + 1, 0);
32          for (int i = 1; i <= Solver::N; i++) {
33              alpha[i] = A[i][i] - A[i][i - 1] * beta[i - 1];
34              beta[i] = A[i][i + 1] / alpha[i];
35          }
36
37          return std::make_pair(alpha, beta);
38      }
39
40      const std::vector<double> calculate_z(
41          const std::vector<double> alpha,
42          const std::vector<double> beta
43      ) {
44          std::vector<double> z(N + 1);
45          z[1] = (1 - this->c / 2.0) * Solver::u[1] - this->c * this->JL * Solver::dx + this
                ->c / 2.0 * Solver::u[2];
46          for (int i = 2; i <= Solver::N - 1; i++) {
47              z[i] = (1 - this->c) * Solver::u[i] + this->c / 2.0 * (Solver::u[i - 1] +
                    Solver::u[i + 1]);
48          }
49          z[N] = (1 - this->c / 2.0) * Solver::u[N] + this->c * this->JR * Solver::dx + this
                ->c / 2.0 * Solver::u[N - 1];
50
51          return z;
52      }
53
54      const std::vector<double> calculate_y(
55          const std::vector<double> alpha,
56          const std::vector<std::vector<double>> A,
57          const std::vector<double> z
58      ) {
59          std::vector<double> y(N + 1, 0);
60          for (int i = 1; i <= Solver::N; i++) {
61              y[i] = (z[i] - A[i][i - 1] * y[i - 1]) / alpha[i];
62          }
63
64          return y;
65      }
66
67      const std::vector<double> calculate_x(
68          const std::vector<double> beta,
69          const std::vector<double> y
70      ) {
71          std::vector<double> x(N + 2, 0);
72          for (int i = N; i >= 1; i--) {
```

```
            x[i] = y[i] - beta[i] * x[i + 1];
        }

        return x;
    }

public:
    CrankSolverWithNeumannCondition(
        const double dt,
        const double dx,
        const double N,

        const double (*u0)(const double),
        const double JL,
        const double JR
    ): Solver(dt, dx, N) {
        c = Solver::dt / Solver::dx / Solver::dx;

        this->JL = JL;
        this->JR = JR;

        this->set_initial_values(u0);
    }

    void solve(const double t_end) override {
        const std::vector<std::vector<double>> A = this->calculate_A();

        std::vector<double> alpha, beta;
        std::tie(alpha, beta) = this->LU_decomposition(A);

        for (int n = 1; n * Solver::dt <= t_end; n++) {
            const std::vector<double> z = this->calculate_z(alpha, beta);
            const std::vector<double> y = this->calculate_y(alpha, A, z);
            const std::vector<double> x = this->calculate_x(beta, y);

            for (int i = 1; i <= N; i++) {
                Solver::u[i] = x[i];
            }

            this->set_condition_values();

            Solver::print_u(n * Solver::dt);
        }
    }
};


int main() {
    std::cout << std::fixed << std::setprecision(16);

    CrankSolverWithNeumannCondition solver = CrankSolverWithNeumannCondition(0.01, 0.05,
        200, u0, 0, 0);
    solver.solve(5);
}
```

$t = 1, 2, 3, 4, 5$ における数値解を表 4, 図 4 にまとめる。

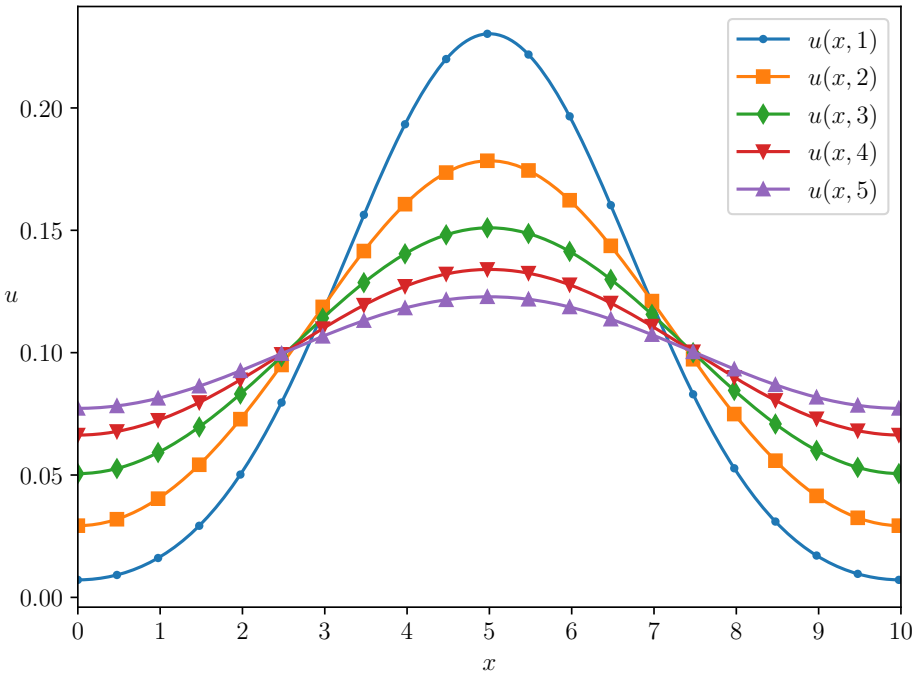

図 4　クランク-ニコルソン法を用いて Neumann 境界条件のもとで解いた解

3.2　課題 2 Fisher 方程式

式 (2.3) に非線形項 $f(u) = u(1-u)$ を加えた式を Fisher 方程式と呼ぶ。

$$\frac{\partial u}{\partial t} = u(1-u) + \frac{\partial^2 u}{\partial x^2} \tag{3.1}$$

これは、式 (3.2) に示すオイラー陽解法で解くことができる。

$$\begin{aligned} \frac{u_j^{n+1} - u_j^n}{\Delta t} &= f(u_j^n) + \frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2} \\ \rightarrow u_j^{n+1} &= u_j^n + \Delta t \left\{ f(u_j^n) + \frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2} \right\} \end{aligned} \tag{3.2}$$

式 (3.1) を、$x \in [0, L = 200]$ について、初期条件

$$u_0(x) = \frac{1}{(1 + e^{bx-5})^2} \ (b = 0.25, 0.5, 1)$$

のもとで解く。境界条件は Dirichlet 境界条件

$$\begin{aligned} u(0,t) &= 1 \\ u(L,t) &= 0 \end{aligned}$$

を用い、各定数は $\Delta t = 0.001, \Delta x = 0.05, N = 4000$ とした。

3.2.1 解法

作成したソースコードをコード 7, 8 に示す。なお、コード 1 に示した solver.h を使用している。

コード 7　式 (3.2) の実装

```
#include <iomanip>
#include <iterator>

#include "solver.h"


double b;


class EulerSolverWithDirichletCondition: Solver {
private:
    double uL, uR;

    void set_condition_values() override {
        Solver::u[0] = this->uL;
        Solver::u[Solver::N + 1] = this->uR;
    }

public:
    EulerSolverWithDirichletCondition(
        const double dt,
        const double dx,
        const double N,

        const double (*u0)(const double),
        const double uL,
        const double uR
    ): Solver(dt, dx, N) {
        this->uL = uL;
        this->uR = uR;

        this->set_initial_values(u0);
    }

    void solve(const double t_end) override {
        for (int n = 1; n * Solver::dt <= t_end; n++) {
            std::vector<double> u_;
            std::copy(Solver::u.begin(), Solver::u.end(), std::back_inserter(u_));

            for (int i = 1; i <= Solver::N; i++) {
                Solver::u[i] += Solver::dt * (u_[i] * (1 - u_[i]) + (u_[i - 1] - 2 * u_[i] +
                     u_[i + 1]) / Solver::dx / Solver::dx);
            }
            this->set_condition_values();

            Solver::print_u(n * Solver::dt);
        }
    }
};


```

```
int main(const int argc, const char *argv[]) {
    if (argc != 2) {
        return EXIT_FAILURE;
    }
    b = atof(argv[1]);

    std::cout << std::fixed << std::setprecision(16);

    EulerSolverWithDirichletCondition solver = EulerSolverWithDirichletCondition(0.001,
        0.05, 4000, u0, 1, 0);
    solver.solve(40);
}
```

コード 8　$u_0(x)$ の実装

```
#include <cmath>

const double u0(const double x) {
    return 1.0 / (1.0 + exp(b * x - 5.0)) / (1.0 + exp(b * x - 5.0));
}
```

3.2.2　結果

$t = 10, 20, 30, 40$ 数値解を表 5, 6, 7, 図 5, 6, 7 にまとめる。

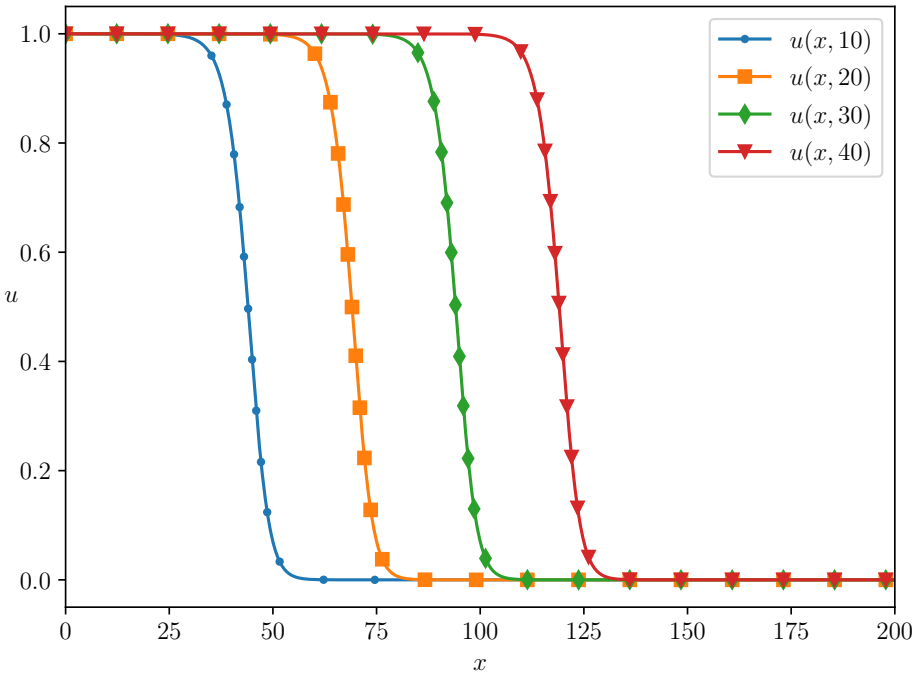

図 5　Fisher 方程式の数値解 ($b = 0.25$)

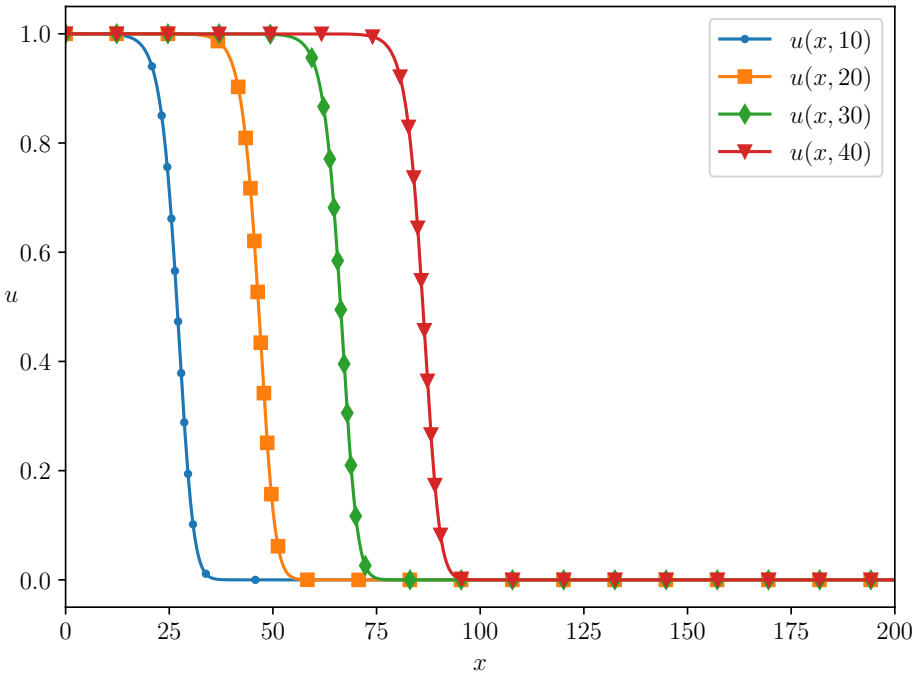

図 6　Fisher 方程式の数値解 ($b = 0.5$)

3.2.3　考察

図 5, 6, 7 を比較すると、b が大きくなるにつれ、ある時刻で u が 1 から 0 へと変化する x が小さくなっていることがわかる。この様子を図 8 に示す。これは、初期条件 $u_0(x)$ に現れている特徴でもあるため、当然であると言える。

3.3　課題 3 1 次元調和振動子の Schrödinger 方程式

式 (3.3) に Schrödinger 方程式を示す。

$$i\hbar\frac{\partial\psi}{\partial t}(x,t) = \left(-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + \frac{k}{2}x^2\right)\psi(x,t) \tag{3.3}$$

$\psi(x,t) \in \mathbb{C}$ を $\psi(x,t) = \psi_R(x,t) + i\psi_I(x,t)$ $(\psi_R, \psi_I \in \mathbb{R})$ のように実部と虚部に分解すると、式 (3.3) は

$$\hbar\frac{\partial\psi_R}{\partial t}(x,t) = \left(-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + \frac{k}{2}x^2\right)\psi_I(x,t)$$
$$-\hbar\frac{\partial\psi_I}{\partial t}(x,t) = \left(-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + \frac{k}{2}x^2\right)\psi_R(x,t)$$

と書き換えられる。このとき、粒子の位置の確率密度は $\psi_R^2 + \psi_I^2$ で示される。

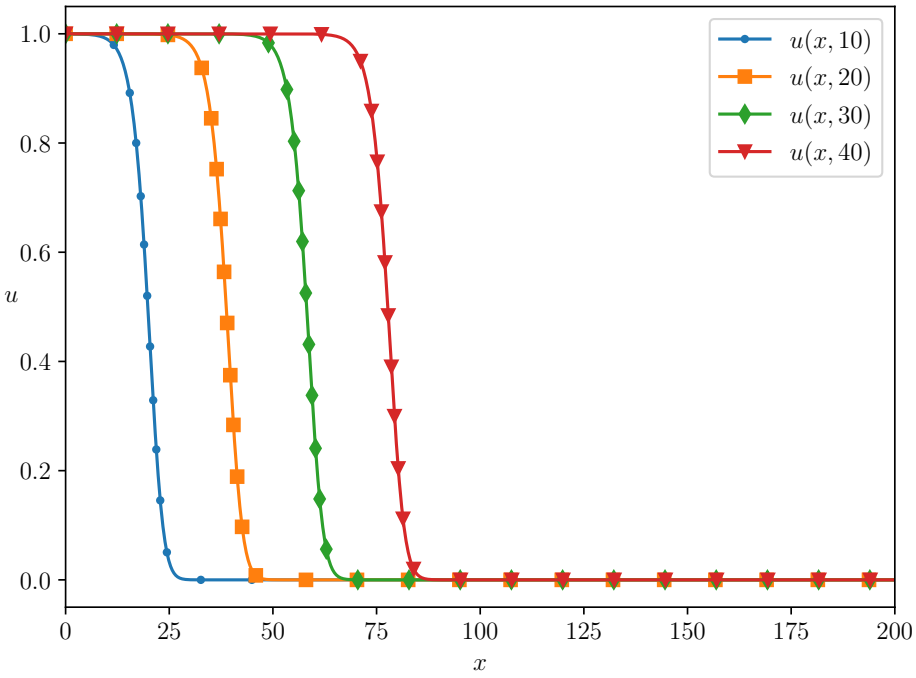

図 7　Fisher 方程式の数値解 ($b = 1$)

3.3.1　数値解法

$h = m = k = 1$ とした方程式

$$\frac{\partial \psi_R}{\partial t} = \left(-\frac{1}{2}\frac{\partial^2}{\partial x^2} + \frac{1}{2}x^2\right)\psi_I$$
$$-\frac{\partial \psi_I}{\partial t} = \left(-\frac{1}{2}\frac{\partial^2}{\partial x^2} + \frac{1}{2}x^2\right)\psi_R$$

は次に示すオイラー陽解法を用いて解くことができる。

$$R_j^{n+1} = R_j^n + \Delta t\left(-\frac{1}{2}\frac{I_{j-1}^n - 2I_j^n + I_{j+1}^n}{\Delta x^2} + \frac{1}{2}x_j^2 I_j^n\right) \quad (\psi_R(x, n\Delta t) \coloneqq \{R_j^n\})$$
$$I_j^{n+1} = I_j^n - \Delta t\left(-\frac{1}{2}\frac{R_{j-1}^{n+1} - 2R_j^{n+1} + R_{j+1}^{n+1}}{\Delta x^2} + \frac{1}{2}x_j^2 R_j^{n+1}\right) \quad (\psi_I(x, n\Delta t) \coloneqq \{I_j^n\})$$
$$x_j \coloneqq \left(j - \frac{1}{2}\right)\Delta x - \frac{L}{2}$$

本実験では周期境界条件 $\psi\left(\frac{L}{2}, t\right) = \psi\left(-\frac{L}{2}, t\right)$ を用いて解く。これは、次のように表される。

$$R_0^n = R_N^n, R_{N+1}^n = R_1^n$$
$$I_0^n = I_N^n, I_{N+1}^n = I_1^n$$

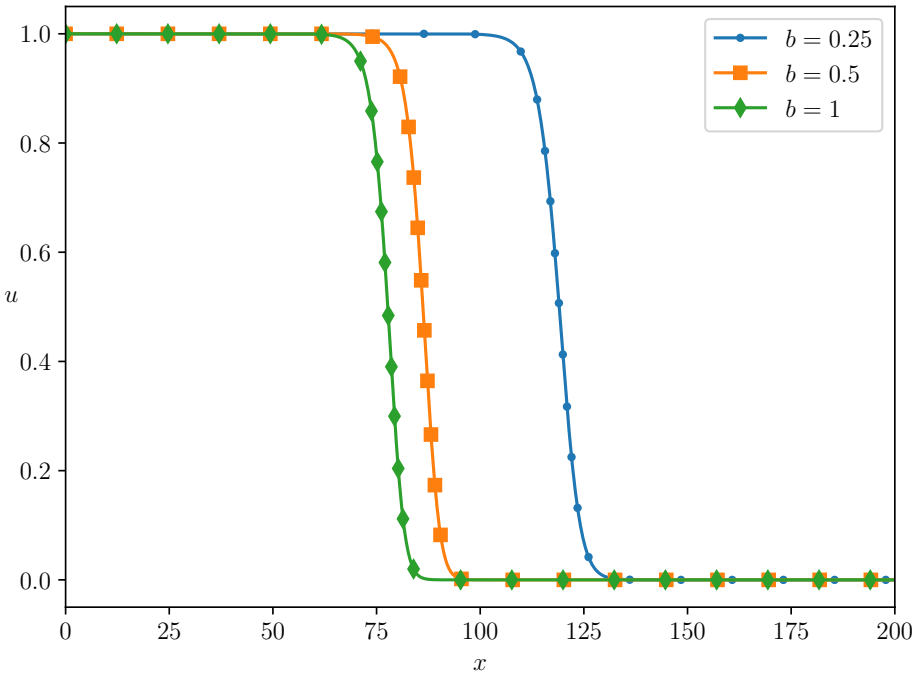

図 8　Fisher 方程式の数値解の比較 ($t = 40$)

初期条件は $\psi_0(x) = \dfrac{\sqrt{2}}{\pi^{\frac{1}{4}}} e^{-2(x-5)^2}$ とし、これは

$$R_j^0 = \frac{1}{2}\left\{\psi\left((j-1)\Delta x - \frac{L}{2}, 0\right) + \psi\left(j\Delta x - \frac{L}{2}, 0\right)\right\}$$
$$I_j^0 = -\Delta t\left(-\frac{1}{2}\frac{R_{j-1}^0 - 2R_j^0 + R_{j+1}^0}{\Delta x^2} + \frac{1}{2}x_j^2 R_j^0\right)$$

と書ける。

3.3.2　数値解の計算

$L = 20, x \in \left[-\dfrac{L}{2}, \dfrac{L}{2}\right]$ のもとで、定数 $\Delta t = 0.001, \Delta x = 0.05, N = 400$ として Schrödinger 方程式を解く。

作成したソースコードをコード 9, 10 に示す。

コード 9　オイラー陽解法による Schrödinger 方程式の解法

```
1 #include <iostream>
2 #include <vector>
3 #include <iomanip>
4 #include <iterator>
5
6
7 class SchroedingerSolver {
8 private:
9     double dt, dx, N;
```

```
10      std::vector<double> R, I;
11
12      void set_condition_values() {
13          this->R[0] = this->R[this->N];
14          this->R[this->N + 1] = this->R[1];
15          this->I[0] = this->I[this->N];
16          this->I[this->N + 1] = this->I[1];
17      }
18
19      void set_initial_values(
20          const double (*psi0)(const double)
21      ) {
22          for (int i = 1; i <= this->N; i++) {
23              this->R[i] = (psi0((i - 1) * this->dx - this->N * this->dx / 2.0) + psi0(i *
                    this->dx - this->N * this->dx / 2.0)) / 2.0;
24              const double xi = (i - 1.0 / 2.0) * this->dx - this->N * this->dx / 2.0;
25              this->I[i] = -this->dt * (-(this->R[i - 1] - 2.0 * this->R[i] + this->R[i + 1])
                     / this->dx / this->dx + xi * xi * this->R[i]) / 2.0;
26          }
27          this->set_condition_values();
28      }
29
30      void print_RI(
31          const double t,
32          const std::vector<double> I_
33      ) {
34          std::cout << t;
35          for (int i = 0; i <= this->N + 1; i++) {
36              std::cout << "␣" << this->R[i] * this->R[i] + this->I[i] * I_[i];
37          }
38          std::cout << std::endl;
39      }
40
41  public:
42      SchroedingerSolver(
43          const double dt,
44          const double dx,
45          const int N,
46
47          const double (*psi0)(const double)
48      ): R(N + 2, 0), I(N + 2, 0) {
49          this->dt = dt;
50          this->dx = dx;
51          this->N = N;
52
53          this->set_initial_values(psi0);
54      }
55
56      void solve(
57          const double t_end
58      ) {
59          for (int n = 1; n * this->dt <= t_end; n++) {
60              for (int i = 1; i <= this->N; i++) {
61                  const double xi = (i - 1.0 / 2.0) * this->dx - this->N * this->dx / 2.0;
62                  this->R[i] += this->dt * (-(this->I[i - 1] - 2.0 * this->I[i] + this->I[i +
                        1]) / this->dx / this->dx + xi * xi * this->I[i]) / 2.0;
63              }
```

```
64
65              std::vector<double> I_;
66              std::copy(this->I.begin(), this->I.end(), std::back_inserter(I_));
67              for (int i = 1; i <= this->N; i++) {
68                  const double xi = (i - 1.0 / 2.0) * this->dx - this->N * this->dx / 2.0;
69                  this->I[i] -= this->dt * (-(this->R[i - 1] - 2.0 * this->R[i] + this->R[i +
                        1]) / this->dx / this->dx + xi * xi * this->R[i]) / 2.0;
70              }
71
72              this->set_condition_values();
73
74              this->print_RI(n * this->dt, I_);
75          }
76      }
77  };
78
79
80  int main() {
81      std::cout << std::fixed << std::setprecision(16);
82
83      SchroedingerSolver solver = SchroedingerSolver(0.001, 0.05, 400, psi0);
84      solver.solve(8);
85  }
```

コード 10　$\psi_0(x)$ の実装

```
1  #include <cmath>
2
3  const double psi0(
4      const double x
5  ) {
6      return sqrt(2.0) * exp(-2.0 * (x - 5.0) * (x - 5.0)) / pow(M_PI, 1.0 / 4.0);
7  }
```

3.3.3　結果

$t = 1, 2, 3, 4, 5, 6, 7, 8$ における確率密度 $\left(R_j^n\right)^2 + I_j^{n-1} I_j^n$ の値を表 8, 9, 図 9 にまとめる。

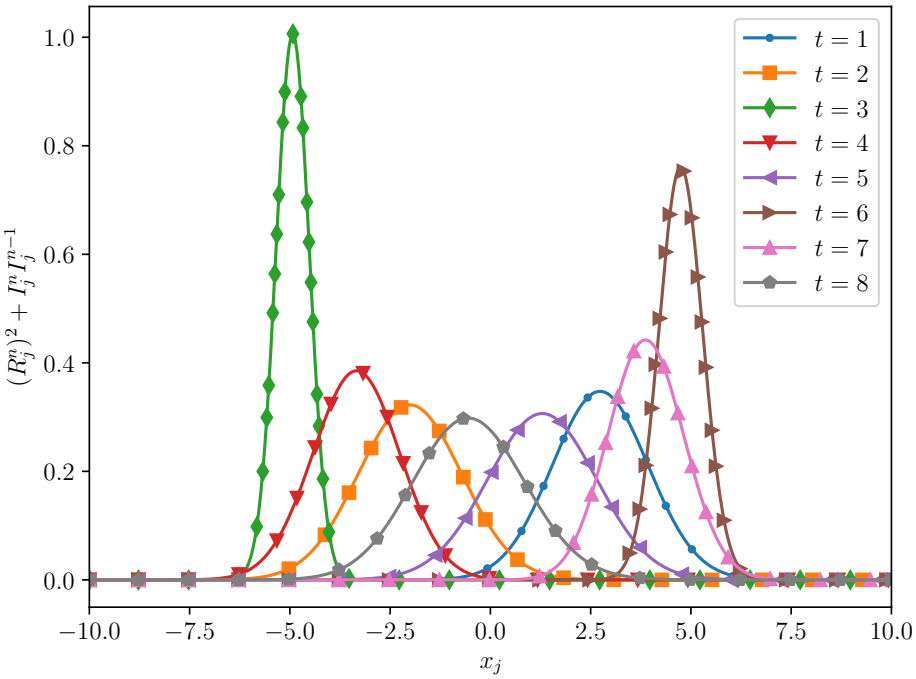

図 9　Schrödinger 方程式の解

参考文献

[1] 実験演習ワーキンググループ、“数理工学実験 2022 年度版”、京都大学工学部情報学科数理工学コース (2022)

表 3　クランク-ニコルソン法を用いて Dirichlet 境界条件のもとで解いた解

x	$u(x,1)$	$u(x,2)$	$u(x,3)$	$u(x,4)$	$u(x,5)$
0.025	0.000574	0.001436	0.001783	0.001826	0.001743
0.075	0.001151	0.002873	0.003566	0.003651	0.003486
0.125	0.001734	0.004313	0.005350	0.005476	0.005228
0.175	0.002324	0.005757	0.007135	0.007301	0.006970
0.225	0.002924	0.007207	0.008921	0.009126	0.008710
0.275	0.003537	0.008665	0.010710	0.010950	0.010449
0.325	0.004166	0.010131	0.012500	0.012773	0.012186
0.375	0.004813	0.011607	0.014294	0.014596	0.013921
0.425	0.005481	0.013095	0.016090	0.016418	0.015654
0.475	0.006172	0.014596	0.017889	0.018239	0.017385
⋮	⋮	⋮	⋮	⋮	⋮
4.775	0.228385	0.177493	0.150007	0.131605	0.117522
4.825	0.229148	0.177849	0.150226	0.131764	0.117649
4.875	0.229721	0.178116	0.150391	0.131883	0.117744
4.925	0.230105	0.178295	0.150501	0.131962	0.117808
4.975	0.230296	0.178384	0.150556	0.132002	0.117840
5.025	0.230296	0.178384	0.150556	0.132002	0.117840
5.075	0.230105	0.178295	0.150501	0.131962	0.117808
5.125	0.229721	0.178116	0.150391	0.131883	0.117744
5.175	0.229148	0.177849	0.150226	0.131764	0.117649
5.225	0.228385	0.177493	0.150007	0.131605	0.117522
⋮	⋮	⋮	⋮	⋮	⋮
9.525	0.006172	0.014596	0.017889	0.018239	0.017385
9.575	0.005481	0.013095	0.016090	0.016418	0.015654
9.625	0.004813	0.011607	0.014294	0.014596	0.013921
9.675	0.004166	0.010131	0.012500	0.012773	0.012186
9.725	0.003537	0.008665	0.010710	0.010950	0.010449
9.775	0.002924	0.007207	0.008921	0.009126	0.008710
9.825	0.002324	0.005757	0.007135	0.007301	0.006970
9.875	0.001734	0.004313	0.005350	0.005476	0.005228
9.925	0.001151	0.002873	0.003566	0.003651	0.003486
9.975	0.000574	0.001436	0.001783	0.001826	0.001743

表 4　クランク-ニコルソン法を用いて Neumann 境界条件のもとで解いた解

x	$u(x,1)$	$u(x,2)$	$u(x,3)$	$u(x,4)$	$u(x,5)$
0.025	0.007156	0.029303	0.050575	0.066323	0.077230
0.075	0.007200	0.029362	0.050621	0.066356	0.077253
0.125	0.007287	0.029479	0.050714	0.066421	0.077297
0.175	0.007419	0.029655	0.050853	0.066520	0.077364
0.225	0.007595	0.029889	0.051038	0.066650	0.077454
0.275	0.007815	0.030181	0.051270	0.066813	0.077565
0.325	0.008081	0.030532	0.051547	0.067009	0.077698
0.375	0.008393	0.030941	0.051870	0.067236	0.077854
0.425	0.008751	0.031409	0.052238	0.067496	0.078031
0.475	0.009158	0.031934	0.052652	0.067787	0.078230
⋮	⋮	⋮	⋮	⋮	⋮
4.775	0.228385	0.177527	0.150491	0.133665	0.122611
4.825	0.229148	0.177881	0.150701	0.133801	0.122702
4.875	0.229721	0.178148	0.150859	0.133903	0.122770
4.925	0.230105	0.178326	0.150964	0.133971	0.122815
4.975	0.230296	0.178415	0.151017	0.134005	0.122837
5.025	0.230296	0.178415	0.151017	0.134005	0.122837
5.075	0.230105	0.178326	0.150964	0.133971	0.122815
5.125	0.229721	0.178148	0.150859	0.133903	0.122770
5.175	0.229148	0.177881	0.150701	0.133801	0.122702
5.225	0.228385	0.177527	0.150491	0.133665	0.122611
⋮	⋮	⋮	⋮	⋮	⋮
9.525	0.009158	0.031934	0.052652	0.067787	0.078230
9.575	0.008751	0.031409	0.052238	0.067496	0.078031
9.625	0.008393	0.030941	0.051870	0.067236	0.077854
9.675	0.008081	0.030532	0.051547	0.067009	0.077698
9.725	0.007815	0.030181	0.051270	0.066813	0.077565
9.775	0.007595	0.029889	0.051038	0.066650	0.077454
9.825	0.007419	0.029655	0.050853	0.066520	0.077364
9.875	0.007287	0.029479	0.050714	0.066421	0.077297
9.925	0.007200	0.029362	0.050621	0.066356	0.077253
9.975	0.007156	0.029303	0.050575	0.066323	0.077230

表 5　Fisher 方程式の数値解 ($b = 0.25$)

x	$u(x, 10)$	$u(x, 20)$	$u(x, 30)$	$u(x, 40)$
0.025	1.000000	1.000000	1.000000	1.000000
0.075	1.000000	1.000000	1.000000	1.000000
0.125	1.000000	1.000000	1.000000	1.000000
0.175	1.000000	1.000000	1.000000	1.000000
0.225	1.000000	1.000000	1.000000	1.000000
0.275	1.000000	1.000000	1.000000	1.000000
0.325	1.000000	1.000000	1.000000	1.000000
0.375	1.000000	1.000000	1.000000	1.000000
0.425	1.000000	1.000000	1.000000	1.000000
0.475	1.000000	1.000000	1.000000	1.000000
⋮	⋮	⋮	⋮	⋮
99.775	0.000000	0.000000	0.077225	0.999005
99.825	0.000000	0.000000	0.075562	0.998987
99.875	0.000000	0.000000	0.073930	0.998969
99.925	0.000000	0.000000	0.072330	0.998951
99.975	0.000000	0.000000	0.070759	0.998932
100.025	0.000000	0.000000	0.069219	0.998914
100.075	0.000000	0.000000	0.067708	0.998894
100.125	0.000000	0.000000	0.066227	0.998875
100.175	0.000000	0.000000	0.064774	0.998855
100.225	0.000000	0.000000	0.063350	0.998835
⋮	⋮	⋮	⋮	⋮
199.525	0.000000	0.000000	0.000000	0.000000
199.575	0.000000	0.000000	0.000000	0.000000
199.625	0.000000	0.000000	0.000000	0.000000
199.675	0.000000	0.000000	0.000000	0.000000
199.725	0.000000	0.000000	0.000000	0.000000
199.775	0.000000	0.000000	0.000000	0.000000
199.825	0.000000	0.000000	0.000000	0.000000
199.875	0.000000	0.000000	0.000000	0.000000
199.925	0.000000	0.000000	0.000000	0.000000
199.975	0.000000	0.000000	0.000000	0.000000

表 6　Fisher 方程式の数値解 ($b = 0.5$)

x	$u(x, 10)$	$u(x, 20)$	$u(x, 30)$	$u(x, 40)$
0.025	1.000000	1.000000	1.000000	1.000000
0.075	0.999999	1.000000	1.000000	1.000000
0.125	0.999999	1.000000	1.000000	1.000000
0.175	0.999999	1.000000	1.000000	1.000000
0.225	0.999998	1.000000	1.000000	1.000000
0.275	0.999998	1.000000	1.000000	1.000000
0.325	0.999998	1.000000	1.000000	1.000000
0.375	0.999998	1.000000	1.000000	1.000000
0.425	0.999997	1.000000	1.000000	1.000000
0.475	0.999997	1.000000	1.000000	1.000000
⋮	⋮	⋮	⋮	⋮
99.775	0.000000	0.000000	0.000000	0.000037
99.825	0.000000	0.000000	0.000000	0.000036
99.875	0.000000	0.000000	0.000000	0.000034
99.925	0.000000	0.000000	0.000000	0.000033
99.975	0.000000	0.000000	0.000000	0.000031
100.025	0.000000	0.000000	0.000000	0.000030
100.075	0.000000	0.000000	0.000000	0.000028
100.125	0.000000	0.000000	0.000000	0.000027
100.175	0.000000	0.000000	0.000000	0.000026
100.225	0.000000	0.000000	0.000000	0.000025
⋮	⋮	⋮	⋮	⋮
199.525	0.000000	0.000000	0.000000	0.000000
199.575	0.000000	0.000000	0.000000	0.000000
199.625	0.000000	0.000000	0.000000	0.000000
199.675	0.000000	0.000000	0.000000	0.000000
199.725	0.000000	0.000000	0.000000	0.000000
199.775	0.000000	0.000000	0.000000	0.000000
199.825	0.000000	0.000000	0.000000	0.000000
199.875	0.000000	0.000000	0.000000	0.000000
199.925	0.000000	0.000000	0.000000	0.000000
199.975	0.000000	0.000000	0.000000	0.000000

表 7　Fisher 方程式の数値解 ($b = 1$)

x	$u(x, 10)$	$u(x, 20)$	$u(x, 30)$	$u(x, 40)$
0.025	0.999996	1.000000	1.000000	1.000000
0.075	0.999992	1.000000	1.000000	1.000000
0.125	0.999988	1.000000	1.000000	1.000000
0.175	0.999984	1.000000	1.000000	1.000000
0.225	0.999981	1.000000	1.000000	1.000000
0.275	0.999977	1.000000	1.000000	1.000000
0.325	0.999973	1.000000	1.000000	1.000000
0.375	0.999969	1.000000	1.000000	1.000000
0.425	0.999965	1.000000	1.000000	1.000000
0.475	0.999961	1.000000	1.000000	1.000000
⋮	⋮	⋮	⋮	⋮
99.775	0.000000	0.000000	0.000000	0.000000
99.825	0.000000	0.000000	0.000000	0.000000
99.875	0.000000	0.000000	0.000000	0.000000
99.925	0.000000	0.000000	0.000000	0.000000
99.975	0.000000	0.000000	0.000000	0.000000
100.025	0.000000	0.000000	0.000000	0.000000
100.075	0.000000	0.000000	0.000000	0.000000
100.125	0.000000	0.000000	0.000000	0.000000
100.175	0.000000	0.000000	0.000000	0.000000
100.225	0.000000	0.000000	0.000000	0.000000
⋮	⋮	⋮	⋮	⋮
199.525	0.000000	0.000000	0.000000	0.000000
199.575	0.000000	0.000000	0.000000	0.000000
199.625	0.000000	0.000000	0.000000	0.000000
199.675	0.000000	0.000000	0.000000	0.000000
199.725	0.000000	0.000000	0.000000	0.000000
199.775	0.000000	0.000000	0.000000	0.000000
199.825	0.000000	0.000000	0.000000	0.000000
199.875	0.000000	0.000000	0.000000	0.000000
199.925	0.000000	0.000000	0.000000	0.000000
199.975	0.000000	0.000000	0.000000	0.000000

表 8　$t = 1, 2, 3, 4$ における確率密度

x	$t = 1$	$t = 2$	$t = 3$	$t = 4$
-9.975	0.00000000	0.00000000	0.00000000	0.00000000
-9.925	0.00000000	0.00000000	0.00000000	0.00000000
-9.875	0.00000000	0.00000000	0.00000000	0.00000000
-9.825	0.00000000	0.00000000	0.00000000	0.00000000
-9.775	0.00000000	0.00000000	0.00000000	0.00000000
-9.725	0.00000000	0.00000000	0.00000000	0.00000000
-9.675	0.00000000	0.00000001	0.00000000	0.00000001
-9.625	0.00000000	0.00000001	0.00000000	0.00000001
-9.575	0.00000000	0.00000001	0.00000000	0.00000001
-9.525	0.00000000	0.00000001	0.00000000	0.00000002
⋮	⋮	⋮	⋮	⋮
-0.225	0.01556952	0.12424230	0.00000000	0.00506276
-0.175	0.01729531	0.11768819	0.00000000	0.00437881
-0.125	0.01917681	0.11131161	0.00000000	0.00377783
-0.075	0.02122364	0.10512166	0.00000000	0.00325118
-0.025	0.02344552	0.09912702	0.00000000	0.00279071
0.025	0.02585219	0.09333522	0.00000000	0.00238899
0.075	0.02845336	0.08775196	0.00000000	0.00203939
0.125	0.03125860	0.08238077	0.00000000	0.00173608
0.175	0.03427732	0.07722334	0.00000000	0.00147389
0.225	0.03751860	0.07228023	0.00000000	0.00124808
⋮	⋮	⋮	⋮	⋮
9.525	0.00000003	0.00000000	0.00000000	0.00000000
9.575	0.00000003	0.00000000	0.00000000	0.00000000
9.625	0.00000002	0.00000000	0.00000000	0.00000000
9.675	0.00000002	0.00000000	0.00000000	0.00000000
9.725	0.00000001	0.00000000	0.00000000	0.00000000
9.775	0.00000001	0.00000000	0.00000000	0.00000000
9.825	0.00000001	0.00000000	0.00000000	0.00000000
9.875	0.00000000	0.00000000	0.00000000	0.00000000
9.925	0.00000000	0.00000000	0.00000000	0.00000000
9.975	0.00000000	0.00000000	0.00000000	0.00000000

表 9　$t = 5, 6, 7, 8$ における確率密度

x	$t = 5$	$t = 6$	$t = 7$	$t = 8$
-9.975	0.00000000	0.00000000	0.00000000	0.00000000
-9.925	0.00000000	0.00000000	0.00000000	0.00000000
-9.875	0.00000000	0.00000000	0.00000000	0.00000000
-9.825	0.00000000	0.00000000	0.00000000	0.00000000
-9.775	0.00000000	0.00000000	0.00000000	0.00000000
-9.725	0.00000000	0.00000000	0.00000000	0.00000000
-9.675	0.00000000	0.00000000	0.00000000	0.00000000
-9.625	0.00000000	0.00000000	0.00000000	0.00000000
-9.575	0.00000000	0.00000000	0.00000000	0.00000000
-9.525	0.00000000	0.00000000	0.00000000	0.00000000
⋮	⋮	⋮	⋮	⋮
-0.225	0.16440913	0.00000000	0.00001070	0.29014157
-0.175	0.17114183	0.00000000	0.00001412	0.28751902
-0.125	0.17791635	0.00000000	0.00001863	0.28456163
-0.075	0.18471618	0.00000000	0.00002455	0.28127951
-0.025	0.19152308	0.00000000	0.00003225	0.27768268
0.025	0.19831737	0.00000000	0.00004212	0.27378247
0.075	0.20507862	0.00000000	0.00005461	0.26959268
0.125	0.21178655	0.00000000	0.00007037	0.26512997
0.175	0.21842148	0.00000000	0.00009024	0.26041298
0.225	0.22496421	0.00000000	0.00011535	0.25546090
⋮	⋮	⋮	⋮	⋮
9.525	0.00000000	0.00000000	0.00000000	0.00000000
9.575	0.00000000	0.00000000	0.00000000	0.00000000
9.625	0.00000000	0.00000000	0.00000000	0.00000000
9.675	0.00000000	0.00000000	0.00000000	0.00000000
9.725	0.00000000	0.00000000	0.00000000	0.00000000
9.775	0.00000000	0.00000000	0.00000000	0.00000000
9.825	0.00000000	0.00000000	0.00000000	0.00000000
9.875	0.00000000	0.00000000	0.00000000	0.00000000
9.925	0.00000000	0.00000000	0.00000000	0.00000000
9.975	0.00000000	0.00000000	0.00000000	0.00000000