プログラミング演習（2023）

2023.04.28 田中

0.0.　この演習の目的

Matlab を使って波動方程式を差分法で解き，結果を可視化する．

0.1.　Matlab の基本的な使い方

Matlab でプログラミングする方法は主に下記の二つです．
・コマンドウィンドウに直接コードを書く．
・スプリクトファイルにコードを書き込む．
将来的には後者を使うことが多いと思います．

基本的な操作はこのサイトを見ればほぼ問題ないと思います．
他の操作もインターネットで「matlab　＊＊」と検索すればたいてい出てきます．

1.0.　1 階常微分方程式

まずは，常微分方程式

$$\frac{\mathrm{d}y}{\mathrm{d}x} = -y$$

を，条件

$$y(x = 0) = 1$$

で

$$0 \leq x \leq 1$$

の区間で解きます．

これをコンピューターで(Matlab を使って)解くためには，微分方程式を”離散化”する必要があります．離散化について次の節で説明します．

1.1. 微分方程式の離散化（差分法）

コンピューターは離散的なデータしか記憶できません．微分方程式を数値的に解くとき，記憶させるのは，飛び飛びの座標 $x=\{x_1,x_2,\cdots,x_i,\cdots,x_{N+1}\}$ と，その座標における変数の値 $y=\{y_1,y_2,\cdots,y_i,\cdots,y_{N+1}\}$ です．間の座標における値が必要な場合は適当な関数で補間します．

微分方程式を離散化する方法はいくつか(有限要素法，有限体積法など)ありますが，ここでは差分法を紹介します．（他の手法でも本質的にはほぼ変わりません．）

差分法(１次精度)では１階微分を

$$\frac{\mathrm{d}y}{\mathrm{d}x}\approx\frac{y(x+\Delta x)-y(x)}{\Delta x}\quad \text{あるいは}\quad \frac{\mathrm{d}y}{\mathrm{d}x}\approx\frac{y(x)-y(x-\Delta x)}{\Delta x}$$

のように近似します．
そのために，まず，区間 $0<x<1$ を N 等分して

$$\Delta x=\frac{1}{N},\qquad x_i=i\Delta x,\qquad y_i=y(x_i)$$

としてみます．このとき差分法で近似した方程式は

$$\frac{y_{i+1}-y_i}{\Delta x}=-y_i\quad \text{あるいは}\quad \frac{y_i-y_{i-1}}{\Delta x}=-y_i$$

となります．例えば左の式を使うならば，

$$y_{i+1}=(1-\Delta x)y_i$$

というように y_i の値を使ってお隣の y_{i+1} の値を求めることができます．

実際に計算しましょう．Matlab でコードを書いてみると以下のような感じになります．

```
% ←これを書いた後ろ(緑字部分)は実行されない

% プログラムが開始したら画面に「始まった!!」と出す
disp('始まった!!');


%////////////////////////////////////////////////////////
% 前世の記憶を削除(これまでの計算が影響しないようにワークスペースを初期化)
%////////////////////////////////////////////////////////
clear;
```

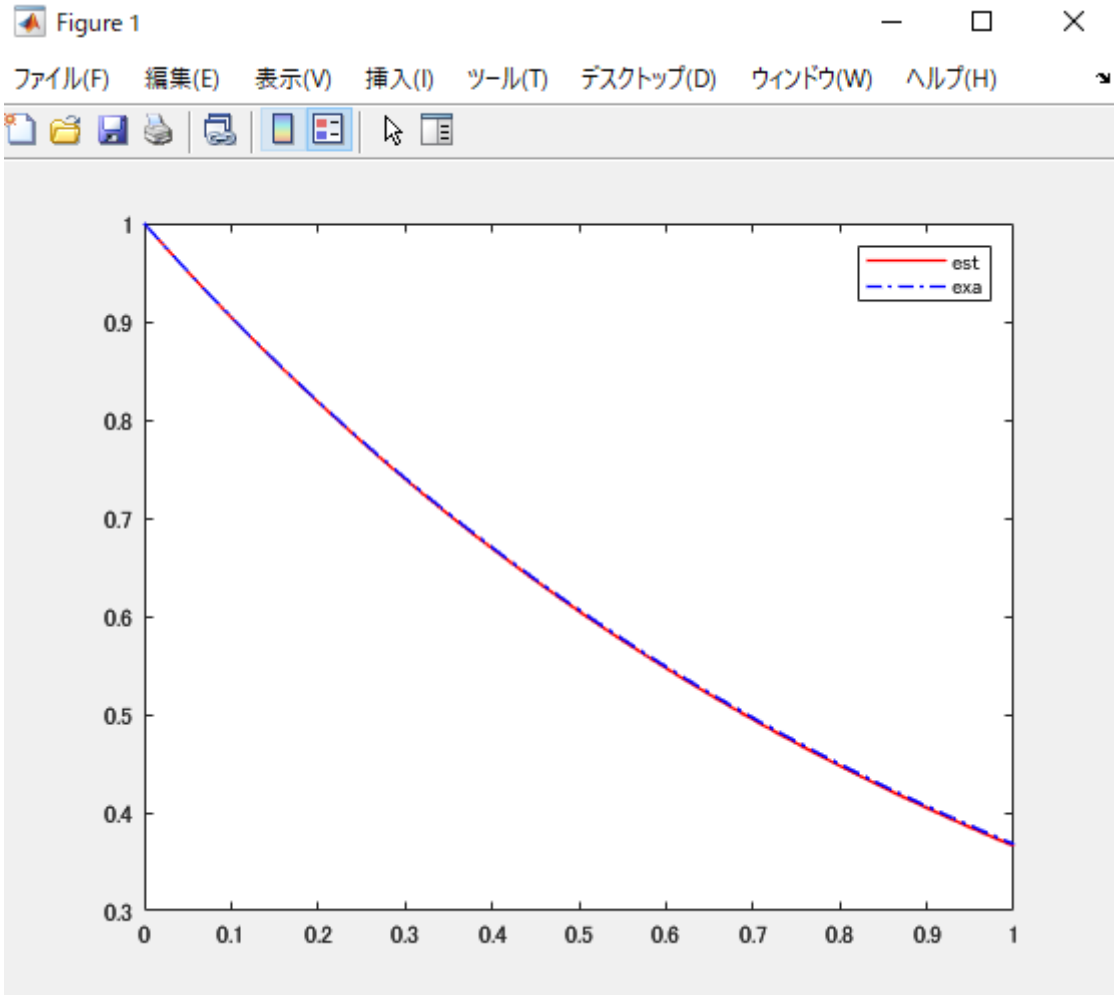
```
%//////////////////////////////////////////////////////////
% 配列(記憶してほしい値の入れ物)を用意する(用意しなくてもできる)
%//////////////////////////////////////////////////////////

N=100;   % 区間の分割数
dx=1/N;  % 区間の分割幅

% 101×1 の配列(中身はすべて 0)を定義している
x=zeros(101,1);  % x の配列
y=zeros(101,1);  % y の配列
% 配列の要素には1から順番に番号がついている


%//////////////////////////////////////////////////////////
% x と y(x=0)の値を配列に格納する
%//////////////////////////////////////////////////////////

% for 文-----------------------------------------------------
% i=1~N+1 まで for と end の間に書いた処理を繰り返す
%-----------------------------------------------------------

for i=1:N+1
    x(i)=(i-1)*dx; %さっき用意した配列 x の i 番目に(i-1)*dx が格納される
end

y(1)=1;


%//////////////////////////////////////////////////////////
% 差分法で微分方程式を解く
%//////////////////////////////////////////////////////////

for i=2:N+1
    y(i)=(1-dx)*y(i-1);
end
```

```
%//////////////////////////////////////////////////////////
% 結果をグラフにする
%//////////////////////////////////////////////////////////

% 差分法で計算した結果をプロット
plot(x(:),y(:),'-','Color','r','LineWidth',1);

% plot コマンド-------------------------------------------------
% x(:)で配列 x すべてを x 軸に指定
% x(1:50)などとすれば一部だけ指定することも可能

% '-'で線の形を実線に指定，'Color','r'で色を赤に指定，
% 'LineWidth',1 で線の太さを 1 に指定

% 他にもいろいろオプションがある
%-------------------------------------------------------------

% 解析解もプロットしてみる
for i=1:N+1
    z(i)=exp(-x(i));
    % z 配列を先に宣言しなかったのでこの処理をするたびに z のサイズが変わる
end
hold on; % hold on で同じグラフにプロット
plot(x(:),z(:),'-.','Color','b','LineWidth',1);

legend('est','exa'); %凡例を plot した順につける

% x=1 での計算誤差を画面に表示させてみる
error=y(101)-z(101);
disp('x=1 での誤差は');
disp(error);


% プログラムが終了したら画面に「終わった!!」と出す
disp('終わった!!');
```

これを実行するとコマンドウィンドウには以下のように出力されます.

```
>> test1
始まった!!
x=1での誤差は
   -0.0018

終わった!!
```

また，Figure1 というウィンドウに以下のようなグラフが表示されます.



2.0.　2 階常微分方程式(その 1)

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} = -y$$

を，条件

$$y(x=0)=1, \qquad \frac{\mathrm{d}y}{\mathrm{d}x}(x=0)=0$$

で

$$0 \leq x \leq 2\pi$$

の区間で解きます.

この場合は，連立微分方程式

$$\frac{\mathrm{d}y}{\mathrm{d}x}=z, \qquad \frac{\mathrm{d}z}{\mathrm{d}x}=-y$$

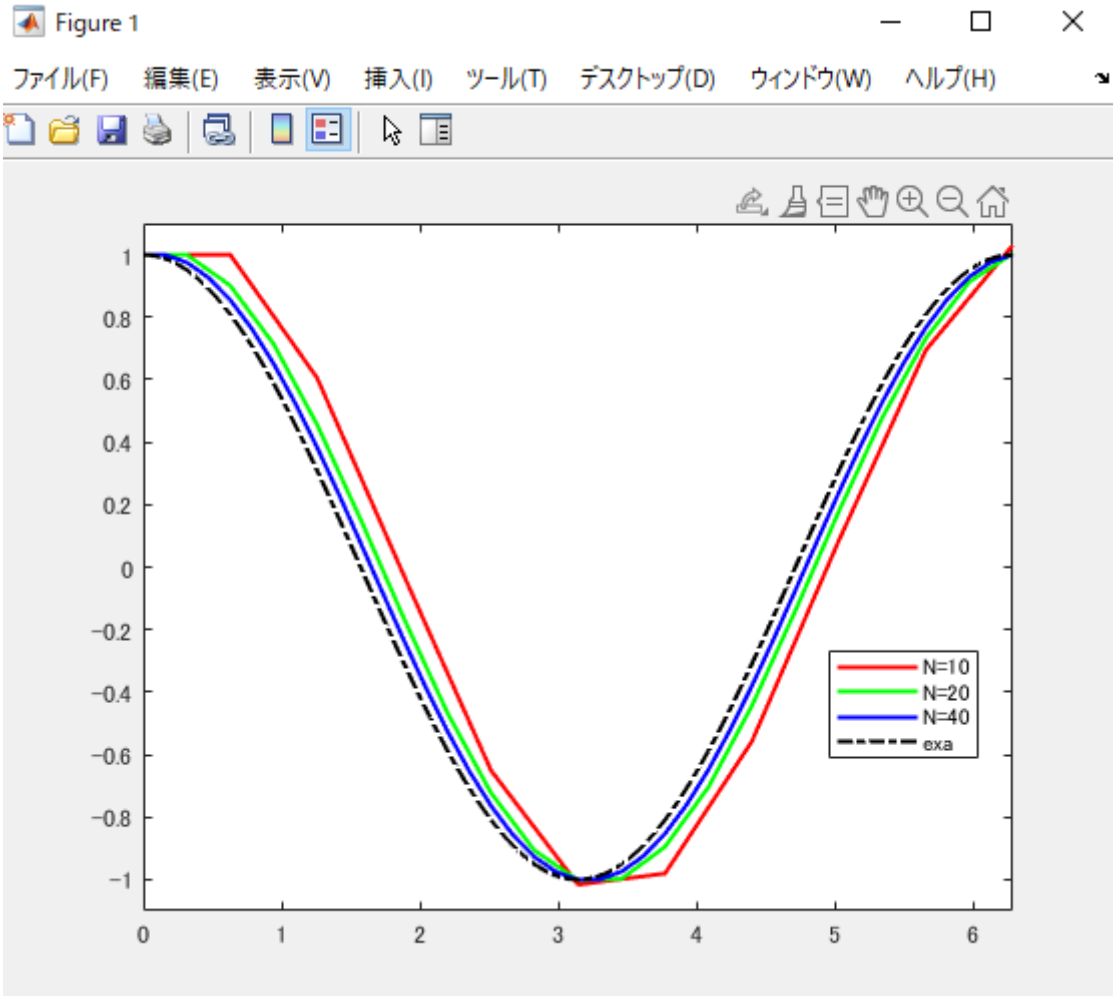
を解けばよいので，1.2. と同じく 1 階微分を差分法で近似して解くことができます.

コードを書いてみると以下のような感じになります.（分割数を 3 パターン用意して計算しています.）

```
% プログラムが開始したら画面にメッセージを出す
disp('2 階常微分方程式(その 1)が始まった!!');

clear;

% 配列は for ループの前に宣言. ループ内で毎回サイズが変わると計算時間が増える.
x=zeros(41,3);
y=zeros(41,3);
z=zeros(41,3);

for j=1:3
    N=10*2^(j-1); % 分割数を 10,20,40 と変えてみる
    dx=2*pi/N;  % 区間の分割幅

    % x と y(x=0),z(x=0)の値を配列に格納する
    for i=1:N+1
        x(i,j)=(i-1)*dx;
    end
    y(1,j)=1;
    z(1,j)=0;

    % 差分法で微分方程式を解く
    for i=1:N
```

```
        y(i+1,j)=y(i,j)+dx*z(i,j);
        z(i+1,j)=z(i,j)-dx*y(i+1,j);
    end

    % 計算結果をプロット
    if j==1
        clf('reset'); % figure をリセット
        plot(x(1:N+1,j),y(1:N+1,j),'-','Color','r','LineWidth',1.5);
        hold on;
    elseif j==2
        plot(x(1:N+1,j),y(1:N+1,j),'-','Color','g','LineWidth',1.5);
    else
        plot(x(1:N+1,j),y(1:N+1,j),'-','Color','b','LineWidth',1.5);
    end

end

% 解析解も
x_exa=zeros(101,1);
y_exa=zeros(101,1);
for i=1:101
    x_exa(i)=(i-1)*2*pi/100;
    y_exa(i)=cos(x_exa(i));
end
plot(x_exa(:),y_exa(:),'-.','Color','k','LineWidth',1.5);

% グラフの表示設定
legend('N=10','N=20','N=40','exa','Location','best');
xlim([0 2*pi]);   % x 軸の表示範囲
ylim([-1.1 1.1]); % y 軸の表示範囲

% プログラムが終了したら画面にメッセージと出す
disp('終わった!!');
```

グラフは以下のようなものが出力されます.



当然分割数 N を大きくするほど計算量は増えますが，計算結果が解析解に近づきます．

2.1. 2 階常微分方程式(その 2)

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} = -y$$

を，条件

$$y(x=0) = y_1 = 1, \qquad y\left(x=\frac{\pi}{2}\right) = y_{N+1} = 1$$

で

$$0 \le x \le \frac{\pi}{2}$$

の区間で解きます．

この場合は差分法で 2 階微分を

$$\frac{\mathrm{d}^2y}{\mathrm{d}x^2} \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{(\Delta x)^2}$$

のように近似します.

これを $i = 1, \cdots, N$ で連立すると,

$$\begin{bmatrix} 2-\Delta x^2 & -1 & & & & \\ -1 & 2-\Delta x^2 & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & -1 & 2-\Delta x^2 & -1 & \\ & & & -1 & 2-\Delta x^2 & \end{bmatrix} \begin{bmatrix} y_2 \\ y_3 \\ \vdots \\ y_{N-1} \\ y_N \end{bmatrix} = \begin{bmatrix} y_1 \\ 0 \\ \vdots \\ 0 \\ y_{N+1} \end{bmatrix}$$

となります. この連立方程式を自力で解いても良いですが, 今回は Matlab に解いてもらうことにします.

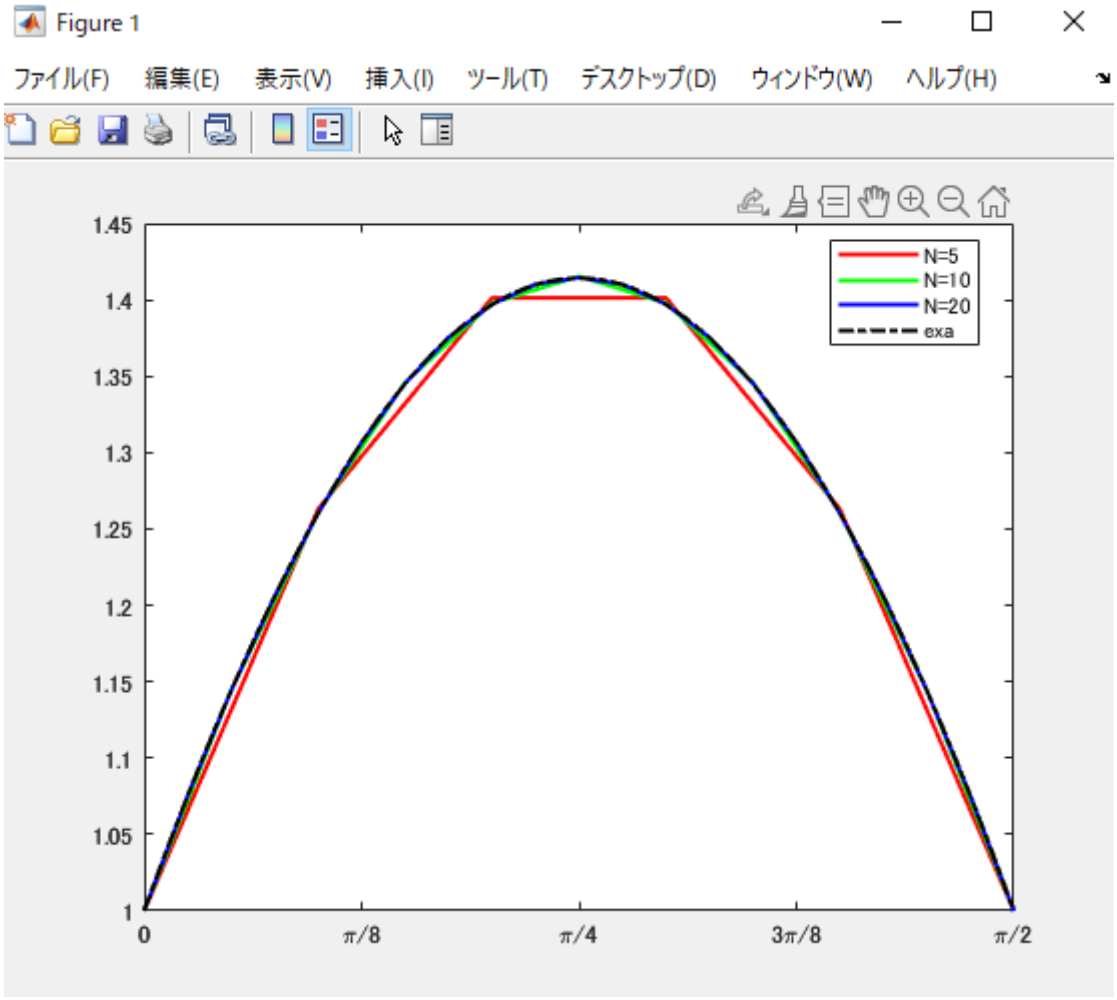
コードを書いてみると以下のような感じになります.

```
% プログラムが開始したら画面にメッセージを出す
% (プログラム要所でメッセージを出せばバグがあったときに分かりやすい)
disp('2 階常微分方程式(その 2)が始まった!!');

clear;

% 配列は for ループの前に宣言. ループ内で毎回サイズが変わると計算時間が増える.
x=zeros(41,3);
y=zeros(41,3);

for j=1:3
    N=5*2^(j-1); % 分割数を 5,10,20 と変えてみる
    dx=pi/2/N;  % 区間の分割幅

    A=zeros(N-1,N-1); % 係数行列を宣言
    b=zeros(N-1,1);   % 右辺ベクトルを宣言

    % x と y(x=0),y(x=1)の値を配列に格納する
    for i=1:N+1
        x(i,j)=(i-1)*dx;
    end
    y(1,j)=1;
    y(N+1,j)=1;
```

```
    % 係数行列，右辺ベクトルの値を格納する
    A(1,1)=2-dx^2; A(1,2)=-1; b(1)=y(1,j);
    for i=2:N-2
        A(i,i-1)=-1; A(i,i)=2-dx^2; A(i,i+1)=-1;
    end
    A(N-1,N-2)=-1; A(N-1,N-1)=2-dx^2; b(N-1)=y(N+1,j);

    % Matlab に連立方程式を解かせる(部分ピボットを使った LU 分解で解いている)
    y(2:N,j)=linsolve(A,b);
    % LU 分解についてはこちらを参照
    % 部分ピボットを使った LU 分解についてはこちらを参照

    % 計算結果をプロット
    if j==1
        clf('reset'); %figure をリセット
        plot(x(1:N+1,j),y(1:N+1,j),'-','Color','r','LineWidth',1.5);
        hold on;
    elseif j==2
        plot(x(1:N+1,j),y(1:N+1,j),'-','Color','g','LineWidth',1.5);
    else
        plot(x(1:N+1,j),y(1:N+1,j),'-','Color','b','LineWidth',1.5);
    end

end

% 解析解も
x_exa=zeros(101,1);
y_exa=zeros(101,1);
for i=1:101
    x_exa(i)=(i-1)*pi/2/100;
    y_exa(i)=sin(x_exa(i))+cos(x_exa(i));
end
plot(x_exa(:),y_exa(:),'-.','Color','k','LineWidth',1.5);

% % グラフの表示設定
```

```
legend('N=5','N=10','N=20','exa','Location','best');
xlim([0 pi/2]); % x 軸の表示範囲
ylim([1 1.45]); % y 軸の表示範囲
xticks([0 pi/8 pi/4 pi*3/8 pi/2]); % x 軸の目盛
xticklabels({'0','¥pi/8','¥pi/4','3¥pi/8','¥pi/2'}); % x 軸の目盛

% プログラムが終了したら画面にメッセージと出す
disp('終わった!!');
```

グラフは以下のようなものが出力されます.



3.0. 線形移流方程式(陽解法)

1 次元線形移流方程式

$$\frac{\partial u}{\partial t}(x,t) + c\frac{\partial u}{\partial x}(x,t) = 0 \qquad (c > 0)$$

を初期条件

$$u(x > 0, t = 0) = 0$$

と境界条件

$$u(x = 0, t) = 1$$

で区間

$$0 \le x \le 1, \qquad t \ge 0$$

で解きます.

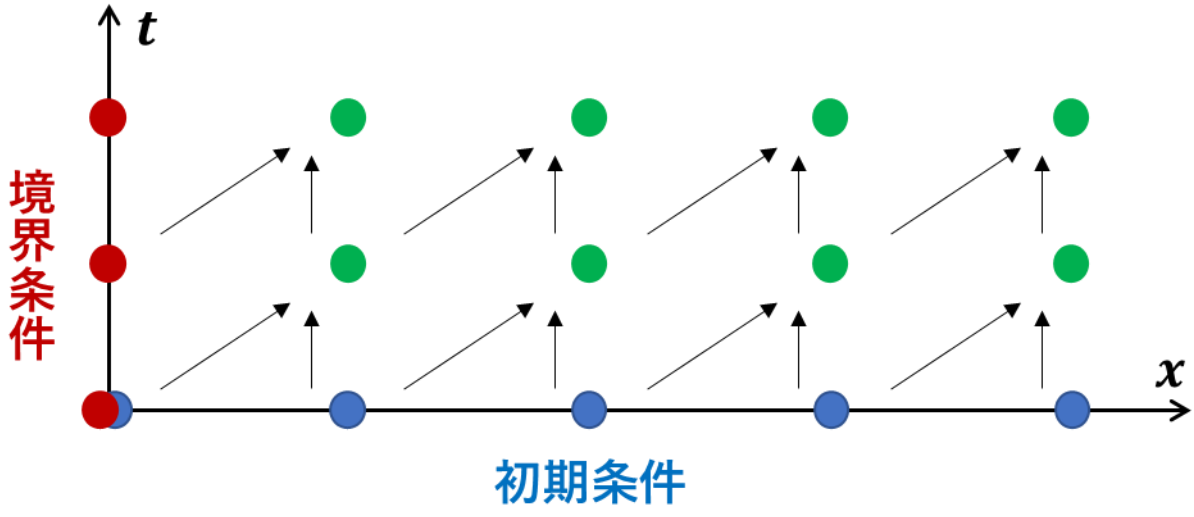
まずは時間 t と空間 x を離散化します.

$$t^n = n\Delta t, \qquad x_i = i\Delta x \ \ (i = 1, \cdots, N+1)$$

そして, 差分近似を以下のようにとってみます.

$$\frac{\partial u}{\partial t}(x,t) \approx \frac{u(x,t+\Delta t) - u(x,t)}{\Delta t}, \qquad \frac{\partial u}{\partial x}(x,t) \approx \frac{u(x,t) - u(x-\Delta x,t)}{\Delta x}$$

これは図で考えてみると以下のようになります.



初期値 $u_i^0\ (i = 1, \cdots, N+1)$ (と境界条件 u_1^0) を使って $u_i^1\ (i = 2, \cdots, N+1)$ を計算し, $u_i^1\ (i = 2, \cdots, N+1)$ と境界条件 u_1^1 を使って $u_i^2\ (i = 2, \cdots, N+1)$ を計算できます.

差分近似の取り方によっては計算ができないので注意が必要です.

差分近似した方程式は

$$u(x,t+\Delta t)=(1-c\frac{\Delta t}{\Delta x})u(x,t)+c\frac{\Delta t}{\Delta x}u(x-\Delta x,t)$$

となります. 時刻 $t+\Delta t$ の値を時刻 t の値だけで計算することができます(←陽解法と言う).

コードを書いてみると以下のような感じになります.

```
% プログラムが開始したら画面にメッセージを出す
disp('線形移流方程式(陽解法)が始まった!!');

clear;

% 配列の準備. (3 つの異なる dt で計算する)
x=zeros(11,1);
u_old=zeros(11,3); %古い時間ステップでの値
u_new=zeros(11,3); %新しい時間ステップでの値

% パラメーター設定
c=1;
N=10;       % 空間分割数
step_tot=5; % 時間ステップ総数
dx=1/N;

% x 座標
for i=1:11
    x(i,1)=(i-1)*dx;
end

% 初期値
u_old(1,:)=1; % 境界条件
u_old(2:11,:)=0;

for j=1:3
    dt=dx/c/2*j; % dt=0.5*dx/c,dx/c,1.5*dx/c の 3 パターン
    for t=1:step_tot
        u_new(1,j)=1; % 境界条件
```

```
        for i=2:11
            % 差分方程式から u_new を計算
            u_new(i,j)=(1-c*dt/dx)*u_old(i,j)+c*dt/dx*u_old(i-1,j);
        end
        if t==1
            clf('reset');
            fig=figure; % Figure を生成
            hold off;
            plot(x(:,1),u_old(:,j),'-','LineWidth',1.5); % 初期値をプロット
            hold on;
            plot(x(:,1),u_new(:,j),'-','LineWidth',1.5); % 直近の結果をプロット
        else
            plot(x(:,1),u_new(:,j),'-','LineWidth',1.5); % 直近の結果をプロット
        end
        u_old(:,j)=u_new(:,j);  % 次の時間ステップのために u_old を更新
    end
    legend('t=0','t=dt','t=2dt','t=3dt','t=4dt','t=5dt','Location','best');

    % Figure を JPEG で保存
    % 保存したいフォルダー
    dname='保存したいフォルダーのパス(C:¥Users¥tanaka¥Documents¥とか)';
    % 絶対パス込みのファイル名
    fname=append(dname,'好きな名前',sprintf('%i',j),'.jpg');
    % JPEG で保存
    print(fig,'-djpeg',fname,'-r600');
end

% プログラムが終了したら画面にメッセージと出す
disp('終わった!!');
```
