数理工学実験
テーマ:熱方程式の差分法

田中風帆 (1029321151)
実施場所:自宅

実施:2021 年 10 月 25 日
提出:2021 年 11 月 7 日

1　概要

このレポートでは、全体を通して熱方程式を差分法によって解き、議論する。問題 1 ではオイラー陽解法とクランクニコルソン法、および Dirichlet 境界条件と Neumann 境界条件を組み合わせて拡散方程式を解き、グラフとして出力する。問題 2 では、拡散方程式に非線形項を加えた偏微分方程式である Fisher 方程式を、オイラー陽解法を用いて数値的に解く。問題 3 では、一次元調和振動子のシュレディンガー方程式を数値的に解き、いくつかの時刻で得られた確率密度の値をプロットする。

2　問題 1

この問題では、拡散方程式 $\frac{\partial u(x,t)}{\partial t} = \frac{\partial^2 u(x,t)}{\partial x^2}$...(∗) を、初期条件 $u_0 = \frac{\exp(\frac{-(x-5)^2}{2})}{\sqrt{2\pi}}$ のもとで解く。この際、境界条件として Dirichlet 境界条件および Neumann 境界条件を、手法としてオイラー陽解法およびクランクニコルソン法を用いた。なお、以下 u_j^n は座標 x, 時刻 t($n = t/\Delta t, j = x/\Delta x$、ただし Δt は時間の刻み幅、Δx は座標の刻み幅) における方程式の解とする。
1. ここでは、Dirichlet 境界条件において $u_L = u_R = 0$ を用い、オイラー陽解法 $u_j^{n+1} - u_j^n = \frac{\Delta t(u_{j-1}^n - 2u_j^n + u_{j+1}^n)}{\Delta x^2}$ によって (∗) を解いた。ただし、Dirichlet 境界条件は座標を N 区間に離散化したとき $u_0^n = u_L, u_{N+1} = u_R$ と表せる。時刻の刻み幅は $\Delta t = 0.01$, 座標の刻み幅は $\Delta x = 0.5$ として、$x = 0$ から $x = 10$ までを対象に計算を行った。計算の結果得られた出力 u の値は $x = 5$ に関して対称であり、$x = 4.25, 4.75$ の時が最も大きく、x の値がそれらから離れれば離れるほど小さくなって行った。結果のグラフは以下のようになった。

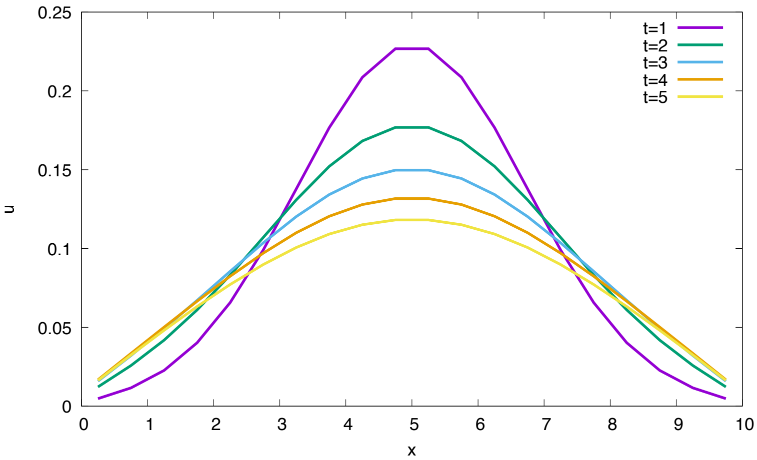

図 1: オイラー陽解法、Dirichlet 境界条件を用いて解いた拡散方程式の解

コードは以下である。

コード 1: オイラー陽解法、Dirichlet 境界条件を用いて拡散方程式を解くコード

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

double u_0(double x){
    return exp(-(x-5)*(x-5)/2) / sqrt(2*M_PI);
}

int main(){
    //ハイパーパラメータの定義
    int N = 20;
    double dt = 0.01;//時間の刻み幅
    double dx = 0.5;//座標の刻み幅
    double u[N+2], u_tmp[N+2];//u の値
    double time_limit = 6.0;
    double step_num = (int)time_limit/dt;//繰り返し回数
    double t = 0;//時刻の初期値

    FILE *fp1_1_1;//ファイルに値を出力
    FILE *fp1_1_2;
    FILE *fp1_1_3;
    FILE *fp1_1_4;
    FILE *fp1_1_5;
    fp1_1_1= fopen("kadai1_1_1.txt", "w");
    fp1_1_2 = fopen("kadai1_1_2.txt", "w");
    fp1_1_3 = fopen("kadai1_1_3.txt", "w");
    fp1_1_4 = fopen("kadai1_1_4.txt", "w");
    fp1_1_5 = fopen("kadai1_1_5.txt", "w");


    //境界条件
    double u_L = 0, u_R = 0;
    u[0] = u_L; u[N+1] = u_R; u_tmp[0] = u_L; u_tmp[N+1] = u_R;

    //の初期条件 u
    for(int j=1; j<=N; j++){
        u[j] = (u_0((j-1)*dx) + u_0(j*dx)) / 2;
        u_tmp[j] = 0;
    }

    //更新
    for(int i=1; i<=step_num; i++){
        t += dt;
        for(int j=1; j<=N; j++){
            u_tmp[j] = u[j] + dt*(u[j-1]-2*u[j]+u[j+1])/(dx*dx);
        }

        for(int j=1; j<=N; j++){
            u[j] = u_tmp[j];
        }

        //出力
        if(i==100){
            for(int j=1; j<=N; j++){
                double x = dx*((double)j-0.5);
                fprintf(fp1_1_1,"%.8f %.8f \n", x, u[j]);
            }
```

```
        }
        else if(i==200){
            for(int j=1; j<=N; j++){
                double x = dx*((double)j-0.5);
                fprintf(fp1_1_2,"%.8f %.8f \n", x, u[j]);
            }
        }
        else if(i==300){
            for(int j=1; j<=N; j++){
                double x = dx*((double)j-0.5);
                fprintf(fp1_1_3,"%.8f %.8f \n", x, u[j]);
            }
        }
        else if(i==400){
            for(int j=1; j<=N; j++){
                double x = dx*((double)j-0.5);
                fprintf(fp1_1_4,"%.8f %.8f \n", x, u[j]);
            }
        }
        else if(i==500){
            for(int j=1; j<=N; j++){
                double x = dx*((double)j-0.5);
                fprintf(fp1_1_5,"%.8f %.8f \n", x, u[j]);
            }
        }
    }

    //ファイルを閉じる
    fclose(fp1_1_1);
    fclose(fp1_1_2);
    fclose(fp1_1_3);
    fclose(fp1_1_4);
    fclose(fp1_1_5);
}
```

コードの説明を行う。まず、u の初期条件 $u_0 = \frac{\exp(\frac{-(x-5)^2}{2})}{\sqrt{2\pi}}$ を定義する。コード内において、u[i] に x_i の値を代入する。u[0] および u[N+1] は境界である。実行部分では、境界条件を定義したのち、u を表す配列に初期条件を格納する。この時、u_j に区間 [(j-1)Δt,jΔt] を割り当てていることから、初期条件として $u_j^0 = [u_0((j-1)\Delta x) + u_0(j\Delta x)]$ を採用した。その後、時刻 t が 5 を超えるまで更新を繰り返し、$t = 1, 2, 3, 4, 5$ の時の x および u の値を時刻ごとに別々のファイルに出力する。なお、初期条件の採用と同じ理由により、出力する x 座標として $(j-1/2)\Delta x$ を採用した。

2. ここでは、オイラー陽解法 $u_j^{n+1} - u_j^n = \frac{\Delta t(u_{j-1}^n - 2u_j^n + u_{j+1}^n)}{\Delta x^2}$ および Neumann 境界条件 $(J_L = J_R = 0)$ を用いて数値的に解を求める。ただし、Neumann 境界条件は座標を N 区間に離散化したとき $u_0^n = u_1^n - J_L\Delta x, u_{N+1}^n = u_N^n + J_R\Delta x$ と表せる。ここでは、$\Delta t = 0.01, \Delta x = 0.5$ とした。計算の結果得られた出力 u の値は $x = 5$ に関して対称であり、$x = 4.25, 4.75$ の時が最も大きく、x の値がそれらから離れれば離れるほど小さくなって行った。結果のグラフは以下である。

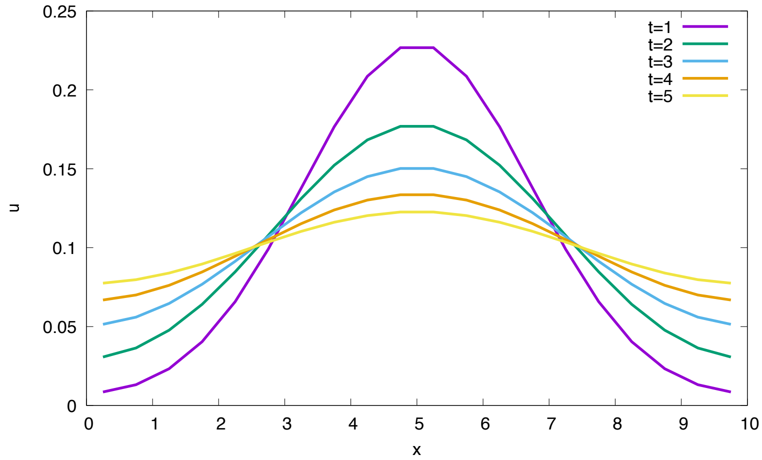

図 2: オイラー陽解法、Neumann 境界条件を用いて解いた拡散方程式の解

使用したコードは以下である。

コード 2: オイラー陽解法、Neumann 境界条件を用いて拡散方程式を解くコード

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

double u_0(double x){
    return exp(-(x-5)*(x-5)/2) / sqrt(2*M_PI);
}

int main(){
    //ハイパーパラメータの定義
    int N = 20;
    double dt = 0.01;
    double dx = 0.5;
    double u[N+2], u_tmp[N+2];
    double time_limit = 6.0;
    double step_num = (int)time_limit/dt;
    double t = 0;
    double J_L = 0, J_R = 0;
    bool recorded_1=false, recorded_2=false, recorded_3=false, recorded_4
        =false, recorded_5=false;

    FILE *fp1_1_1;//ファイルに値を出力
    FILE *fp1_1_2;
    FILE *fp1_1_3;
    FILE *fp1_1_4;
    FILE *fp1_1_5;
    fp1_1_1= fopen("kadai1_2_1.txt", "w");
    fp1_1_2 = fopen("kadai1_2_2.txt", "w");
    fp1_1_3 = fopen("kadai1_2_3.txt", "w");
    fp1_1_4 = fopen("kadai1_2_4.txt", "w");
    fp1_1_5 = fopen("kadai1_2_5.txt", "w");

```

```c

    //境界条件
    double u_L = 0, u_R = 0;
    u[0] = u_L; u[N+1] = u_R; u_tmp[0] = u_L; u_tmp[N+1] = u_R;

    //の初期条件 u
    for(int j=1; j<=N; j++){
        u[j] = (u_0((j-1)*dx) + u_0(j*dx)) / 2;
        u_tmp[j] = 0;
    }

    for(int i=1; i<=step_num; i++){
        for(int j=1; j<=N; j++){
            u_tmp[j] = u[j] + dt*(u[j-1]-2*u[j]+u[j+1])/(dx*dx);
        }

        for(int j=1; j<=N; j++){
            u[j] = u_tmp[j];
        }
        //境界条件の更新
        u_tmp[0] = u[1] - J_L*dx;
        u_tmp[N+1] = u[N] + J_R*dx;
        u[0] = u_tmp[0];
        u[N+1] = u_tmp[N+1];

        t += dt;

        //ファイルに出力
        if(i==100){
            recorded_1 = true;
            printf("%.8f", t);
            for(int j=1; j<=N; j++){
                double x = dx*((double)j-0.5);
                fprintf(fp1_1_1,"%.8f %.8f \n", x, u[j]);
            }
        }
        else if(i==200){
            recorded_2 = true;
            printf("%.8f", t);
            for(int j=1; j<=N; j++){
                double x = dx*((double)j-0.5);
                fprintf(fp1_1_2,"%.8f %.8f \n", x, u[j]);
            }
        }
        else if(i==300){
            recorded_3 = true;
            printf("%.8f", t);
            for(int j=1; j<=N; j++){
                double x = dx*((double)j-0.5);
                fprintf(fp1_1_3,"%.8f %.8f \n", x, u[j]);
            }
        }
        else if(i==400){
            recorded_4 = true;
            printf("%.8f", t);
            for(int j=1; j<=N; j++){
                double x = dx*((double)j-0.5);
                fprintf(fp1_1_4,"%.8f %.8f \n", x, u[j]);
            }
        }
        else if(i==500){
            recorded_5 = true;
            printf("%.8f", t);
```

```
            for(int j=1; j<=N; j++){
                double x = dx*((double)j-0.5);
                fprintf(fp1_1_5,"%.8f %.8f\n", x, u[j]);
            }
        }
    }

    fclose(fp1_1_1);
    fclose(fp1_1_2);
    fclose(fp1_1_3);
    fclose(fp1_1_4);
    fclose(fp1_1_5);
}
```

コードの構造は先ほどのものと同じである。J_l, J_R を定義し、境界条件として Neumann 境界条件を採用しているのが相違点である。

3. ここでは、クランクニコルソン法および Delichlet 境界条件を用いて解を求める。$\Delta t = 0.01, \Delta x = 0.05$ とした。計算の結果得られた出力 u の値は $x = 5$ に関して対称であり、$x = 4.25, 4.75$ の時が最も大きく、x の値がそれらから離れれば離れるほど小さくなって行った。以下が結果のグラフである。

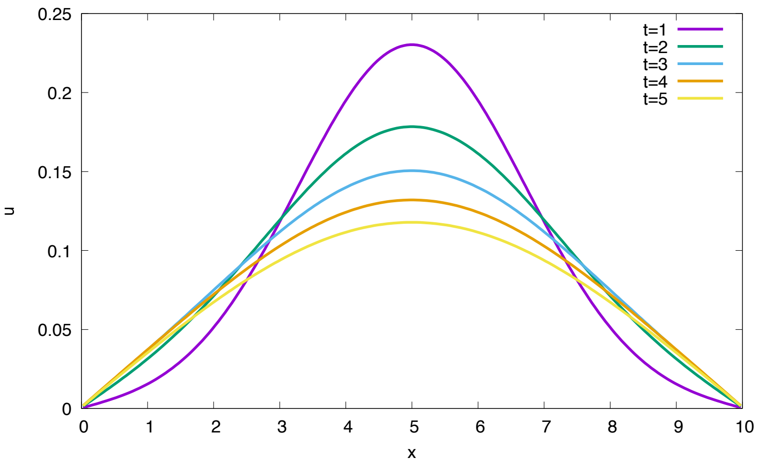

図 3: クランクニコルソン法、Delichlet 境界条件を用いて解いた拡散方程式の解

コードは以下である。

コード 3: クランクニコルソン法、Delichlet 境界条件を用いて拡散方程式を解くコード

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

double u_0(double x){
```

```c
    return exp(-(x-5)*(x-5)/2) / sqrt(2*M_PI);
}

int main(){
    //ハイパーパラメータの定義
    int N = 200;
    double dx = 0.05;
    double dt = 0.01;
    double t = 0;
    double time_limit = 6.0;
    double c_const = dt/(dx*dx);
    double u_L = 0, u_R = 0;
    double a[N+2], b[N+1],c[N+1],alpha[N+2],beta[N+2],z[N+2],u[N+2],y[N+2];//クランクニコルソン法に必要な変数
    //初期化
    a[0] = 0; a[N+1] = 0; b[0] = 0; c[0] = 0;
    alpha[0] = 0; alpha[N+1] = 0; beta[0] = 0; beta[N+1] = 0; u[N+1] = 0;

    int step_num = (int)time_limit/dt;

    FILE *fp1_1_1;//ファイルに値を出力
    FILE *fp1_1_2;
    FILE *fp1_1_3;
    FILE *fp1_1_4;
    FILE *fp1_1_5;
    fp1_1_1= fopen("kadai1_3_1.txt", "w");
    fp1_1_2 = fopen("kadai1_3_2.txt", "w");
    fp1_1_3 = fopen("kadai1_3_3.txt", "w");
    fp1_1_4 = fopen("kadai1_3_4.txt", "w");
    fp1_1_5 = fopen("kadai1_3_5.txt", "w");

    //u の初期条件
    for(int j=1; j<=N; j++){
        u[j] = (u_0((j-1)*dx) + u_0(j*dx)) / 2;
    }

    //Ax=z における A から a,b,c を求める
    for(int j=1; j<=N; j++){
        a[j] = 1+c_const;
        b[j] = -c_const/2;
        c[j] = -c_const/2;
    }

    //LU 分解
    for(int j=1; j<=N; j++){
        alpha[j] = a[j] - c[j-1]*beta[j-1];
        beta[j] = b[j]/alpha[j];
    }
    beta[N] = 0.0;

    //更新
    for(int i=1; i<=step_num; i++){
        t += dt;
        //z を計算
        z[1] = (1-c_const)*u[1] + c_const*u_L + c_const*u[2]/2;
        for(int j=2; j<=N-1; j++){
            z[j] = (1-c_const)*u[j] + c_const*u[j-1]/2 + c_const*u[j+1]/2;
        }
        z[N] = (1-c_const)*u[N] + c_const*u_R + c_const*u[N-1]/2;

```

```c
        //y を計算
        for(int j=1; j<=N; j++){
            y[j] = (z[j]-c[j-1]*y[j-1])/alpha[j];
        }

        //u を計算
        for(int j=N; j>=1; j--){
            u[j] = y[j] - beta[j]*u[j+1];
        }

        //ファイル出力
        if(i==100){
            printf("%.8f", t);

            for(int j=1; j<=N; j++){
                double x = dx*((double)j-0.5);
                fprintf(fp1_1_1,"%.8f %.8f \n", x, u[j]);
            }

        }
        else if(i==200){
            printf("%.8f", t);
            for(int j=1; j<=N; j++){
                double x = dx*((double)j-0.5);
                fprintf(fp1_1_2,"%.8f %.8f \n", x, u[j]);
            }
        }
        else if(i==300){
            printf("%.8f", t);
            for(int j=1; j<=N; j++){
                double x = dx*((double)j-0.5);
                fprintf(fp1_1_3,"%.8f %.8f \n", x, u[j]);
            }
        }
        else if(i==400){
            printf("%.8f", t);
            for(int j=1; j<=N; j++){
                double x = dx*((double)j-0.5);
                fprintf(fp1_1_4,"%.8f %.8f \n", x, u[j]);
            }
        }
        else if(i==500){
            printf("%.8f", t);
            for(int j=1; j<=N; j++){
                double x = dx*((double)j-0.5);
                fprintf(fp1_1_5,"%.8f %.8f \n", x, u[j]);
            }
        }
    }

    fclose(fp1_1_1);
    fclose(fp1_1_2);
    fclose(fp1_1_3);
    fclose(fp1_1_4);
    fclose(fp1_1_5);
}
```

コードの説明を行う。コードの構造は先ほどまでのものと同じである。実行部では境界条件として Dirichlet 境界条件を用い、クランクニコルソン法を実行している。クランクニコルソン法について説明する。まず、初期条

件として $u_j^n(j=1...N)$ および境界条件 $u_L=u_R=0$ を設定する。以下、$c=\Delta t/\Delta x^2$ とする。次に、$\alpha_1=1+c, \beta_1=\frac{-c/2}{\alpha_1}, \alpha_2=a_2-c_1\beta_1..., \alpha_{N-1}=(1+c)-(-c/2)\beta_{N-2}, \beta_{N-1}=\frac{-c/2}{\alpha_{N-1}}, \alpha_N=a_N-(-c/2)\beta_{N-1}$ に従い α_j, β_j を計算する。次に時刻ごとに更新を行う。時刻 n において、$u_j^n(j=1...N)$ から z ベクトルを $z_1=(1-c)u_1^n+cu_L+\frac{cu_2^n}{2}, z_j=(1-c)u_j^n+\frac{cu_{j-1}^n}{2}+\frac{cu_{j+1}^n}{2}(2\leq j\leq N-1), z_N=(1-c)u_N^n+cu_R+\frac{cu_{N-1}^n}{2}$ に従って計算する。次に、$y_1=\frac{z_1}{\alpha_1}, y_2=\frac{z_2-c_1y_1}{\alpha_2}, ..., y_N=\frac{z_N-c_{N-1}y_{N-1}}{\alpha_N}$ に従って y ベクトルを計算し、$x_N=y_N, x_{N-1}=y_{N-1}-\beta_{N-1}x_N, ..., x_1=y_1-\beta_1x_2$ に従って x ベクトルを計算する。これをこの時刻における解とし、出力する。

4. ここでは、クランクニコルソン法および Neumann 境界条件を用いて解を求める。$\Delta t=0.01, \Delta x=0.05$ とした。計算の結果得られた出力 u の値は $x=5$ に関して対称であり、$x=4.25, 4.75$ の時が最も大きく、x の値がそれらから離れれば離れるほど小さくなって行った。以下が結果のグラフである。

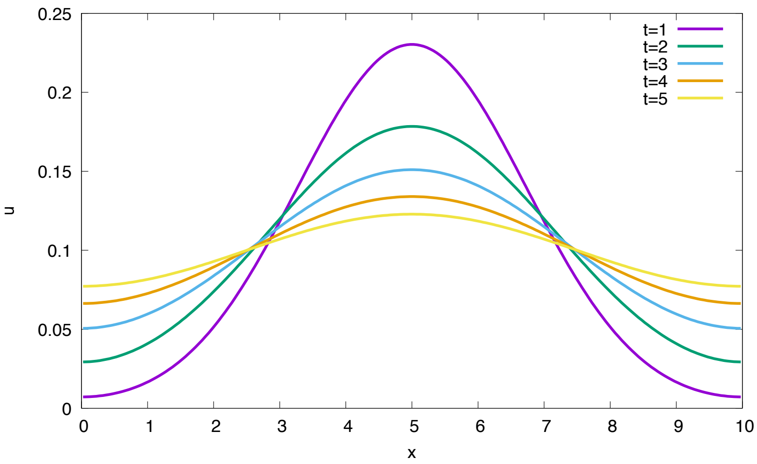

図 4: クランクニコルソン法、Neumann 境界条件を用いて解いた拡散方程式の解

コードは以下である。

コード 4: クランクニコルソン法、Neumann 境界条件を用いて拡散方程式を解くコード

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

//u の初期条件
double u_0(double x){
    return exp(-(x-5)*(x-5)/2) / sqrt(2*M_PI);
```

```
}

int main(){
    //ハイパーパラメータの定義
    int N = 200;
    double dx = 0.05;
    double dt = 0.01;
    double t = 0;
    double time_limit = 6.0;
    double c_const = dt/(dx*dx);
    double J_L = 0, J_R = 0;
    double a[N+2], b[N+1],c[N+1],alpha[N+2],beta[N+2],z[N+2],u[N+2],y[N+2];
    a[0] = 0; a[N+1] = 0; b[0] = 0; c[0] = 0;
    alpha[0] = 0; alpha[N+1] = 0; beta[0] = 0; beta[N+1] = 0; u[N+1] = 0;
    int step_num = (int)time_limit/dt;

    FILE *fp1_1_1;//ファイルに値を出力
    FILE *fp1_1_2;
    FILE *fp1_1_3;
    FILE *fp1_1_4;
    FILE *fp1_1_5;
    fp1_1_1= fopen("kadai1_4_1.txt", "w");
    fp1_1_2 = fopen("kadai1_4_2.txt", "w");
    fp1_1_3 = fopen("kadai1_4_3.txt", "w");
    fp1_1_4 = fopen("kadai1_4_4.txt", "w");
    fp1_1_5 = fopen("kadai1_4_5.txt", "w");

    //u の初期条件
    for(int j=1; j<=N; j++){
        u[j] = (u_0((j-1)*dx) + u_0(j*dx)) / 2;
    }

    //Ax=z における A から a,b,c を求める
    for(int j=1; j<=N; j++){
        a[j] = 1+c_const;
        b[j] = -c_const/2;
        c[j] = -c_const/2;
    }
    a[1] = 1+c_const/2;
    a[N] = 1+c_const/2;

    //LU 分解
    for(int j=1; j<=N; j++){
        alpha[j] = a[j] - c[j-1]*beta[j-1];
        beta[j] = b[j]/alpha[j];
    }
    beta[N] = 0.0;

    for(int i=1; i<=step_num; i++){
        t += dt;
        //z を計算
        z[1] = (1-c_const/2)*u[1] + c_const*J_L*dx + c_const*u[2]/2;
        for(int j=2; j<=N-1; j++){
            z[j] = (1-c_const)*u[j] + c_const*u[j-1]/2 + c_const*u[j+1]/2;
        }
        z[N] = (1-c_const/2)*u[N] + c_const*J_R*dx + c_const*u[N-1]/2;

        //y を計算
        for(int j=1; j<=N; j++){
```

```c
                y[j] = (z[j]-c[j-1]*y[j-1])/alpha[j];
            }

            //u を計算
            for(int j=N; j>=1; j--){
                u[j] = y[j] - beta[j]*u[j+1];
            }

            //ファイル出力
            if(i==100){
                printf("%.8f", t);
                for(int j=1; j<=N; j++){
                    double x = dx*((double)j-0.5);
                    fprintf(fp1_1_1,"%.8f %.8f \n", x, u[j]);
                }
            }
            else if(i==200){
                printf("%.8f", t);
                for(int j=1; j<=N; j++){
                    double x = dx*((double)j-0.5);
                    fprintf(fp1_1_2,"%.8f %.8f \n", x, u[j]);
                }
            }
            else if(i==300){
                printf("%.8f", t);
                for(int j=1; j<=N; j++){
                    double x = dx*((double)j-0.5);
                    fprintf(fp1_1_3,"%.8f %.8f \n", x, u[j]);
                }
            }
            else if(i==400){
                printf("%.8f", t);
                for(int j=1; j<=N; j++){
                    double x = dx*((double)j-0.5);
                    fprintf(fp1_1_4,"%.8f %.8f \n", x, u[j]);
                }
            }
            else if(i==500){
                printf("%.8f", t);
                for(int j=1; j<=N; j++){
                    double x = dx*((double)j-0.5);
                    fprintf(fp1_1_5,"%.8f %.8f \n", x, u[j]);
                }
            }
        }

        fclose(fp1_1_1);
        fclose(fp1_1_2);
        fclose(fp1_1_3);
        fclose(fp1_1_4);
        fclose(fp1_1_5);
}
```

コードの説明を行う。

構造は Dirichlet 境界条件を用いたコードと同じである。相違点は、境界条件として Neumann 境界条件を用いているため、境界の初期条件を $J_L = J_R = 0$ としている点と α, β を $\alpha_1 = 1 + c/2, \beta_1 = \frac{-c/2}{\alpha_1}, ..., \alpha_{N-1} = (1 + c/2) - (-c/2)\beta_{N-2}, \beta_{N-1} = \frac{-c/2}{\alpha_{N-1}}, \alpha_N = a_N - (-c/2)\beta_{N-1}$ と更新している点、そして z ベクトルを $z_1 = (1-c)u_1^n + cu_L + \frac{cu_2^n}{2}, z_j = (1-c/2)u_j^n + \frac{cu_{j-1}^n}{2} + \frac{cu_{j+1}^n}{2}(2 \leq$

$j \leq N-1), z_N = (1-c/2)u_N^n + cu_R + \frac{cu_{N-1}^n}{2}$ に従って計算する点である。

以上 1 から 4 の結果より、この拡散方程式の解は　$x = 5$ に関して対称な分布となることが視覚的にもわかる。また、Delichlet 境界条件を用いると境界における u の値が 0 に近くなり、Neumann 境界条件を用いると t の値により異なる境界の値が得られることがわかる。さらに、t の値が大きいほど x=5 における u の値が小さくなることがわかる。また、$x = 5$ において、t が大きいほど解の値は小さくなることがわかる。

3　問題 2

ここでは、初期条件 $u_0(x) = \frac{1}{(1+\exp(bx-5))^2}$、境界条件 $u(0,t) = 1, u(L,t) = 0(L = 200)$ のもとで、$b = 0.25, 0.5, 1.0$ の場合に Fisher 方程式をオイラー陽解法 $\frac{u_j^{n+1}-u_j^n}{\Delta t} = f(u_j^n) + \frac{u_{j-1}^n-2u_j^n+u_{j+1}^n}{\Delta x^2}$ を用いて解く。刻み幅は $\Delta x = 0.05, \Delta t = 0.001$ とし、$t = 10, 20, 30, 40$ の $u(x,t)$ の値を出力する。また、$u(x,t)$ の値を b の値ごとに横軸 x、縦軸 u の図に出力し、得られた結果について考察する。なお、以下 u_j^n は座標 x, 時刻 t($n = t/\Delta t, j = x/\Delta x$、ただし Δt は時間の刻み幅、Δx は座標の刻み幅) における方程式の解とする。
得られた出力は初期条件 $u = 1$ に始まり、x の値が大きくなるにつれて単調減少するものであった。
得られた図は以下である。

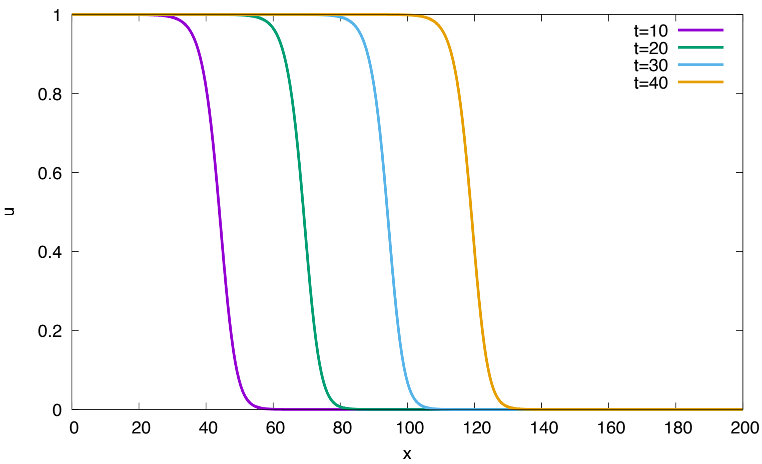

図 5: b=0.25 の時

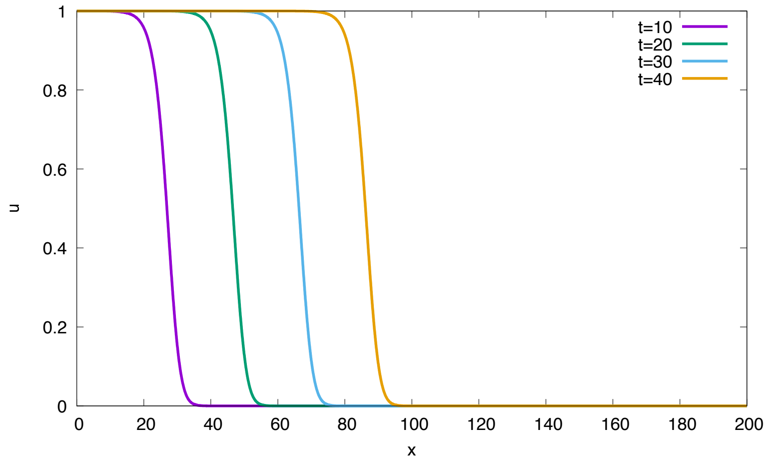

図 6: b=0.5 の時

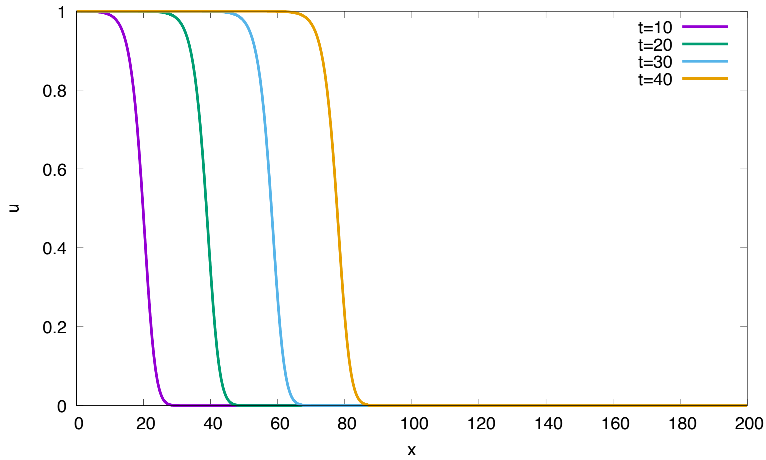

図 7: b=1.0 の時

コードは以下である。

コード 5: Fisher 方程式を解くコード

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

double u_0(double x, double b){
    return 1/((1+exp(b*x-5))*(1+exp(b*x-5)));
}
```

```
double f(double u_){
    return u_*(1-u_);
}

int main(){
    //ハイパーパラメータの定義
    int N = 4000;
    double dt = 0.001;//時刻の刻み幅
    double dx = 0.05;//座標の刻み幅
    double b;
    double time_limit = 50.0;
    double step_num = (int)time_limit/dt;//繰り返し回数

    for(int b_=0; b_<=2; b_++){
        FILE *fp1_1_1;//ファイルに値を出力
        FILE *fp1_1_2;
        FILE *fp1_1_3;
        FILE *fp1_1_4;
        double u[N+2], u_tmp[N+2];
        double t = 0;

        if(b_==0){
            b = 0.25;
            fp1_1_1= fopen("kadai2_b025_10.txt", "w");
            fp1_1_2 = fopen("kadai2_b025_20.txt", "w");
            fp1_1_3 = fopen("kadai2_b025_30.txt", "w");
            fp1_1_4 = fopen("kadai2_b025_40.txt", "w");
            printf("%d ",b_);
        }
        if(b_==1){
            b = 0.5;
            fp1_1_1= fopen("kadai2_b05_10.txt", "w");
            fp1_1_2 = fopen("kadai2_b05_20.txt", "w");
            fp1_1_3 = fopen("kadai2_b05_30.txt", "w");
            fp1_1_4 = fopen("kadai2_b05_40.txt", "w");
            printf("%d ",b_);
        }
        if(b_==2){
            b = 1.0;
            fp1_1_1= fopen("kadai2_b10_10.txt", "w");
            fp1_1_2 = fopen("kadai2_b10_20.txt", "w");
            fp1_1_3 = fopen("kadai2_b10_30.txt", "w");
            fp1_1_4 = fopen("kadai2_b10_40.txt", "w");
            printf("%d ",b_);
        }
        //境界条件
        double u_L = 1, u_R = 0;
        u[0] = u_L; u[N+1] = u_R; u_tmp[0] = u_L; u_tmp[N+1] = u_R;

        //初期条件
        for(int j=1; j<=N; j++){
            u[j] = (u_0((j-1)*dx,b) + u_0(j*dx,b)) / 2;
            u_tmp[j] = 0;
        }

        //オイラー法
        for(int i=1; i<=step_num; i++){
            t += dt;
            for(int j=1; j<=N; j++){
                u_tmp[j] = u[j] + (f(u[j])+(u[j-1]-2*u[j]+u[j+1])/(dx*dx))*dt;
            }
```

```
            for(int j=1; j<=N; j++){
                u[j] = u_tmp[j];
            }

            if(i==10000){
            printf("%.8f ",t);
                for(int j=1; j<=N; j++){
                    double x = dx*((double)j-0.5);
                    fprintf(fp1_1_1,"%.8f %.8f \n", x, u[j]);
                }
            }
            else if(i==20000){
            printf("%.8f ",t);
                for(int j=1; j<=N; j++){
                    double x = dx*((double)j-0.5);
                    fprintf(fp1_1_2,"%.8f %.8f \n", x, u[j]);
                }
            }
            else if(i==30000){
            printf("%.8f ",t);
                for(int j=1; j<=N; j++){
                    double x = dx*((double)j-0.5);
                    fprintf(fp1_1_3,"%.8f %.8f \n", x, u[j]);
                }
            }
            else if(i==40000){
            printf("%.8f ",t);
                for(int j=1; j<=N; j++){
                    double x = dx*((double)j-0.5);
                    fprintf(fp1_1_4,"%.8f %.8f \n", x, u[j]);
                }
            }
        }
        fclose(fp1_1_1);
        fclose(fp1_1_2);
        fclose(fp1_1_3);
        fclose(fp1_1_4);
    }
}
```

コードの説明を行う。まず、時刻の刻み幅や座標の刻み幅、繰り返し回数などの設定を行う。実行部分では、3 種類の b の値それぞれに対し、同じ回数だけオイラー陽解法で解の更新を行う。$t = 10, 20, 30, 40, 50$ の時には、設定されている b の値およびその時の t の値に応じたファイルに、各座標における x の値を出力する。

結果の考察を行う。まず、どの t,b においても、x の値が大きくなるにつれて u の値が単調減少していく。加えて、どの b においても t の値が大きいほど減衰のはじまる x の値が大きくなり、また u が 0 に収束する x の値も大きくなる。また、b の値が大きくなるほど u(x) が減衰し始める x の値が小さくなる。さらに、b の値が小さくなるほど、t=10,20,30,40 において同じ u の値を与える x の間隔が広くなる。

4　問題 3

ここでは、シュレディンガー方程式 $i\frac{\partial\psi}{\partial t} = (-\frac{1}{2}\frac{\partial^2}{\partial x^2} + \frac{x^2}{2})\psi(x,t)$ を、初期条件 $\psi(x,0) = \frac{\sqrt{2}\exp(-2(x-5)^2)}{\pi^{\frac{1}{4}}}$ のもとで数値的にとく。その際、空間領域は [-10,10] とした。座標 x, 時刻 t($n = t/\Delta t, j = x/\Delta x$、ただし Δt は時間の刻み幅、Δx は座標の刻み幅) における波動関数の実部、虚部をそれぞれ R_j^n, I_j^n とおき、$1 \leq j \leq N(N\Delta x = 20)$ に対し $R_j^{n+1} = R_j^n + \Delta t(-\frac{I_{j-1}^n - 2I_j^n + I_{j+1}^n}{2\Delta x^2} + \frac{x_j^2 I_j^n}{2})$、$I_j^{n+1} = R_j^n - \Delta t(-\frac{R_{j-1}^{n+1} - 2R_j^{n+1} + R_{j+1}^{n+1}}{2\Delta x^2} + \frac{x_j^2 I_j^{n+1}}{2})$ の式に従って更新を行う。ただし $x_j = (j-\frac{1}{2})\Delta x - 10$ とし、境界条件は全ての n において $R_0^n = R_N^n, R_{N+1}^n = R_1^n, I_0^n = I_N^n, I_{N+1}^n = I_1^n$

と表す。さらに、初期条件は $1 \leq j \leq N$ で $R_j^0 = \frac{1}{2}[\psi((j-1)\Delta x - a)]$、

$-\Delta t(-\frac{R_{j-1}^0 - 2R_j^0 + R_{j+1}^0}{2\Delta x^2} + \frac{x_j^2 R_j^0}{2})$ とす

る。これらの条件下において、$\Delta x = 0.05, \Delta t = 0.001$ として系を数値的に解き、$t = 1,2,3,4,5,6,7,8$ における確率密度 $(R_j^n)^2 + I_j^n I_j^{n-1}(1 \leq j \leq N)$ の値を出力した。また、得られた確率密度 (以下 u とおく) を一つのグラフとして出力した。結果、それぞれの t において、途中まで単調増加し、途中からは単調減少する数値の出力が得られた。t により、どこで u がピークに達するかは異なっていた。

出力された数値を横軸 x、縦軸 u としてプロットすると、それぞれの t に対し左右対称な確率密度関数が得られた。以下が得られたグラフである。

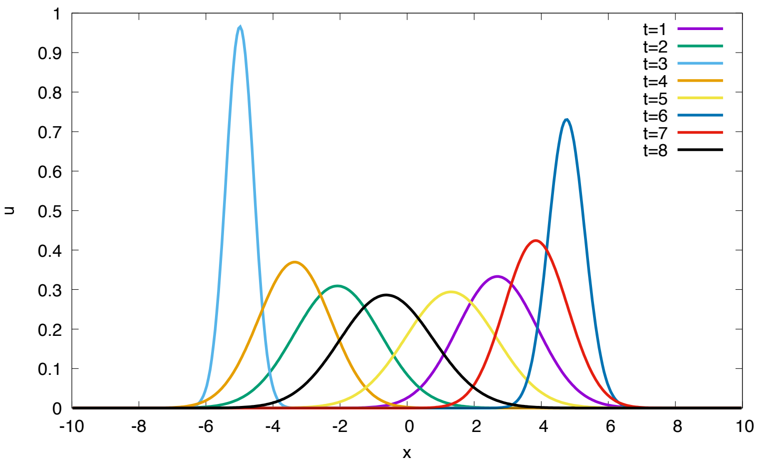

図 8: シュレディンガー方程式の解. 縦軸が確率密度 u, 横軸が x

以下がコードである。

コード 6: シュレディンガー方程式を解き確立密度を求めるコード

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

```c
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>

//Ψ の定義
double psai(double x){
    return sqrt(2)*exp(-2*(x-5)*(x-5)) / pow(M_PI, 1.0/4);
}

int main(){
    //ハイパーパラメータ
    int N = 400;
    double dx = 0.05;
    double dt = 0.001;
    double R[N+2], I[N+2], I_kari[N+2],x[N+2];
    double time_limit = 10;
    int step_num = (int)time_limit/dt;
    double L = 20;
    double t=0;
    FILE *fp;

    //を定義 x
    for(int i=1; i<=N; i++){
        x[i] = (i-1/2)*dx - L/2;
    }
    //R の初期条件
    for(int i=1; i<=N; i++){
        R[i] = (psai((i-1)*dx-L/2) + psai(i*dx-L/2)) / 2;
    }
    //R の境界を初期化
    R[0] = R[N];
    R[N+1] = R[1];

    //I の初期条件
    for(int i=1; i<=N; i++){
        I[i] = -dt * (-(R[i-1] - 2*R[i] + R[i+1])/(2*dx*dx) + x[i]*x[i]*
            R[i]/2);
    }

    //I の境界を初期化
    I[0] = I[N];
    I[N+1] = I[1];

    //出力対象の t によって出力ファイルを変更する
    for(int i=1; i<=8; i++){
        //ファイルを開く
        if(i==1) fp = fopen("kadai3_1.txt","w");
        if(i==2) fp = fopen("kadai3_2.txt","w");
        if(i==3) fp = fopen("kadai3_3.txt","w");
        if(i==4) fp = fopen("kadai3_4.txt","w");
        if(i==5) fp = fopen("kadai3_5.txt","w");
        if(i==6) fp = fopen("kadai3_6.txt","w");
        if(i==7) fp = fopen("kadai3_7.txt","w");
        if(i==8) fp = fopen("kadai3_8.txt","w");

        //次の出力対象の t まで更新を行う
        for(int k=1; k<=1000; k++){
            t += dt;
            //R の更新
            for(int j=1; j<=N; j++){
                R[j] = R[j] + dt*(-(I[j-1]-2*I[j]+I[j+1]) / (2*dx*dx) +
                    x[j]*x[j]*I[j]/2);
            }
```

```
            //R の境界を更新
            R[0] = R[N];
            R[N+1] = R[1];
            //I の更新確率密度を求めるのに一つ前の時刻の.I が必要なので
                に格納 I_kari.
            for(int j=1; j<=N; j++){
                I_kari[j] = I[j];
                I[j] = I[j] - dt*(-(R[j-1]-2*R[j]+R[j+1]) / (2*dx*dx)
                    + x[j]*x[j]*R[j]/2);
            }
            //I の境界を更新
            I[0] = I[N];
            I[N+1] = I[1];
        }
        //出力
        for(int j=1; j<=N; j++){
            double X = dx*((double)j-0.5)-L/2;
            double mitudo = R[j]*R[j] + I[j]*I_kari[j];//確率密度
            fprintf(fp,"%.8f %.8f \n", X, mitudo);
        }
        fclose(fp);
    }
}
```

コードの説明を行う。まず、x に対して $\psi(x,0) = \frac{\sqrt{2}\exp(-2(x-5)^2)}{\pi^{\frac{1}{4}}}$ を返す関数 psai を定義する。実行部では、まず x を定義し、順に配列 x に格納していく。次に R_1 R_N を求めて配列に格納し、R の境界条件を求める。その後 I_1 から I_N を求め、I の境界条件を求めてそれぞれ配列に格納する。その後は上述した方法によって R と I および R と I の境界条件の更新を行い、更新が 1000 回行われるごとに、その時の確率分布をファイルに出力する。なお、確率分布を求める際に一つ前の時刻の I の値が必要となるため、これを I_{kari} という配列に格納しておき、利用する。

5　まとめ

このレポートでは、問題 1 で拡散方程式 $\frac{\partial u(x,t)}{\partial t} = \frac{\partial^2 u(x,t)}{\partial x^2}...(*)$ を、初期条件 $u_0 = \frac{\exp(\frac{-(x-5)^2}{2})}{\sqrt{2\pi}}$ のもとで解いた。この際、境界条件として Dirichlet 境界条件および Neumann 境界条件を、手法としてオイラー陽解法およびクランクニコルソン法を用いた。結果、u の値は $x = 5$ に関して対称であり、x の値が山の中央から離れれば離れるほど小さくなって行った。さらに、Delichlet 境界条件を用いると境界における u の値が 0 に近くなり Neumann 境界条件を用いると t の値により異なる境界の値が得られること、そして t の値が大きいほど x=5 における u の値が小さくなることがわかった。問題 2 では、初期条件 $u_0(x) = \frac{1}{(1+\exp(bx-5))^2}$、境界条件 $u(0,t) = 1, u(L,t) = 0 (L = 200)$ のもとで、$b = 0.25, 0.5, 1.0$ の場合に Fisher 方程式をオイラー陽解法 $\frac{u_j^{n+1} - u_j^n}{\Delta t} = f(u_j^n) + \frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2}$ を用いて解き、b の値ごとに $t = 10, 20, 30, 40$ の時の u の値を出力した。結果、どの t,b においても、x の値が大きくなるにつ

れて u の値が単調減少していくこと、どの b においても t の値が大きいほど減衰のはじまる x の値が大きくなり、また u が 0 に収束する x の値も大きくなること、b の値が大きくなるほど u(x) が減衰し始める x の値が小さくなること、b の値が小さくなるほど t=10,20,30,40 において同じ u の値を与える x の間隔が広くなることがわかった。問題 3 では、シュレディンガー方程式 $i\frac{\partial\psi}{\partial t}=(-\frac{1}{2}\frac{\partial^2}{\partial x^2}+\frac{x^2}{2})\psi(x,t)$ を、初期条件 $\psi(x,0)=\frac{\sqrt{2}\exp(-2(x-5)^2)}{\pi^{\frac{1}{4}}}$ のもとで数値的に解き、$t=1,2,3,4,5,6,7,8$ における解を出力、プロットした。結果、それぞれの t に対し左右対称な確率密度関数が得られた。また、対称軸となる x の値は t によって異なった。