数理工学実験
テーマ３　熱方程式の差分法

2021年11月8日 提出

工学部情報学科数理工学コース２年
1029-32-7314　岡本淳志

1　はじめに

今回は、熱方程式（拡散方程式）の差分法についての数値実験を行う。この章では、拡散方程式を数値的に解くための実験の準備を行う。

1.1　拡散方程式

拡散とは微粒子などが自発的に散らばり広がっていく現象である。1 次元空間内での拡散を考える。微粒子の濃度を $C(x,t)$ とする。微粒子の流れを $J(x,t)$ とすると、粒子数の保存を表す連続の式は

$$\frac{\partial}{\partial t}C(x,t) = -\frac{\partial}{\partial x}J(x,t) \tag{1}$$

とかける。また、流れ $J(x,t)$ は濃度勾配に比例すると仮定すると

$$J(x,t) = -D\frac{\partial}{\partial x}C(x,t) \tag{2}$$

と表される。これから拡散方程式

$$\frac{\partial}{\partial t}C(x,t) = D\frac{\partial^2}{\partial x^2}C(x,t) \tag{3}$$

以下では、$D = 1$ とした拡散方程式

$$\frac{\partial}{\partial t}u(x,t) = \frac{\partial^2}{\partial x^2}u(x,t) \tag{4}$$

を考える。

拡散方程式を解くためには初期条件

$$u(x,0) = u_0(x) \tag{5}$$

と境界 $x = 0, L$ における境界条件を考える必要がある。代表的な境界条件には

- Dirichlet 境界条件: 境界上での u の値が指定されている。

$$u(0,t) = u_L, \quad u(L,t) = u_R \tag{6}$$

- Neumann 境界条件: 境界上での u の微分の値が指定されている。

$$\frac{\partial u}{\partial x}(0,t) = J_L, \quad \frac{\partial u}{\partial x}(L,t) = J_R \tag{7}$$

の 2 つがある。

1.2 数値解法

ここでは、時間と空間を離散化して拡散方程式 (4) を数値的に解くことを考える。時間と空間の刻み幅をそれぞれ Δt, Δx とする ($L = N\Delta x$)。離散化された時空間上での平均濃度を

$$u_j^n \equiv \frac{1}{\Delta x}\int_{(j-i)\Delta x}^{j\Delta x} u(y, n\Delta t)dy \tag{8}$$

と書く。ただし、u_j^n は区間 $[(j\ -\ 1)\Delta x, j\Delta x]$ に対して割り当てられている。また、u_0^n, u_{N+1}^n は境界条件のために用いる。方程式 (1) を $[n\Delta t, (n+1)\Delta t]$ で積分すると

$$u(x,(n+1)\Delta t) - u(x, n\Delta t) = -\int_{n\Delta t}^{(n+1)\Delta t} \frac{\partial}{\partial x} J(x,s)ds \tag{9}$$

となる。(9) の右辺の積分を時刻 $n\Delta t$ の値で近似し、整理していくと、差分方程式

$$u_j^{n+1} - u_j^n \simeq \frac{\Delta t}{\Delta x^2}(u_{j-1}^n - 2u_j^n + u_{j+1}^n) \tag{10}$$

を得る。

一方、(9) において積分をより精度の良い台形公式で近似し、整理していくと

$$u_j^{n+1} - u_j^n \simeq \frac{1}{2}\frac{\Delta t}{\Delta x^2}(u_{j-1}^{n+1} - 2u_j^{n+1} + u_{j+1}^{n+1}) + \frac{1}{2}\frac{\Delta t}{\Delta x^2}(u_{j-1}^n - 2u_j^n + u_{j+1}^n) \tag{11}$$

を得る。

以下では

$$c \equiv \frac{\Delta t}{\Delta x^2} \tag{12}$$

と書くことにする。

1.3 オイラー陽解法

(10) のような最も単純な差分方程式による数値解法をオイラー陽解法と呼ぶ。(10) を書き直すと

$$u_j^{n+1} - u_j^n \simeq c(u_{j-1}^n - 2u_j^n + u_{j+1}^n) \tag{13}$$

となる。初期条件は $\left\{u_j^0\right\}_{j=1}^N$ と表される。また、境界条件は

- Dirichlet 境界条件

$$u_0^n = u_L, \quad u_{N+1}^n = u_R \tag{14}$$

- Neumann 境界条件

$$u_0^n = u_1^n - J_L\Delta x, \quad u_{N+1}^n = u_N^n + J_R\Delta x \tag{15}$$

である。

また、オイラー陽解法は $c \leq \frac{1}{2}$ について安定である。

1.4 クランク・ニコルソン法

(11) はクランク・ニコルソン法と呼ばれる。この方法はオイラー陽解法よりも精度が良くなっている。(11) を書き直すと

$$-\frac{c}{2}u_{j-1}^{n+1}+(1+c)u_j^{n+1}-\frac{c}{2}u_{j+1}^{n+1}=(1-c)u_j^n+\frac{c}{2}u_{j-1}^n+\frac{c}{2}u_{j+1}^n \qquad (16)$$

となる。ただし、境界条件は

- Dirichlet 境界条件: $u_0^n=u_L$, $u_{N+1}^n=u_R$ より

$$\begin{aligned}(1+c)u_1^{n+1}-\frac{c}{2}u_2^{n+1}&=(1-c)u_1^n+cu_L+\frac{c}{2}u_2^n,\\ -\frac{c}{2}u_{N-1}^{n+1}+(1+c)u_N^{n+1}&=(1-c)u_N^n+cu_R+\frac{c}{2}u_{N-1}^n\end{aligned} \qquad (17)$$

- Neumann 境界条件: $u_0^n=u_1^n-J_L\Delta x$, $u_{N+1}^n=u_N^n+J_R\Delta x$ より

$$\begin{aligned}(1+\frac{c}{2})u_1^{n+1}-\frac{c}{2}u_2^{n+1}&=(1-\frac{c}{2})u_1^n-cJ_L\Delta x+\frac{c}{2}u_2^n\\ -\frac{c}{2}u_{N-1}^{n+1}+(1+\frac{c}{2})u_N^{n+1}&=(1-\frac{c}{2})u_N^n+cJ_R\Delta x+\frac{c}{2}u_{N-1}^n\end{aligned} \qquad (18)$$

である。実験では、これらの連立方程式を解くために LU 分解を用いる。

また、クランク・ニコルソン法は全ての c について安定である。

2 問題１

拡散方程式 (4)($L=10$) を初期条件

$$u_0(x)=\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}(x-5)^2} \qquad (19)$$

の下で、以下の 4 つの場合で数値的に解く。

- オイラー陽解法で境界条件は Dirichlet 境界条件 $u_L=u_R=0$, 刻み幅は $\Delta t=0.01$, $\Delta x=0.5$
- オイラー陽解法で境界条件は Neumann 境界条件 $J_L=J_R=0$, 刻み幅は $\Delta t=0.01$, $\Delta x=0.5$
- クランク・ニコルソン法で境界条件は Dirichlet 境界条件 $u_L=u_R=0$, 刻み幅は $\Delta t=0.01$, $\Delta x=0.05$
- クランク・ニコルソン法で境界条件は Neumann 境界条件 $J_L=J_R=0$, 刻み幅は $\Delta t=0.01$, $\Delta x=0.05$

いずれの場合も $t = 1, 2, 3, 4, 5$ における $u(x, t)$ の値を出力する。得られた $u(x, t)$ の値をそれぞれ一つの図（横軸 x, 縦軸 u）に図示する。

今回は (8) のように u_j^n に区間 $[(j-1)\Delta x, j\Delta x]$ を割り当てるという離散化を行なっているので、初期条件を

$$u_j^0 = \frac{1}{2}[u_0((j-1)\Delta x) + u_0(j\Delta x)] \tag{20}$$

と考えた。また、$\left\{u_j^n\right\}_{j=1}^{N}$ をプロットする際、各 j に対応する座標として $x = (j - 1/2)\Delta x$ を採用した。

2.1 オイラー陽解法 Dirichlet 境界条件の場合

オイラー陽解法 (13) で境界条件は Dirichlet 境界条件 $u_L = u_R = 0$ を用いて数値的に解いた。刻み幅は $\Delta t = 0.01$, $\Delta x = 0.5$ とした。

用いたソースコードを付録のソースコード 1 に示す。

出力では、$x = 5$ を対称として、$0 \leq x < 5$ では u が増加、$5 < x \leq 10$ では u が減少していく様子がみられた。また、t が大きくなるにつれてその増減は小さくなっていく様子が見られた。

出力に基づき、図 1 を作成した。横軸は $0 \leq x \leq 10$、縦軸は $0 \leq u(x, t) \leq 0.25$ とした。

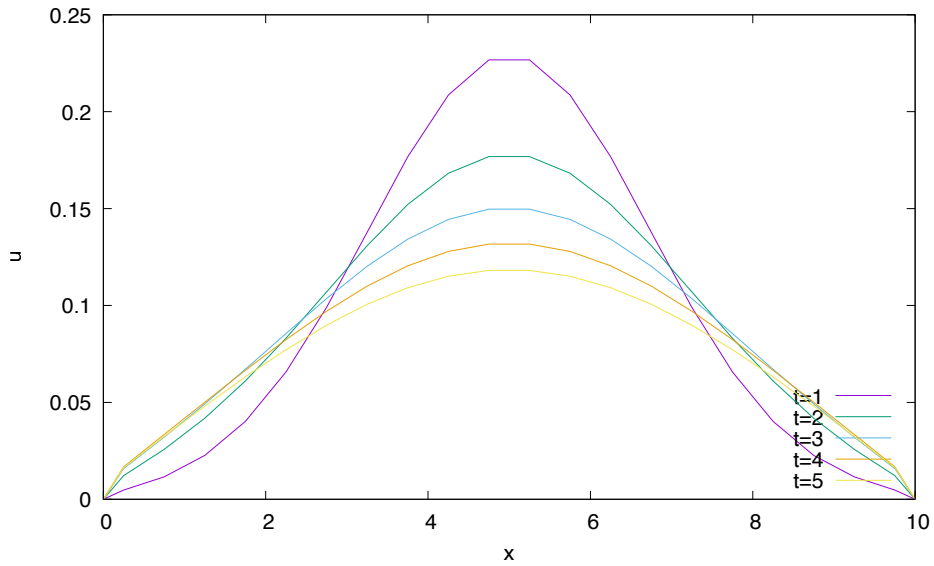

図 1: オイラー陽解法 (Dirichlet 境界条件) による拡散方程式 (4) の数値解 ($t = 1, 2, 3, 4, 5$)

2.2　オイラー陽解法 Neumann 境界条件の場合

オイラー陽解法 (13) で境界条件は Neumann 境界条件 $J_L = J_R = 0$ を用いて数値的に解いた。刻み幅は $\Delta t = 0.01$, $\Delta x = 0.5$ とした。

用いたソースコードを付録のソースコード 2 に示す。

出力では同様に、$x = 5$ を対称として、$0 \leq x < 5$ では u が増加、$5 < x \leq 10$ では u が減少していく様子がみられた。また、t が大きくなるにつれてその増減は小さくなっていく様子が見られた。

出力に基づき、図 2 を作成した。横軸は $0 \leq x \leq 10$、縦軸は $0 \leq u(x,t) \leq 0.25$ とした。

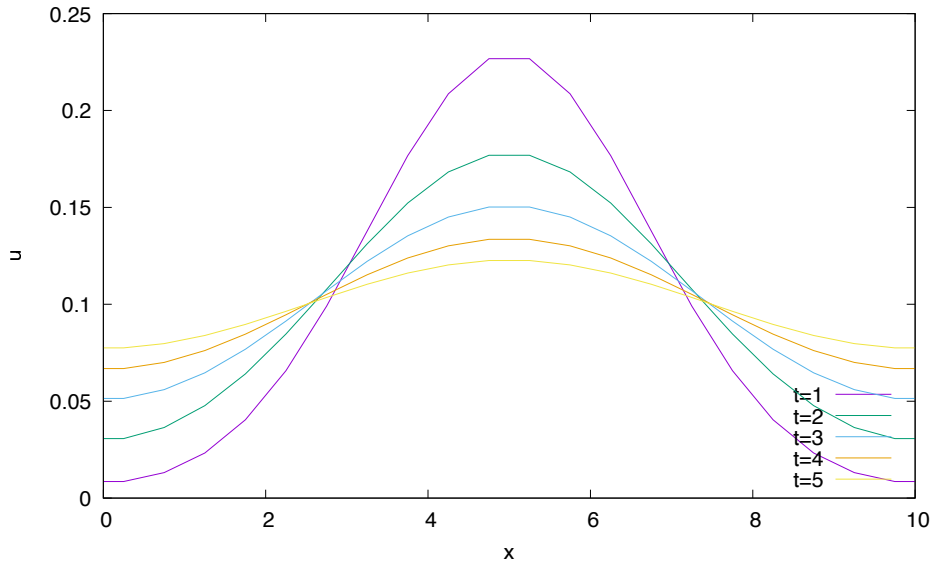

図 2: オイラー陽解法 (Neumann 境界条件) による拡散方程式 (4) の数値解 ($t = 1, 2, 3, 4, 5$)

2.3　クランク・ニコルソン法 Dirichlet 境界条件の場合

クランク・ニコルソン法 (16) で境界条件は Dirichlet 境界条件 $u_L = u_R = 0$ を用いて数値的に解いた。刻み幅は $\Delta t = 0.01$, $\Delta x = 0.05$ とした。(16) と境界条件 (17) を合わせた連立方程式は、LU 分解を用いて解いた。

用いたソースコードを付録のソースコード 3 に示す。

出力では同様に、$x = 5$ を対称として、$0 \leq x < 5$ では u が増加、$5 < x \leq 10$ では u が減少していく様子がみられた。また、t が大きくなるにつれてその増減は小さくなっていく様子が見られた。

出力に基づき、図 3 を作成した。横軸は $0 \leq x \leq 10$、縦軸は $0 \leq u(x,t) \leq 0.25$ とした。

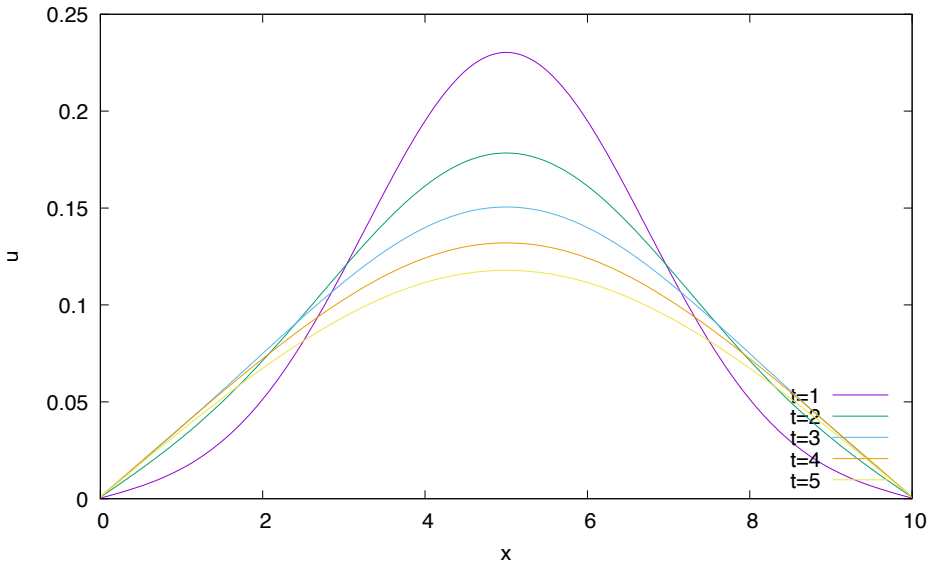

図 3: クランク・ニコルソン法 (Dirichlet 境界条件) による拡散方程式 (4) の数値解 ($t = 1, 2, 3, 4, 5$)

2.4 クランク・ニコルソン法 Neumann 境界条件の場合

クランク・ニコルソン法 (16) で境界条件は Neumann 境界条件 $J_L = J_R = 0$ を用いて数値的に解いた。刻み幅は $\Delta t = 0.01$, $\Delta x = 0.05$ とした。(16) と境界条件 (18) を合わせた連立方程式は、LU 分解を用いて解いた。

用いたソースコードを付録のソースコード 4 に示す。

出力では同様に、$x = 5$ を対称として、$0 \leq x < 5$ では u が増加、$5 < x \leq 10$ では u が減少していく様子がみられた。また、t が大きくなるにつれてその増減は小さくなっていく様子が見られた。

出力に基づき、図 4 を作成した。横軸は $0 \leq x \leq 10$、縦軸は $0 \leq u(x,t) \leq 0.25$ とした。

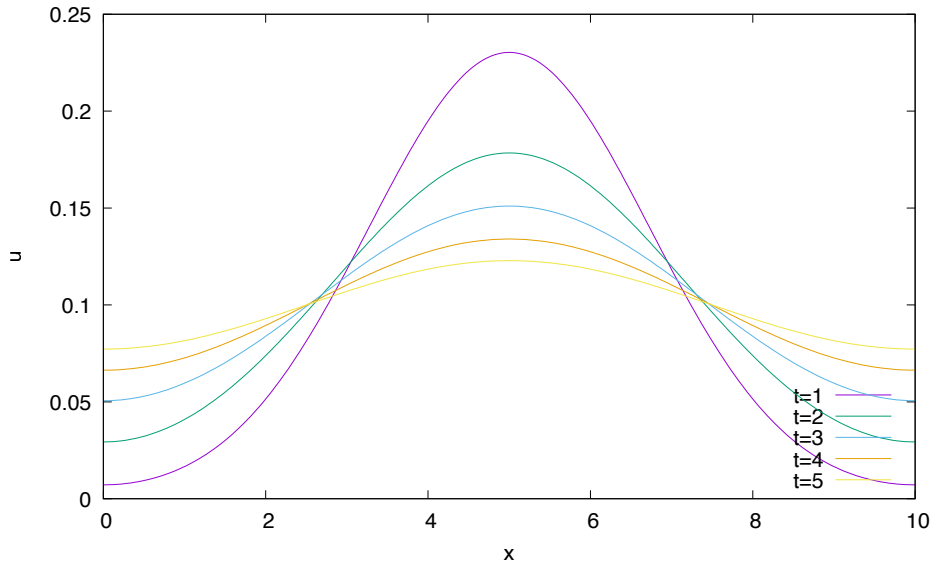

図 4: クランク・ニコルソン法 (Neumann 境界条件) による拡散方程式 (4) の数値解 ($t = 1, 2, 3, 4, 5$)

2.5　考察

これらの実験結果より、いずれの場合も、$x = 5$ を対称として、$0 \leq x < 5$ では u が増加、$5 < x \leq 10$ では u が減少していく様子がみられた。これは、初期条件が $x = 5$ で対称であることと、境界条件が $u(0, 0) = u(10, 0)$ であることによるものだと考えられる。t が大きくなるにつれてその増減は小さくなっていく様子が見られたことから、t が大きくなるにつれて分散が進み、直線へと近づいていくとかんがえられる。

Dirichlet 境界条件と Neumann 境界条件のいずれも、u_0, u_{N+1} の値（またそれに近い u の値）は境界条件に依るところが大きいことがわかった。

この方程式では、オイラー陽解法とクランク・ニコルソンの精度の違いについて明確には分からなかった。

3　問題２

拡散方程式に非線形項 $f(u) = u(1 - u)$ を加えた偏微分方程式（Fisher 方程式）

$$\frac{\partial u}{\partial t} = u(1 - u) + \frac{\partial^2 u}{\partial x^2} \tag{21}$$

を考える。初期条件

$$u_0(x) = \frac{1}{(1 + e^{bx-5})^2} \tag{22}$$

境界条件（L=200）

$$u(0,t) = 1, \quad u(L,t) = 0 \tag{23}$$

の下で $b = 0.25, 0.5, 1.0$ の場合に Fisher 方程式をオイラー陽解法

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = f(u_j^n) + \frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2} \tag{24}$$

を用いて数値的に解くことを考える。

3.1 実験方法

(24) は、

$$u_j^{n+1} = u_j^n + \Delta t\Big(f(u_j^n) + \frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2}\Big) \tag{25}$$

のように書き直すことができる。これを用いて、Fisher 方程式を数値的に解いた。ただし、刻み幅は $\Delta x = 0.05$, $\Delta t = 0.001$ とし、$t = 10, 20, 30, 40$ の $u(x,t)$ の値を出力した。

3.2 実験結果

用いたソースコードを付録のソースコード 5 に示す。

出力では、いずれの $b = 0.25, 0.5, 1.0$ の場合も、$x = 0$ からしばらく $u \approx 1.0$ の値を取り、そこから一気に u は減少、$u \approx 0$ の値を取るとそこからほぼ一定で $u \approx 0$ を取り続けた。t が大きくなるにつれて、u の急減少が始まる x の値は大きくなった。また、b が大きくなるにつれて、u の急減少が始まる x の値は小さくなった。

出力に基づき、b の値ごとに以下の図 5, 6, 7 を作成した。横軸は $0 \leq x \leq 200$、縦軸は $0 \leq u(x,t) \leq 1.0$ とした。ただし、問題 1 と同様に、初期条件を

$$u_j^0 = \frac{1}{2}[u_0((j-1)\Delta x) + u_0(j\Delta x)] \tag{26}$$

と考えた。また、$\left\{u_j^n\right\}_{j=1}^N$ をプロットする際、各 j に対応する座標として $x = (j - 1/2)\Delta x$ を採用した。

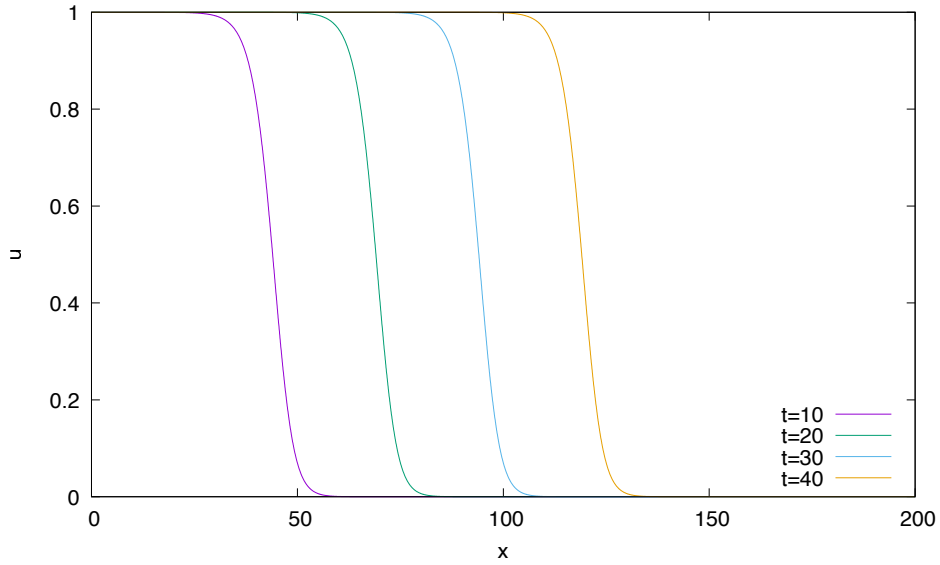

図 5: オイラー陽解法用いた、$b = 0.25$ の場合の Fisher 方程式の数値解 $(t = 10, 20, 30, 40)$

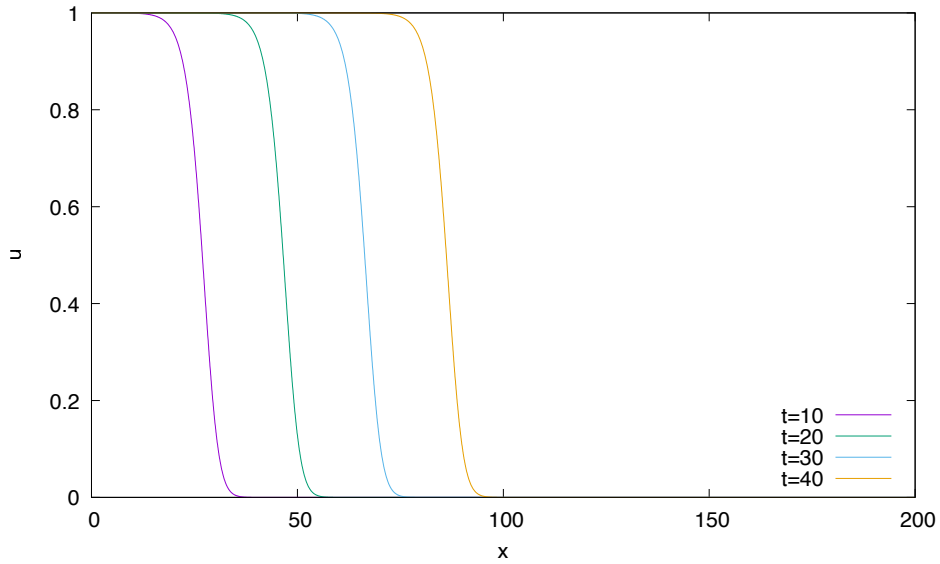

図 6: オイラー陽解法用いた、$b = 0.5$ の場合の Fisher 方程式の数値解 $(t = 10, 20, 30, 40)$

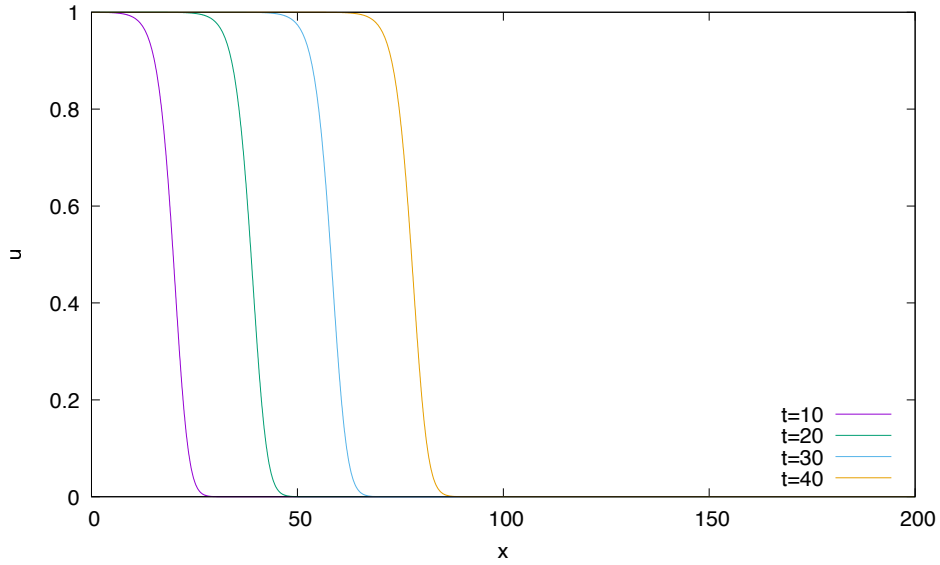

図 7: オイラー陽解法用いた、$b = 1.0$ の場合の Fisher 方程式の数値解 ($t = 10, 20, 30, 40$)

3.3 考察

これらの実験結果より、先述のように、いずれの（$b = 0.25, 0.5, 1.0$）場合も、$x = 0$ からしばらく $u \approx 1.0$ の値を取り、そこから一気に u は減少、$u \approx 0$ の値を取るとそこからほぼ一定で $u \approx 0$ を取り続けた。t が大きくなるにつれて、u の急減少が始まる x の値は大きくなり、b が大きくなるにつれて、u の急減少が始まる x の値は小さくなった。また、図を見ても分かるように、同じ b の値に対して t の値を変えても、u の急減少が始まる x の値が変わるだけで u が $x - u$ グラフで描く軌道はほとんど同じであるといえる。

初期条件 (22) によるところもあると考えられるが、このことから、t を大きくしていくとグラフが右寄りに（t を小さくしていくとグラフが左寄りに）、b を大きくしていくとグラフが左寄りに（b を小さくしていくとグラフが右寄りに）なっていくことが分かる。

また、$b = 0.25$ の場合の図 5 と $b = 0.5$ の場合の図 6 を見比べると、t の値を変えていった時に u が $x - u$ グラフで描く軌道の幅が狭まっている。しかし、$b = 0.5$ の場合の図 6 と $b = 1.0$ の場合の図 7 を見比べてもその様子は顕著にはみられない。

このことから、b の値を大きくしていってもこのスケールで見る限り、t の値を変えていった時に u が $x - u$ グラフで描く軌道の幅という観点ではあまり変化が見られなくなっていくと考えられる。

4　問題３

量子力学において、1 次元調和振動子のダイナミクスは、次の Schrödinger 方程式

$$i\hbar\frac{\partial\psi}{\partial t}(x,t)=\Big(-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}+\frac{k}{2}x^2\Big)\psi(x,t) \tag{27}$$

に従う。$\psi(x,t)=\psi_R(x,t)+i\psi_I(x,t)$ $(\psi_R,\psi_I\in\mathbb{R})$ のように波動関数を実部と虚部に分けると、それぞれの従う方程式は

$$\begin{aligned}\hbar\frac{\partial\psi_R}{\partial t}(x,t)&=\Big(-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}+\frac{k}{2}x^2\Big)\psi_I(x,t)\\ -\hbar\frac{\partial\psi_I}{\partial t}(x,t)&=\Big(-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}+\frac{k}{2}x^2\Big)\psi_R(x,t)\end{aligned} \tag{28}$$

となり、2 つの拡散型方程式の組で表される。この際、粒子の位置の確率密度は $P(x,t)\equiv|\psi(x,t)|^2=\psi_R(x,t)^2+\psi_I(x,t)^2$ で与えられる。

今回は、パラメータを $\hbar=1$, $m=1$, $k=1$ とし、初期条件

$$\psi(x,0)=\frac{\sqrt{2}}{\pi^{\frac{1}{4}}}e^{-2(x-5)^2} \tag{29}$$

の下で Schrödinger 方程式を数値的に解き、確率密度 $P(x,t)$ を求めることを考える。ただし、空間領域は $[-\frac{L}{2},\frac{L}{2}]$ とし、境界条件は周期境界条件

$$\psi\Big(\frac{L}{2},t\Big)=\psi\Big(-\frac{L}{2},t\Big) \tag{30}$$

とする。

4.1　実験方法

数値的に解くために、Visscher のスキームと呼ばれる、実部と虚部の時間発展を半ステップだけずらして解く陽解法を用いる。即ち、離散化された波動関数の実部、虚部をそれぞれ $\Big\{R_j^n\Big\}$, $\Big\{I_j^n\Big\}$ と書くと、$1\le j\le N$ $(L=N\Delta x)$ に対して時間発展は

$$\begin{aligned}R_j^{n+1}&=R_j^n+\Delta t\Big(-\frac{1}{2}\frac{I_{j-1}^n-2I_j^n+I_{j+1}^n}{\Delta x^2}+\frac{1}{2}x_j^2I_j^n\Big)\\ I_j^{n+1}&=I_j^n-\Delta t\Big(-\frac{1}{2}\frac{R_{j-1}^{n+1}-2R_j^{n+1}+I_{j+1}^{n+1}}{\Delta x^2}+\frac{1}{2}x_j^2R_j^{n+1}\Big)\end{aligned} \tag{31}$$

と記述される。ただし、

$$x_j\equiv\Big(j-\frac{1}{2}\Big)\Delta x-\frac{L}{2} \tag{32}$$

であり、境界条件は全ての n で

$$\begin{aligned}R_0^n&=R_N^n\\ R_{N+1}^n&=R_1^n\\ I_0^n&=I_N^n\\ I_{N+1}^n&=I_1^n\end{aligned} \tag{33}$$

と表される。また、初期条件は $1 \leq j \leq N$ で

$$
\begin{aligned}
R_j^0 &= \frac{1}{2}\Big[\psi\Big((j-1)\Delta x - \frac{L}{2}, 0\Big) + \psi\Big(j\Delta x - \frac{L}{2}, 0\Big)\Big] \\
I_j^0 &= -\Delta t\Big(-\frac{1}{2}\frac{R_{j-1}^0 - 2R_j^0 + R_{j+1}^0}{\Delta x^2} + \frac{1}{2}x_j^2 R_j^0\Big)
\end{aligned}
\tag{34}
$$

と表される。$L = 20$, $\Delta x = 0.05$, $\Delta t = 0.001$ の下でこの系を数値的に解き、$t = 1, 2, 3, 4, 5, 6, 7, 8$ における確率密度 $P_j^n = \Big(R_j^n\Big)^2 + I_j^n I_j^{n-1} (1 \leq j \leq N)$ の値を出力する。また、得られた確率密度 P_j^n の値を一つの図（横軸は x_j）に図示する。

4.2 実験結果

用いたソースコードを付録のソースコード 6 に示す。

出力では、いずれの $t = 1, 2, 3, 4, 5, 6, 7, 8$ の場合も、$x = -10$ からしばらく $P \approx 0$ の値を取り、ある x（x_s とする）からある x（x_m とする）まで P は増加し、そこからある x（x_f とする）まで減少し、$x = 10$ まで $P \approx 0$ の値を取り続けた。また、t の値を変えると、x_s, x_m, x_f の値は変化した。

出力に基づき、以下の図 8 を作成した。横軸は $(-\frac{L}{2} =) -10 \leq x \leq 10 (= \frac{L}{2})$、縦軸は $0 \leq P(x,t) \leq 1.0$ とした。ただし、先述のように、各 j に対応する座標として $x_j = \Big(j - \frac{1}{2}\Big)\Delta x - \frac{L}{2}$ を採用した。

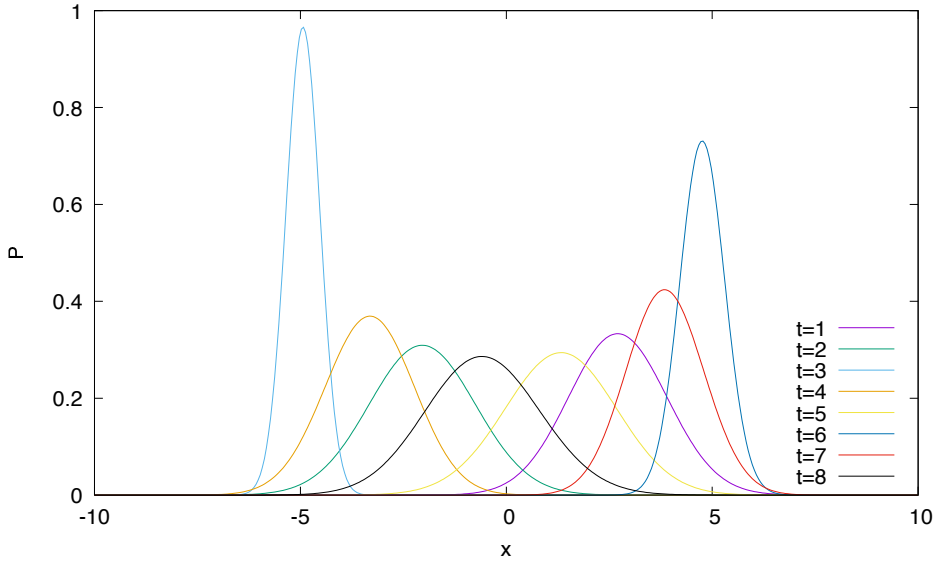

図 8: $t = 1, 2, 3, 4, 5, 6, 7, 8$ における Schrödinger 方程式を Visscher のスキームで数値的に解き、確率密度 $P_j^n = \Big(R_j^n\Big)^2 + I_j^n I_j^{n-1} (1 \leq j \leq N)$ で表現したもの

4.3 考察

これらの実験結果より、いずれの $t = 1, 2, 3, 4, 5, 6, 7, 8$ の場合も、$x = -10$ からしばらく $P \approx 0$ の値を取り、x_s から x_m まで P は増加、そこから x_f まで減少、$x = 10$ まで $P \approx 0$ の値を取り続けるということが分かった。すなわち、図 8 から見ても分かるように、Schrödinger 方程式の数値解を確率密度 $P(x,t) = \psi_R(x,t)^2 + \psi_I(x,t)^2$ で表現した場合、$x - P$ グラフでは上に凸の軌道を描くことが分かった。

また、t の値を変えた時、x_s, x_m, x_f の値は変化し、$P(x_m, t)$ の値も変化することが分かった。この図からは t の値の変化と P が $x - P$ グラフで描く軌道の変化の規則性についてはあまり理解することができなかったが、少なくとも x_m が中心 $x = 0$ により近い値を取れば、$x_f - x_s$（上に凸の軌道の幅）はより大きい値、$P(x_m, t)$（上に凸の軌道の高さ）はより小さい値をとり、逆に x_m が中心 $x = 0$ により遠い値を取れば、$x_f - x_s$ はより小さい値、$P(x_m, t)$ はより大きい値を取ることが分かった。

5　付録

ソースコード 1: C 言語を用いて拡散方程式を数値的に解くプログラム

```
1  //数理工学実験テーマ３ //問題１ //オイラー陽解法Dirichlet
2  #include <stdio.h>
3  #include <math.h>
4
5  const double pi = 3.14159265358979; //円周率の定義
6
7  double uic(double x) //u_1~u_N の初期条件を求めるための関数定義
8  {
9      double u0 = 1 / sqrt(2 * pi) * exp(-0.5 * pow(x - 5,
           2));
10     return u0;
11 }
12
13 int main(int argc, char **argv)
14 {
15     int n, j;
16
17     int L = 10; //幅
18
19     //境界条件
20     double ul = 0.0;
21     double ur = 0.0;
22
23     double deltaT = 0.01; //時間の刻み幅
24     double deltaX = 0.5; //空間の刻み幅
25     double c = deltaT / pow(deltaX, 2); //c の定義
26
27     for (int t = 1; t <= 5; t++) //t=1,2,3,4,5の場合(時間の
           刻み幅ではないことに注意)
28     {
29
30         int T = t / deltaT; //時間の刻み数
31         int N = L / deltaX; //空間の刻み数=20
32
33         double u[N + 2]; //u_j を格納する配列
34         double preu[N + 2]; //一段階前ののu_j を格納する配列
35
36         preu[0] = ul;
37         preu[N + 1] = ur;
38         for (j = 1; j <= N; j++) //u_1~
               u_N の初期条件を求めるための for 文
39         {
```

```
40              preu[j] = 0.5 * (uic((j - 1) * deltaX) + uic(j
                    * deltaX));
41          }
42
43          for (n = 1; n <= T; n++)
44          {
45              for (j = 1; j <= N; j++)
46              {
47                  u[j] = preu[j] + c * (preu[j - 1] - 2 * preu
                        [j] + preu[j + 1]); //オイラー陽解法
48              }
49              for (j = 1; j <= N; j++) //
                    u_j を次の段階で、一段前の u_j として用いるための for 文

50              {
51                  preu[j] = u[j];
52              }
53          }
54
55          printf("t=%d の時(x の値　u の値) \n", t);
56
57          printf("%lf %lf\n", 0.0, ul); //x=0のu:u_0
58          for (j = 1; j <= N; j++) //u1~
                u_N を出力するための for 文
59          {
60              printf("%lf %lf\n", (j - 0.5) * deltaX, u[j]);
61          }
62          printf("%lf %lf\n", 10.0, ur); //x=10のu:u_{N+1}
63          printf("\n");
64      }
65
66      return 0;
67  }
```

ソースコード 2: C 言語を用いて拡散方程式を数値的に解くプログラム

```
1  //数理工学実験テーマ３ //問題１ //オイラー陽解法Neumann
2  #include <stdio.h>
3  #include <math.h>
4
5  const double pi = 3.14159265358979; //円周率の定義
6
7  double uic(double x) //u_1~u_N の初期条件を求めるための関数定義
8  {
9      double u0 = 1 / sqrt(2 * pi) * exp(-0.5 * pow(x - 5,
           2));
10     return u0;
11 }
12
13 int main(int argc, char **argv)
14 {
15     int n, j;
16
17     int L = 10; //幅
18
19     double deltaT = 0.01; //時間の刻み幅
20     double deltaX = 0.5; //空間の刻み幅
21     double c = deltaT / pow(deltaX, 2); //c の定義
22
23     for (int t = 1; t <= 5; t++) //t=1,2,3,4,5の場合(時間の
           刻み幅ではないことに注意)
24     {
25
26         int T = t / deltaT; //時間の刻み数
27         int N = L / deltaX; //空間の刻み数=20
28
29         double u[N + 2]; //u_j を格納する配列
30         double preu[N + 2]; //一段階前ののu_j を格納する配列
31
32         for (j = 1; j <= N; j++) //u_1~
               u_N の初期条件を求めるための for 文
33         {
34             preu[j] = 0.5 * (uic((j - 1) * deltaX) + uic(j
                   * deltaX));
35         }
36
37         //境界条件
38         preu[0] = preu[1];
39         preu[N + 1] = preu[N];
40
41         for (n = 1; n <= T; n++)
```

```
42          {
43              for (j = 1; j <= N; j++)
44              {
45                  u[j] = preu[j] + c * (preu[j - 1] - 2 * preu
                        [j] + preu[j + 1]); //オイラー陽解法
46              }
47              for (j = 1; j <= N; j++) //
                    u_j を次の段階で、一段前の u_j として用いるための for 文

48              {
49                  preu[j] = u[j];
50              }
51
52              //次の段階のための境界条件
53              preu[0] = preu[1];
54              preu[N + 1] = preu[N];
55          }
56
57          printf("t=%d の時(x の値   u の値) \n", t);
58
59          printf("%lf %lf\n", 0.0, u[1]); //x=0のu:u_0
60          for (j = 1; j <= N; j++) //u1~
                u_N を出力するための for 文
61          {
62              printf("%lf %lf\n", (j - 0.5) * deltaX, u[j]);
63          }
64          printf("%lf %lf\n", 10.0, u[N]); //x=10のu:u_{N+1}
65          printf("\n");
66      }
67
68      return 0;
69  }
```

ソースコード 3: C 言語を用いて拡散方程式を数値的に解くプログラム

```
1  //数理工学実験テーマ３ //問題１ //クランク･ニコルソン法
       Dirichlet
2  #include <stdio.h>
3  #include <math.h>
4
5  const double pi = 3.14159265358979; //円周率の定義
6
7  double uic(double x) //u_1~u_N の初期条件を求めるための関数定義
8  {
9      double u0 = 1 / sqrt(2 * pi) * exp(-0.5 * pow(x - 5,
           2));
10     return u0;
11 }
12
13 int main(int argc, char **argv)
14 {
15     int n, j;
16
17     int L = 10; //幅
18
19     //境界条件
20     double ul = 0.0;
21     double ur = 0.0;
22
23     double deltaT = 0.01; //時間の刻み幅
24     double deltaX = 0.05; //空間の刻み幅
25     double c = deltaT / pow(deltaX, 2); //c の定義
26
27     for (int t = 1; t <= 5; t++) //t=1,2,3,4,5の場合(時間の
           刻み幅ではないことに注意)
28     {
29
30         int T = t / deltaT; //時間の刻み数
31         int N = L / deltaX; //空間の刻み数=200
32
33         double u[N + 2]; //u_j を格納する配列
34         double postu[N + 2]; //一段階後ののu_j を格納する配列
35
36         u[0] = ul;
37         u[N + 1] = ur;
38         for (j = 1; j <= N; j++) //u_1~
               u_N の初期条件を求めるための for 文
39         {
40             u[j] = 0.5 * (uic((j - 1) * deltaX) + uic(j *
                   deltaX));
```

```
41          }
42
43          //LU 分解のための配列（u,
                postu の配列と同様にアドレスが一致するようにしている）
44          double a[N + 1], b[N]; //この場合c（行列の要素）は、b=
                c なので b にまとめる。
45          double alpha[N + 1], beta[N];
46          double x[N + 1], y[N + 1], z[N + 1]; //
                x は、postu に対応
47          a[0] = 0;
48          b[0] = 0;
49          alpha[0] = 0;
50          beta[0] = 0;
51          x[0] = 0;
52          y[0] = 0;
53          z[0] = 0;
54
55          for (n = 1; n <= T; n++)
56          {
57              for (j = 1; j <= N - 1; j++) //a_1~N-1, b_1~N-1
                    に(Dirichlet 境界条件の場合の）値を代入
58              {
59                  a[j] = 1.0 + c;
60                  b[j] = -c / 2;
61              }
62              a[N] = 1.0 + c;
63
64              for (j = 1; j <= N - 1; j++) //α_1~N-1, β_1~N
                    -1に(Dirichlet 境界条件の場合の）値を代入
65              {
66                  alpha[j] = a[j] - b[j - 1] * beta[j - 1];
                        //b_0(c_0)=β_0=0と代入済み
67                  beta[j] = b[j] / alpha[j];
68              }
69              alpha[N] = a[N] - b[N - 1] * beta[N - 1]; //α
                    _N の値
70
71              z[1] = (1 - c) * u[1] + c * ul + c / 2 * u[2];
                     //z_1 の値
72              for (j = 2; j <= N - 1; j++) //z_2~N-1に(
                    Dirichlet 境界条件の場合の）値を代入
73              {
74                  z[j] = (1 - c) * u[j] + c / 2 * u[j - 1] +
                        c / 2 * u[j + 1];
75              }
76              z[N] = (1 - c) * u[N] + c * ur + c / 2 * u[N -
```

```
                1]; //z_N の値
77
78              for (j = 1; j <= N; j++) //y_1~N の値 //b_0(c_0
                    )=0と代入済み
79              {
80                  y[j] = (z[j] - b[j - 1] * y[j - 1]) / alpha
                        [j];
81              }
82
83              x[N] = y[N];
84              for (j = N - 1; j >= 1; j--) //x_N-1~1(降順)の
                    値
85              {
86                  x[j] = y[j] - beta[j] * x[j + 1];
87              }
88
89              for (j = 1; j <= N; j++) //postu に x の値を格納
90              {
91                  postu[j] = x[j];
92              }
93
94              for (j = 1; j <= N; j++) //postu を u に格納
95              {
96                  u[j] = postu[j];
97              }
98          }
99
100         printf("t=%d の時(x の値   u の値) \n", t);
101
102         printf("%lf %lf\n", 0.0, ul); //x=0のu:u_0
103         for (j = 1; j <= N; j++) //u1~
                u_N を出力するための for 文
104         {
105             printf("%lf %lf\n", (j - 0.5) * deltaX, u[j]);
106         }
107         printf("%lf %lf\n", 10.0, ur); //x=10のu:u_{N+1}
108         printf("\n");
109     }
110
111     return 0;
112 }
```

ソースコード 4: C 言語を用いて拡散方程式を数値的に解くプログラム

```
1  //数理工学実験テーマ３ //問題１ //クランク･ニコルソン法
       Neumann
2  #include <stdio.h>
3  #include <math.h>
4
5  const double pi = 3.14159265358979; //円周率の定義
6
7  double uic(double x) //u_1~u_N の初期条件を求めるための関数定義
8  {
9      double u0 = 1 / sqrt(2 * pi) * exp(-0.5 * pow(x - 5,
           2));
10     return u0;
11 }
12
13 int main(int argc, char **argv)
14 {
15     int n, j;
16
17     int L = 10; //幅
18
19     double deltaT = 0.01; //時間の刻み幅
20     double deltaX = 0.05; //空間の刻み幅
21     double c = deltaT / pow(deltaX, 2); //c の定義
22
23     for (int t = 1; t <= 5; t++) //t=1,2,3,4,5の場合(時間の
           刻み幅ではないことに注意)
24     {
25
26         int T = t / deltaT; //時間の刻み数
27         int N = L / deltaX; //空間の刻み数=200
28
29         double u[N + 2]; //u_j を格納する配列
30         double postu[N + 2]; //一段階後ののu_j を格納する配列
31
32         for (j = 1; j <= N; j++) //u_1~
               u_N の初期条件を求めるための for 文
33         {
34             u[j] = 0.5 * (uic((j - 1) * deltaX) + uic(j *
                   deltaX));
35         }
36
37         //境界条件
38         u[0] = u[1];
39         u[N + 1] = u[N];
40
```

```
41          //LU 分解のための配列（u,
                postu の配列と同様にアドレスが一致するようにしている）
42          double a[N + 1], b[N]; //この場合c（行列の要素）は、b=
                c なので b にまとめる。
43          double alpha[N + 1], beta[N];
44          double x[N + 1], y[N + 1], z[N + 1]; //
                x は、postu に対応
45          a[0] = 0;
46          b[0] = 0;
47          alpha[0] = 0;
48          beta[0] = 0;
49          x[0] = 0;
50          y[0] = 0;
51          z[0] = 0;
52
53          for (n = 1; n <= T; n++)
54          {
55              a[1] = 1.0 + c / 2; //a_1 の値
56              b[1] = -c / 2; //b_1 の値
57              for (j = 2; j <= N - 1; j++) //a_2~N-1, b_2~N-1
                    に(Dirichlet 境界条件の場合の）値を代入
58              {
59                  a[j] = 1.0 + c;
60                  b[j] = -c / 2;
61              }
62              a[N] = 1.0 + c / 2; //a_N の値
63
64              for (j = 1; j <= N - 1; j++) //α_1~N-1, β_1~N
                    -1に(Dirichlet 境界条件の場合の）値を代入
65              {
66                  alpha[j] = a[j] - b[j - 1] * beta[j - 1];
                        //b_0(c_0)=β_0=0と代入済み
67                  beta[j] = b[j] / alpha[j];
68              }
69              alpha[N] = a[N] - b[N - 1] * beta[N - 1]; //α
                    _N の値
70
71              z[1] = (1 - c / 2) * u[1] + c / 2 * u[2]; //
                    z_1 の値
72              for (j = 2; j <= N - 1; j++) //z_2~N-1に(
                    Dirichlet 境界条件の場合の）値を代入
73              {
74                  z[j] = (1 - c) * u[j] + c / 2 * u[j - 1] +
                        c / 2 * u[j + 1];
75              }
76              z[N] = (1 - c / 2) * u[N] + c / 2 * u[N - 1];
```

```
                //z_N の値
77
78              for (j = 1; j <= N; j++) //y_1~N の値 //b_0(c_0
                    )=0と代入済み
79              {
80                  y[j] = (z[j] - b[j - 1] * y[j - 1]) / alpha
                        [j];
81              }
82
83              x[N] = y[N];
84              for (j = N - 1; j >= 1; j--) //x_N-1~1(降順)の
                    値
85              {
86                  x[j] = y[j] - beta[j] * x[j + 1];
87              }
88
89              for (j = 1; j <= N; j++) //postu に x の値を格納
90              {
91                  postu[j] = x[j];
92              }
93
94              for (j = 1; j <= N; j++) //postu を u に格納
95              {
96                  u[j] = postu[j];
97              }
98
99              //次の段階のための境界条件
100             u[0] = postu[1];
101             u[N + 1] = postu[N];
102         }
103
104         printf("t=%d の時(x の値   u の値) \n", t);
105
106         printf("%lf %lf\n", 0.0, u[1]); //x=0のu:u_0
107         for (j = 1; j <= N; j++) //u1~
                u_N を出力するための for 文
108         {
109             printf("%lf %lf\n", (j - 0.5) * deltaX, u[j]);
110         }
111         printf("%lf %lf\n", 10.0, u[N]); //x=10のu:u_{N+1}
112         printf("\n");
113     }
114
115     return 0;
116 }
```

ソースコード 5: C 言語を用いて Fisher 方程式を数値的に解くプログラム

```
1  //数理工学実験テーマ３ //問題２ //Fisher 方程式 オイラー陽解法 Dirichlet
2  #include <stdio.h>
3  #include <math.h>
4
5  double uic(double b, double x) //u_1~u_N の初期条件を求めるための関数定義
6  {
7      double u0 = 1 / pow(1 + exp(b * x - 5), 2);
8      return u0;
9  }
10
11 int main(int argc, char **argv)
12 {
13     int n, j;
14
15     int L = 200; //幅
16
17     //境界条件
18     double ul = 1.0;
19     double ur = 0.0;
20
21     double deltaT = 0.001; //時間の刻み幅
22     double deltaX = 0.05; //空間の刻み幅
23
24     for (double b = 0.25; b <= 1.0; b = b * 2)
25     {
26         for (int t = 10; t <= 40; t = t + 10) //t=10,20,30,40の場合
27         {
28             int T = t / deltaT; //時間の刻み数
29             int N = L / deltaX; //空間の刻み数=4000
30
31             double u[N + 2]; //u_j を格納する配列
32             double postu[N + 2]; //一段階後のu_j を格納する配列
33
34             u[0] = ul;
35             u[N + 1] = ur;
36             for (j = 1; j <= N; j++) //u_1~u_N の初期条件を求めるための for 文
37             {
38                 u[j] = 0.5 * (uic(b, (j - 1) * deltaX) +
                       uic(b, j * deltaX));
39             }
40
```

```
41              for (n = 1; n <= T; n++)
42              {
43                  for (j = 1; j <= N; j++)
44                  {
45                      postu[j] = u[j] + deltaT * (u[j] * (1 -
                            u[j]) + (u[j - 1] - 2 * u[j] + u[j
                            + 1]) / pow(deltaX, 2)); //オイラー
                            陽解法
46                  }
47                  for (j = 1; j <= N; j++) //
                        u_j を次の段階で、一段前の u_j として用いるための for 文

48                  {
49                      u[j] = postu[j];
50                  }
51              }
52
53              printf("b=%lf t=%d の時(x の値   u の値) \n", b, t);
54
55              printf("%lf %lf\n", 0.0, ul); //x=0のu:u_0
56              for (j = 1; j <= N; j++) //u1~
                    u_N を出力するための for 文
57              {
58                  printf("%lf %lf\n", (j - 0.5) * deltaX, u[j
                        ]);
59              }
60              printf("%lf %lf\n", 200.0, ur); //x=200のu:u_{N
                    +1}
61              printf("\n");
62          }
63      }
64
65      return 0;
66  }
```

ソースコード 6: C 言語を用いて Schrödinger 方程式を数値的に解くプログラム

```
1  //数理工学実験テーマ３ //問題３ //Visscher のスキーム
2  #include <stdio.h>
3  #include <math.h>
4
5  const double pi = 3.14159265358979; //円周率の定義
6
7  double Ric(double x) //R_1~R_N の初期条件を求めるための関数定義
8  {
9      double u0 = sqrt(2) / pow(pi, 0.25) * exp(-2 * pow(x -
           5, 2));
10     return u0;
11 }
12
13 int main(int argc, char **argv)
14 {
15     int n, j;
16
17     int L = 20; //幅
18
19     double deltaT = 0.001; //時間の刻み幅
20     double deltaX = 0.05; //空間の刻み幅
21
22     for (int t = 1; t <= 8; t++) //t=1,2,3,4,5,6,7,8の場合
23     {
24         int T = t / deltaT; //時間の刻み数
25         int N = L / deltaX; //空間の刻み数=400
26
27         double R[N + 2]; //R_j を格納する配列
28         double postR[N + 2]; //一段階後のR_j を格納する配列
29         double I[N + 2]; //I_j を格納する配列
30         double postI[N + 2]; //一段階後のI_j を格納する配列
31
32         double P[N + 2]; //確率密度P_j を格納する配列
33
34         //R_1~N の初期条件を求めるための for 文
35         for (j = 1; j <= N; j++)
36         {
37             R[j] = 0.5 * (Ric((j - 1) * deltaX - L / 2) +
                   Ric(j * deltaX - L / 2));
38         }
39         //R の境界条件(t=0)
40         R[0] = R[N];
41         R[N + 1] = R[1];
42
```

```
43          //I_1~N の初期条件を求めるための for 文
44          for (j = 1; j <= N; j++)
45          {
46              I[j] = -deltaT * (-0.5 * (R[j - 1] - 2 * R[j] +
                     R[j + 1]) / pow(deltaX, 2) + 0.5 * pow((j
                    - 0.5) * deltaX - L / 2, 2) * R[j]);
47          }
48          //I の境界条件(t=0)
49          I[0] = I[N];
50          I[N + 1] = I[1];
51
52          for (n = 1; n <= T; n++)
53          {
54              //R の時間発展
55              for (j = 1; j <= N; j++)
56              {
57                  postR[j] = R[j] + deltaT * (-0.5 * (I[j -
                        1] - 2 * I[j] + I[j + 1]) / pow(deltaX,
                         2) + 0.5 * pow((j - 0.5) * deltaX - L
                        / 2, 2) * I[j]);
58              }
59              //R の境界条件
60              postR[0] = postR[N];
61              postR[N + 1] = postR[1];
62
63              //I の時間発展
64              for (j = 1; j <= N; j++)
65              {
66                  postI[j] = I[j] - deltaT * (-0.5 * (postR[j
                        - 1] - 2 * postR[j] + postR[j + 1]) /
                        pow(deltaX, 2) + 0.5 * pow((j - 0.5) *
                        deltaX - L / 2, 2) * postR[j]);
67              }
68              //I の境界条件
69              postI[0] = postI[N];
70              postI[N + 1] = postI[1];
71
72              //確率密度
                    P_j の計算（毎回確率密度 P を計算する必要はないが、I
                    [j]がI[j] = postI[j]と代入されてしまう前に計
                    算しておくため）
73              P[0] = pow(postR[0], 2) + postI[0] * I[0];
74              for (j = 1; j <= N; j++)
75              {
76                  P[j] = pow(postR[j], 2) + postI[j] * I[j];
77              }
```

```
78              P[N + 1] = pow(postR[N + 1], 2) + postI[N + 1]
                    * I[N + 1];
79
80              //R_j,I_j を次の段階で、一段前の R_j,
                    I_j として用いるための for 文
81              for (j = 1; j <= N; j++)
82              {
83                  R[j] = postR[j];
84                  I[j] = postI[j];
85              }
86          }
87
88          printf("t=%d の時(x の値   P の値) \n", t);
89
90          printf("%lf %lf\n", -10.0, P[0]); //x=-10のP
91          for (j = 1; j <= N; j++) //u1~
                u_N を出力するための for 文
92          {
93              printf("%lf %lf\n", (j - 0.5) * deltaX - L / 2,
                     P[j]);
94          }
95          printf("%lf %lf\n", 10.0, P[N + 1]); //x=10のP
96          printf("\n");
97      }
98
99      return 0;
100 }
```

6 参考文献

- 数理工学実験テキスト
- Cloud LaTeX
  https://cloudlatex.io
- LaTeX コマンド一覧（リスト）
  https://medemanabu.net/latex/latex-commands-list/
- LaTeX にソースコードを【美しく】貼る方法
  https://ta-b0.hateblo.jp/entry/2020/08/13/001223
- 範囲外から範囲外へ動く場合の線の描画
  https://qiita.com/bunzaemon/items/c51c1ffb01f011252a27
- コメント（注釈） - 複数行のコメントアウト
  https://medemanabu.net/latex/comment/
- 【LaTeX】箇条書きの方法ついて徹底解説
  https://mathlandscape.com/latex-enum/
- TeX で複素数 C、実数 R、有理数 Q、整数 Z、自然数 N などの白抜き文字を使う方法
  http://did2.blog64.fc2.com/blog-entry-204.html