風上有限要素法を用いた
1 次元移流拡散方程式の数値解析

後藤 聡太

2021 年 3 月 15 日

1　移流拡散方程式とは

何らかの物理量 $u(\boldsymbol{x}, t)$ の一般的な流れを表す 2 階線形偏微分方程式であり，次の方程式で表される．

$$\frac{\partial u(\boldsymbol{x}, t)}{\partial t} + \nabla \cdot (\boldsymbol{c}u) = \nabla \cdot (D \nabla u)$$

ここで，$\boldsymbol{c}$ は物理量の速度，D は拡散係数と言われる物理量である．今回は簡単のため物理量の速度及び拡散係数は定数，かつ 1 次元問題として扱うため，

$$\frac{\partial u(x, t)}{\partial t} + c\, \frac{\partial u(x, t)}{\partial x} = D\, \frac{\partial^2 u(x, t)}{\partial^2 x} \tag{1}$$

を求解する．

2　移流拡散方程式の離散化

当然ながら，方程式 (1) を数値的に解くには時空間双方の離散化の手続きを踏む必要がある．ここでは空間方向の離散化を**最適風上 Galerkin 法（有限要素法の一種）**で，時間方向の離散化を差分法の一種である**クランクニコルソン**を適用し離散化を行った．

2.1　空間方向の離散化：有限要素法

まず，空間方向の離散化を行う．移流拡散方程式を通常の有限要素法で用いられる Galerkin 法によって離散化を行うと，格子ペクレ数が大きい場合，次の図 1 のように数値振動が起こってしまう．ここでペクレ数 P_c とは

$$P_c = \frac{c \Delta x}{D} \tag{2}$$

で定義される定数であり，この値が大きいと移流項の影響が強いことを表す．青線が数値解析解，赤線が解析解を表しており，数値解析解が数値振動を起こしていることが確認できる．

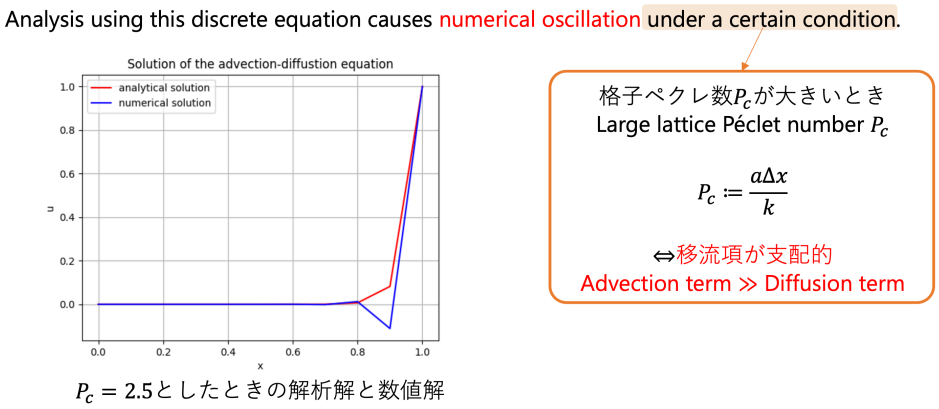

図 1　通常の Galerkin 法を用いたときの移流拡散方程式の数値解析結果

そこで，ペクレ数が大きくなり数値振動が起こってしまうことを防ぐため，人工拡散係数を導入し通常の Galerkin 法とは異なる重み関数を採用する．これは風上差分法から得られる帰結であるが，ここでは話が逸れるため省略する．具体的には次の重み関数を用いる．

$$\tilde{u}^* = u^* + \frac{D_{opt}}{c}\frac{du^*}{dx} \tag{3}$$

$$D_{opt} = \frac{c\Delta}{2}\left(\coth\frac{P_c}{2} - \frac{2}{P_c}\right) \tag{4}$$

ここで u^* は通常の Galerkin 法の重み関数である．さて，この風上差分法から得られる修正した重み関数を乗じ，解析領域 Ω 全体で積分し重みつき残差法により弱形式化すると次のような行列方程式となる．

$$\int_\Omega \left(\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} - D\frac{\partial^2 u}{\partial^2 x}\right)\tilde{u}^* dx = 0 \tag{5}$$

$$\Leftrightarrow (\boldsymbol{M} + \boldsymbol{M}^c)\dot{\boldsymbol{u}} + (\boldsymbol{A} + \boldsymbol{A}^c + \boldsymbol{K})\boldsymbol{u} = \boldsymbol{0} \tag{6}$$

ここで式 (6) の行列の要素は次で定義される．

$$M_{ij} = \int_\Omega \phi_i(x)\phi_j(x)dx \tag{7}$$

$$M^c_{ij} = \int_\Omega \frac{d\phi_i(x)}{dx}\phi_j(x)dx \tag{8}$$

$$A_{ij} = c\int_\Omega \frac{d\phi_i(x)}{dx}\phi_j(x)dx \tag{9}$$

$$A^c_{ij} = D_{opt}\int_\Omega \frac{d\phi_i(x)}{dx}\frac{d\phi_j(x)}{dx}dx \tag{10}$$

$$K_{ij} = D\int_\Omega \frac{d\phi_i(x)}{dx}\frac{d\phi_j(x)}{dx}dx \tag{11}$$

$$\boldsymbol{u} = \{u_1, u_2, \ldots\}^\top \tag{12}$$

$\phi_i(x)$ は節点 i における形状関数を表す．今回は形状関数として線形関数を用いた．より精度を上げたい場合は 2 次，3 次関数を適用するべきである．

2.2　時間方向の離散化：クランクニコルソン法

さて，空間の離散化式が得られたため，次に時間方向の離散化を行う．今回は陰的な 2 次精度の差分法であるクランクニコルソン法を適用し離散化する．式 (6) が時間ステップ $n+\theta(0 \leq \theta \leq 1)$ において成立するための条件を考える．

$$(\boldsymbol{M}+\boldsymbol{M}^c)\dot{\boldsymbol{u}}^{n+\theta}+(\boldsymbol{A}+\boldsymbol{A}^c+\boldsymbol{K})\boldsymbol{u}^{n+\theta}=\boldsymbol{0} \tag{13}$$

ここで，$\dot{\boldsymbol{u}}^{n+\theta}$ と $\boldsymbol{u}^{n+\theta}$ は

$$\dot{\boldsymbol{u}}^{n+\theta}=\frac{\boldsymbol{u}^{n+1}-\boldsymbol{u}^n}{\Delta t} \tag{14}$$

$$\boldsymbol{u}^{n+\theta}=(1-\theta)\boldsymbol{u}^n+\theta\boldsymbol{u}^{n+1} \tag{15}$$

と表せる．ただし，Δt は時間方向の刻み幅である．クランクニコルソン法では $\theta=1/2$ とするため，これらを式 (13) に代入すれば，時間ステップ更新式は

$$(\boldsymbol{M}+\boldsymbol{M}^c)\frac{\boldsymbol{u}^{n+1}-\boldsymbol{u}^n}{\Delta t}+(\boldsymbol{A}+\boldsymbol{A}^c+\boldsymbol{K})\frac{\boldsymbol{u}^n+\boldsymbol{u}^{n+1}}{2}=\boldsymbol{0} \tag{16}$$

さらに変形して

$$\{2(\boldsymbol{M}+\boldsymbol{M}^c)+(\boldsymbol{A}+\boldsymbol{A}^c+\boldsymbol{K})\Delta t\}\boldsymbol{u}^{n+1}=\{2(\boldsymbol{M}+\boldsymbol{M}^c)-(\boldsymbol{A}+\boldsymbol{A}^c+\boldsymbol{K})\Delta t\}\boldsymbol{u}^n \tag{17}$$

この行列方程式 (17) を毎ステップで解くことで解が得られる．

※ちなみに，上式で $\theta=0$ とした場合が前進差分法，$\theta=1$ とした場合が後退差分法に当たる．前進差分法で集中質量行列を用いれば，毎ステップで式 (17) のような複雑な方程式を求解する必要がなく計算速度が比較的速い．一方で陽解法であるため時間刻み幅に安定化条件が課される．

3　プログラムコードの説明と動作方法

3.1　解析プログラムについて

上記の離散化した行列の計算や行列方程式の求解を C++ によって実装した．直接関係ある C++ ファイルはヘッダファイルを含め全部で 7 つのファイルがある．それぞれのファイル名とその役割を以下に示す．

- main.cpp：main 関数があるファイル
- global.h：拡散係数をはじめとする諸変数の定義
- input_file.cpp：メッシュ情報の取得
- geometry.cpp：幾何情報を構成
- Calculate_matrix.cpp：行列を計算
- set_bc.cpp：境界の節点を設定
- utilities.cpp：行列方程式を求解するソルバー

次に main 関数内の処理内容を示す．

ソースコード 1　main 関数内の処理

```
int main(void){
    clock_t start = clock(); // 時間計測のため

    printf("節点数␣:␣%d\n 要素数␣:␣%d\n", n_p, n_e);

    Assign_array(); // 配列の定義

    input_file_reader(); // メッシュ情報の取得

    InitialAdjacency(); // 幾何情報の構成

    Calculate_matrix(); // 行列の計算

    set_bc(); // setting boundary condition

    // Jacobi_static(); // solve the linear equation by Jacobi scheme

    Dynamics(); // dynamical analysis

    // Calculation time
    clock_t end = clock();
    const double time = static_cast <double> (end-start)/CLOCKS_PER_SEC*1000;
    printf("Calculation␣time␣:␣%.3f␣[ms]\n",time);
    return 0;
}
```

次の流れで処理が行われている．

1. 配列の定義
2. メッシュ情報の取得
3. 幾何情報の構成
4. 行列の計算
5. 境界条件の付与
6. 動的解析

行列の計算は 2 節で触れた計算式をもとに行なっている．さて，動的解析において毎ステップで行列方程式 (17) を解く必要があるが，今回はこのソルバーとして双共役勾配法（BiCG 法）を用いる．共役勾配法（CG 法）ではない理由は，逆行列を求めるべき行列が非対称行列であるからであるためである．（具体的には式 (8), (9) で表される行列が非対称行列であるためである．）

3.2　解析プログラムの動作方法

基本的には main.cpp 関数をコンパイルし実行ファイルを生成すればよい．拡散係数や移流速度などの物理定数を変更したい場合は global.h ファイルの各定数を変更すれば良い．実行ファイルを実行し，解析が終了すると各タイムステップごとの $\boldsymbol{u}$ の値が格納されたファイル”u_dynamics_ad=??_Di=??.dat”が animation フォルダ下に生成される．??には設定していた移流速度および拡散係数の値が表示されている．これを次に説明する可視化プログラムが参照し，アニメーションを作成する．

3.3　可視化プログラムの動作方法

解析ファイルを実行し，生成させた解析結果ファイルが animation フォルダ内にあることを確認する．同じく animation フォルダ内にある Python ファイル，animation.py が可視化のためのプログラムである．可視化プログラム内の 12 行目にある解析結果ファイルが先ほど生成した解析ファイル名と一致していることを確認したら，この Python ファイルを実行させれば以下の図のようなアニメーションウィンドウが表示される．アニメーションを保存したい場合は 26 行目のコメントアウトを外せば良い．

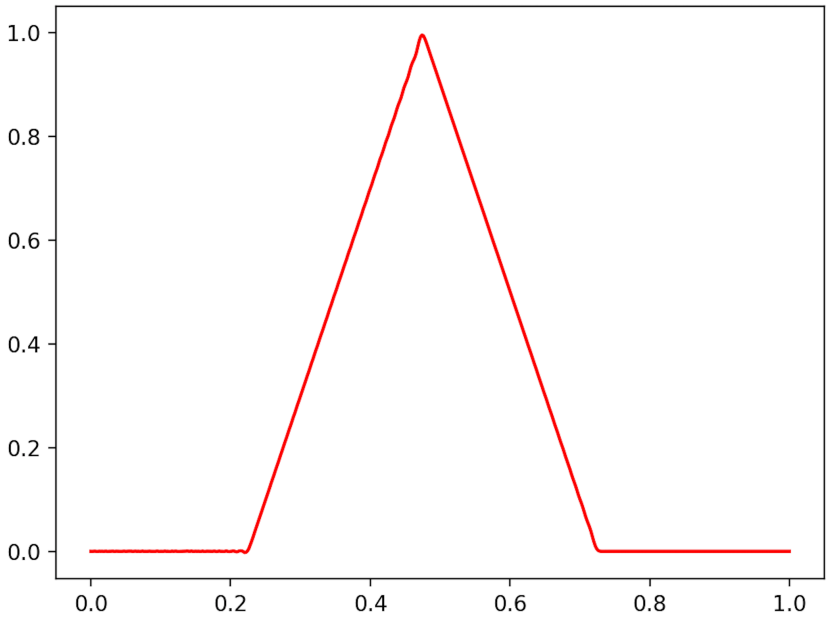

図 2　可視化プログラムを実行させたときに表示されるアニメーションウィンドウ