“私にとっての”
反応拡散系数値シミュレーション入門
基礎編[1]

2002 年度横浜市立大学集中講義ノート + 2005 年度金沢大学特別セミナー

+ 2008 年度京都大学 GCOE セミナー講義ノート + CD-ROM

長山 雅晴

京都大学数理解析研究所 助手 (1999 年 2 月〜2004 年 3 月)

金沢大学大学院自然科学研究科　助教授（2004 年 4 月〜2007 年 3 月）, 准教授（2007 年 4 月〜2008 年 3 月）

金沢大学理工研究域数物科学系　准教授（2008 年 4 月〜）

E-mail : nagayama@kenroku.kanazawa-u.ac.jp

[1]2008 年 12 月 31 日：人に見せても恥ずかしくない程度にはなった.

目 次

第1章　はじめに

1.1　数値シミュレーションとは

与えられたモデル方程式 (ここでは常微分方程式系や偏微分方程式系) の解挙動を調べることは，モデル方程式の妥当性や新しい現象の発見をするために重要である．そのためには解の遷移過程やダイナミックな変化を捉えることが必要となり，**数値シミュレーション** という手段を必要とする．数値シミュレーションはただ数値計算をすることではなく次の三段階から成り立っている [8].

(1) 現象のモデル化：偏微分方程式 (常微分方程式) への定式化
「1 つ 1 つの項の持つ意味をしっかり理解することが重要」

(2) 与えられたモデル方程式を解く：離散化と数値解法

(3) 得られた結果の評価：モデルの妥当性や解法の妥当性を含む[a]

[a]数値計算結果の成否は何で判断すべきか？この問題に対する答えの 1 つは「解が求まる方程式を対象として，数値解と理論解を定性的定量的に比較する」

(2) と (3) を繰り返し行うことで,「直感的に微分方程式の解挙動が理解できるようになる（モデル方程式に対する理解が深まる）[1]」
実際の数値計算ではパラメータを動かして解のダイナミクスを調べる．そのときモデルの妥当性や新しい現象の発見に対する理解が重要となる．このような数値シミュレーションを行う上で何が重要なのか？その答えは難しいがあえて答えるとするならば,

(1) 非線形現象を多く知ること，数値解に対して物理的理解（解釈）が出来ること．

(2) 数値スキームの特徴を知ること（理論＋実践）

(3) 数学的な理解（例えば，分岐理論的理解）

ではないだろうか[2].
このような数値シミュレーションの基本的な訓練方法はまだ確立されていないと思うが，あえて言うなら次のように考えたらいいのではないか[3]：

[1]この講義では (2) の方法を解説する．
[2]私自身はモデル方程式の数値計算を繰り返し行うことで，その方程式の特徴を理解しようとする，そして説明は出来ないけども，パラメータをどう動かしたら目的の解を得ることができるのかがわかってくる．しかしこれでは数値解の現象が説明できないけど…
[3]長山提案ですが，数値計算をしているときに師匠にいつも言われていたことと同じです.

(1) 与えられたモデル方程式に対して出現するであろう数値解の直観的な説明を考える.（当然ではあるが，理論的にわかることは最初にチェックすべきである.）

(2) 数値計算によって得られた結果と比較する.

(3) 直観と異なっていた場合，最初に数値解法を確認する．数値解法が正しい場合は，数値計算結果をみて，直感的説明をつける[a].

(4) その説明を他人に聞いてもらう（これが結構重要）[b].

[a]分岐理論から理解することも重要
[b]このことによって自分の考察の問題点が明らかになる場合が多い.

1.2　数値シミュレーションプログラムのあらすじ

START
↓
使用するファイルの OPEN
↓
パラメータ設定 ····· ファイルから読み込む方がよい.
↓
計算の前処理 (必要があれば)
↓
初期データ設定 ····· ファイルから読み込む方がよい.
↓
グラフ (ファイル) 出力またはデータ出力
↓
*1 タイムステップ計算
↓
グラフ (ファイル) 出力またはデータ出力 (何回かおきに)
↓ *に戻る
終末処理 ····· ファイルの close，グラフィクスの終了，最終データの出力等
↓
END

注意：後で自分の作ったプログラムが何をしているかどうか分かる程度には，面倒でもプログラムにドキュメント書いておくこと．後から何をしているのかわからなくなります.

1.3　入門編のターゲット

入門編では次のような反応拡散系を数値計算するための基本的な数値計算法とその可視化方法について解説する:

$$\frac{\partial \boldsymbol{u}}{\partial t} = D\Delta \boldsymbol{u} + \boldsymbol{f}(\boldsymbol{u}). \tag{1.1}$$

ただし，D は非負対角行列とする．(1.1) に含まれるモデル方程式としては次のような方程式が挙げられる：

神経繊維上の電位の伝播モデル (FitzHugh-Nagumo 方程式)[25]

$$\begin{cases} \dfrac{du}{dt} = du_{xx} + \dfrac{1}{\varepsilon}\left(u(1-u)(u-a) - v\right), \\ \dfrac{dv}{dt} = u - \gamma v. \end{cases} \tag{1.2}$$

ただし，$0 < a < 1/2$ であり，u は電位を表し，v はイオンチャネルの開き具合を表している．

BZ 反応モデル [23, 24, 25]

$$\begin{cases} \dfrac{du}{dt} = d_u \Delta u + \dfrac{1}{\varepsilon}\left(u(1-u) - fw\dfrac{u-q}{u+q}\right), \\ \dfrac{dw}{dt} = d_w \Delta w + u - \gamma w. \end{cases} \tag{1.3}$$

Gray-Scott モデル [24] $\boxed{V \overset{\text{補給率 } h}{\longrightarrow} 2U + V \to 3U \overset{\text{放出率 } a}{\longrightarrow} U}$

$$\begin{cases} \dfrac{du}{dt} = d_u \Delta u - au + vu^2, \\ \dfrac{dv}{dt} = d_v \Delta v + h(1-v) - vu^2. \end{cases} \tag{1.4}$$

発熱反応モデル $\boxed{A \overset{\text{補給率 } h}{\longrightarrow} \mathrm{A} + \mathrm{O} \overset{k(T)}{\longrightarrow} \mathrm{P} + \text{熱} \overset{\text{放熱率 } k}{\longrightarrow} \text{熱}}$

$$\begin{cases} \dfrac{dT}{dt} = d_T \Delta T + \dfrac{1}{\varepsilon}\left(-ku + a\exp(\dfrac{T}{1+T/c})\right), \\ \dfrac{da}{dt} = d_a \Delta a + h(a_0 - a) - a\exp(\dfrac{T}{1+T/c}). \end{cases} \tag{1.5}$$

ここで，u は温度，v は反応物質を表している．

これらの方程式に対して，反応項に対する数値計算

$$\frac{d\boldsymbol{u}}{dt} = \boldsymbol{f}(\boldsymbol{u}) \tag{1.6}$$

と拡散項に対する数値計算

$$\frac{du}{dt} = D\Delta u \tag{1.7}$$

の二つの部分に分けて解説する．

反応拡散方程式の数値シミュレーションは，反応項の性質を調べることから始まる．そのためにも常微分方程式系の数値計算法は必要である．拡散方程式の数値計算ができれば，反応拡散系の数値計算は一応可能となる．

第2章　常微分方程式の数値計算法

反応項に対する数値計算法を解説する．ここでは次のような非自励系を含む一般の常微分方程式系に対する初期値問題の数値解法を解説する．

$$\left\{ \begin{array}{lcl} \dfrac{d\boldsymbol{u}}{dt} & = & \boldsymbol{f}(t,\boldsymbol{u}), \\ \boldsymbol{u}(t_0) & = & \boldsymbol{u}_0. \end{array} \right. \tag{2.1}$$

ただし，$\boldsymbol{u} : \boldsymbol{R} \to \boldsymbol{R}^N, \boldsymbol{f} : \boldsymbol{R}^N \times \boldsymbol{R}_+ \to \boldsymbol{R}^N$ である．時間差分を Δt とし，$t_n = t_0 + n \times \Delta t$,

$$\boldsymbol{u}(t_n) = \boldsymbol{u}_n, \tag{2.2}$$

と書く．

2.1　Euler 法 [9, 10, 11]

Euler 法は最も単純な計算方法であり，テイラー展開を用いて公式を導く．

Euler 法

$$\begin{array}{rcll} \boldsymbol{u}_0 & = & \boldsymbol{u}(t_0), & (2.3) \\ \boldsymbol{u}_{n+1} & = & \boldsymbol{u}_n + \Delta t \boldsymbol{f}(t_n, \boldsymbol{u}_n). & (2.4) \end{array}$$

このスキームは $O(\Delta t)$ の誤差を持つ.
簡単な証明
$\boldsymbol{u}(t + \Delta t)$ を $\Delta t = 0$ のまわりでテーラー展開すると

$$\boldsymbol{u}(t + \Delta t) = \boldsymbol{u}(t) + \frac{d}{dt}\boldsymbol{u}(t)\Delta t + \frac{1}{2}\frac{d^2}{dt^2}\boldsymbol{u}(t)(\Delta t)^2 + O(\Delta t^3).$$

故に

$$\frac{d}{dt}\boldsymbol{u}(t) = \frac{\boldsymbol{u}(t + \Delta t) - \boldsymbol{u}(t)}{dt} - \frac{1}{2}\frac{d^2}{dt^2}\boldsymbol{u}(t)\Delta t + O(\Delta t^2). \tag{2.5}$$

2.2　Runge-Kutta 法 [10]

テイラー展開法とは異なる．$\boldsymbol{u}$ の導関数を使わない方法．次のように導出する：

$$\boldsymbol{u}(t + \Delta t) = \boldsymbol{u}(t) + \Delta t \Phi(t, \boldsymbol{u}(t)) \tag{2.6}$$

という形式を仮定する．このとき

$$\Phi(t, \boldsymbol{u}(t)) = \alpha k_1 + \beta k_2, \tag{2.7}$$

$$k_1 = \boldsymbol{f}(t, \boldsymbol{u}(t)), \tag{2.8}$$

$$k_2 = \boldsymbol{f}(t + p\Delta t, \boldsymbol{u}(t) + q\Delta t k_1) \tag{2.9}$$

である．ここで (2.9) の右辺を $\Delta t = 0$ の周りでテーラー展開すると

$$\begin{aligned} k_2 &= \boldsymbol{f}(t,\boldsymbol{u}(t)) + \frac{\partial}{\partial t}\boldsymbol{f}(t,\boldsymbol{u}(t))p\Delta t \\ &\quad + D_u\boldsymbol{f}(t,\boldsymbol{u}(t))k_1 q\Delta t + O(\Delta t^2). \end{aligned}$$

(2.8) より

$$\begin{aligned} k_2 &= \boldsymbol{f}(t,\boldsymbol{u}(t)) + \frac{\partial}{\partial t}\boldsymbol{f}(t,\boldsymbol{u}(t))p\Delta t \\ &\quad + D_u\boldsymbol{f}(t,\boldsymbol{u}(t))\boldsymbol{f}(t,\boldsymbol{u}(t))q\Delta t + O(\Delta t^2). \end{aligned} \tag{2.10}$$

(2.10) を (2.7) に代入して (2.6) を計算すると

$$\begin{aligned} \boldsymbol{u}(t+\Delta t) &= \boldsymbol{u}(t) + \Delta t(\alpha\boldsymbol{f}(t,\boldsymbol{u}(t)) + \beta(\boldsymbol{f}(t,\boldsymbol{u}(t)) \\ &\quad + p\Delta t\frac{\partial}{\partial t}\boldsymbol{f}(t,\boldsymbol{u}(t)) + D_u\boldsymbol{f}(t,\boldsymbol{u}(t))\boldsymbol{f}(t,\boldsymbol{u}(t))q\Delta t)) + O(\Delta t^3). \\ &= \boldsymbol{u}(t) + \Delta t(\alpha+\beta)\boldsymbol{f}(t,\boldsymbol{u}(t)) \\ &\quad + \Delta t^2\beta(p\frac{\partial}{\partial t}\boldsymbol{f}(t,\boldsymbol{u}(t)) + qD_u\boldsymbol{f}(t,\boldsymbol{u}(t))\boldsymbol{f}(t,\boldsymbol{u}(t))) + O(\Delta t^3). \end{aligned} \tag{2.11}$$

今，$\boldsymbol{u}(t+\Delta t)$ を $\Delta t = 0$ の周りでテーラー展開すると

$$\boldsymbol{u}(t+\Delta t) = \boldsymbol{u}(t) + \Delta t(\boldsymbol{f}(t,\boldsymbol{u}(t)) + \frac{\Delta t^2}{2}\frac{d^2}{dt^2}\boldsymbol{u} \tag{2.12}$$

となる．ところで

$$\begin{aligned} \frac{d}{dt}\boldsymbol{u} &= \boldsymbol{f}(t,\boldsymbol{u}), \\ \frac{d^2}{dt^2}\boldsymbol{u} &= \frac{d}{dt}\boldsymbol{f} + D_u\boldsymbol{f}\frac{d}{dt}\boldsymbol{u}, \\ &= \frac{d}{dt}\boldsymbol{f} + D_u\boldsymbol{f}\boldsymbol{f} \end{aligned} \tag{2.13}$$

となるので，(2.13) を (2.12) に代入すると

$$\boldsymbol{u}(t+\Delta t) = \boldsymbol{u}(t) + \Delta t(\boldsymbol{f}(t,\boldsymbol{u}(t)) + \frac{\Delta t^2}{2}\left(\frac{d}{dt}\boldsymbol{f} + D_u\boldsymbol{f}\boldsymbol{f}\right) \tag{2.14}$$

と書ける．(2.11), (2.14) の係数比較から

$$\alpha+\beta = 1, \quad \beta p = \frac{1}{2}, \quad \beta q = \frac{1}{2}$$

が得られる．ここで

$$\alpha = 0, \quad \beta = 1, \quad p = \frac{1}{2}, \quad q = \frac{1}{2}$$

とおくと，次の原型版修正 Euler 法が得られる．

修正 Euler 法（原型版）

$$\begin{aligned} \boldsymbol{u}_0 &= \boldsymbol{u}(t_0), && (2.15)\\ \boldsymbol{k}_1 &= \boldsymbol{f}(t_n,\boldsymbol{u}_n), && (2.16)\\ \boldsymbol{k}_2 &= \boldsymbol{f}(t_n+\frac{\Delta t}{2},\boldsymbol{u}_n+\frac{\Delta t}{2}\boldsymbol{k}_1), && (2.17)\\ \boldsymbol{u}_{n+1} &= \boldsymbol{u}_n + \Delta t\boldsymbol{k}_2. && (2.18) \end{aligned}$$

通常は上記の方法から変形した次の通常版修正 Euler が使われる．

修正 Euler 法（通常版）

$$\begin{aligned} \boldsymbol{u}_0 &= \boldsymbol{u}(t_0), && (2.19)\\ \boldsymbol{k} &= \boldsymbol{u}_n + \frac{\Delta t}{2}\boldsymbol{f}(t_n, \boldsymbol{u}_n), && (2.20)\\ \boldsymbol{u}_{n+1} &= \boldsymbol{u}_n + \Delta t \boldsymbol{f}(t_n + \frac{\Delta t}{2}, \boldsymbol{k}). && (2.21) \end{aligned}$$

このスキームは $O(\Delta t^2)$ の誤差を持つ．また，

$$\alpha = \beta = \frac{1}{2}, \quad p = q = 1 \tag{2.22}$$

とおくと，Heun 法と呼ばれる．

Heun 法

$$\begin{aligned} \boldsymbol{u}_0 &= \boldsymbol{u}(t_0), && (2.23)\\ \boldsymbol{k}_1 &= \boldsymbol{f}(\boldsymbol{u}_n, t_n), && (2.24)\\ \boldsymbol{k}_2 &= \boldsymbol{f}(t_n + \Delta t, \boldsymbol{u}_n + \Delta t \boldsymbol{k}_1), && (2.25)\\ \boldsymbol{u}_{n+1} &= \boldsymbol{u}_n + \frac{\Delta t}{2}(\boldsymbol{k}_1 + \boldsymbol{k}_2). && (2.26) \end{aligned}$$

次に示す Runge-Kutta 法は $O(\Delta t^4)$ の誤差を持つ：

4 次の Runge-Kutta 法

$$\begin{aligned} \boldsymbol{u}_0 &= \boldsymbol{u}(t_0), && (2.27)\\ \boldsymbol{k}_1 &= \boldsymbol{f}(t_n, \boldsymbol{u}_n), && (2.28)\\ \boldsymbol{k}_2 &= \boldsymbol{f}(t_n + \frac{1}{2}\Delta t, \boldsymbol{u}_n + \frac{1}{2}\Delta t \boldsymbol{k}_1), && (2.29)\\ \boldsymbol{k}_3 &= \boldsymbol{f}(t_n + \frac{1}{2}\Delta t, \boldsymbol{u}_n + \frac{1}{2}\Delta t \boldsymbol{k}_2), && (2.30)\\ \boldsymbol{k}_4 &= \boldsymbol{f}(t_n + \Delta t, \boldsymbol{u}_n + \Delta t \boldsymbol{k}_3), && (2.31)\\ \boldsymbol{u}_{n+1} &= \boldsymbol{u}_n + \frac{\Delta t}{6}\left(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4\right). && (2.32) \end{aligned}$$

この数値計算方法の導出は [11] に詳しく解説されている．

常微分方程式の数値計算には通常 4 次の Runge-Kutta 法を用いる．

2.3 問題

2.3.1 問題 1

例題 1（ロジスティック方程式)[25]

$$\frac{du}{dt} = au(1-u), \quad u(0) = u_0. \tag{2.33}$$

ただし，a は正定数．

(1.1.1) $u_0 > 0$ の初期値に対して Euler 法や Runge-Kutta 法で数値計算しなさい．得られた解を (t, u) 平面でグラフ表示しなさい[1]．

(1.1.2) Euler 法で数値計算したとき，Δt の大きさを変化させて (2.33) の解を求めなさい．

2.3.2　問題２

問題２ (興奮現象モデル)

$$\begin{cases} \dfrac{du}{dt} = \dfrac{1}{\varepsilon}(u(1-u)(u-a)-v) \equiv \dfrac{1}{\varepsilon}f(u,v), \\ \qquad\qquad\qquad\qquad\qquad\qquad t > 0, \\ \dfrac{dv}{dt} = u - \gamma v \equiv g(u,v), \\ (u,v)(0) = (u_0, v_0). \end{cases} \tag{2.34}$$

ただし，$a = 0.2, \varepsilon = 0.001$ とする．

(1.2.1) γ を適当に固定して $f(u,v) = 0, g(u,v) = 0$ を (u,v) 平面に表示しなさい．

(1.2.2) 平衡解 $f(\overline{u}, \overline{v}) = 0,\ \ g(\overline{u}, \overline{v}) = 0$ の線形化安定性を調べよ．

(1.2.3) 初期値を $(u_0, v_0) = (0.18, 0.0)$ と $(u_0, v_0) = (0.22, 0.0)$ と置いたときの解軌道を (u,v) 平面で表示しなさい．

2.3.3　問題３

問題３ (2 体問題の微分方程式，惑星や人工衛星の軌道を求める微分方程式)[7]

$$\begin{cases} \dfrac{d^2x}{dt^2} = -\dfrac{x}{(x^2+y^2)^{3/2}}, \\ \qquad\qquad\qquad\qquad\qquad\qquad t > 0, \\ \dfrac{d^2y}{dt^2} = -\dfrac{y}{(x^2+y^2)^{3/2}}, \\ \\ x(0) = 1-e,\ \ \dot{x}(0) = 0,\ \ y(0) = 0,\ \ \dot{y}(0) = \sqrt{(1+e)/(1-e)},\ \ e = 0.9. \end{cases} \tag{2.35}$$

e が 1 に近いと数値的に解き難いことが知られている．

(1.3.1) Euler 法，修正 Euler 法，Runge-Kutta 法で (2.35) を解き，(x,y) 平面に表示しなさい．

[1] 表示方法およびプリンターへの出力方法は第 5.4 節参照のこと．

第3章　拡散方程式の数値計算法（差分法）

はっきり言って「拡散方程式の数値計算は行列計算である」．いかに速く精度よく行列計算するのかが問題である．

3.1　熱方程式（拡散方程式）の導出 [3]

長さ L の細長い針金を考える．断面積 Ω が十分小さいときは 1 次元として近似できる．区間 $[0, L]$ 上の任意の場所 x, 時刻 t における温度を $u(t, x)$ とする．針金の密度 $\rho(x)$，比熱 $c(x)$，熱伝導係数 $k(x)$ とする．任意の場所 x, 時刻 t における熱量を $q(t, x)$ とする．熱方程式は次のフーリエ則に基づいて導出される：

フーリエ則 (Fourier law)　（実験）
単位時間当りの熱量の流出はその場所 (x) での温度勾配に比例

$$\frac{dq}{dt}(t,x) \propto \frac{\partial u}{\partial x}(t,x).$$

すなわち

$$\frac{\partial q}{\partial t}(t,x) = -k(x)\frac{\partial u}{\partial x}(t,x).^{a}$$

[a] 説明重要

$(0, L)$ の内の任意の区間 (a, b) での総熱量は

$$Q(t) = \int_a^b c(x)\rho(x)u(t,x)dx$$

となる．このとき単位時間当りの総熱量の変化は

$$\frac{dQ}{dt}(t) = \int_a^b c(x)\rho(x)\frac{\partial u}{\partial t}(t,x)dx \tag{3.1}$$

と書ける．ところで熱量の変化は,「両端 a, b を通してどれだけ熱量が流入流出したか」で決まるので

$$\begin{aligned}\frac{dQ}{dt}(t) &= K(b)\frac{\partial u}{\partial x}(t,b) - K(a)\frac{\partial u}{\partial x}(t,a)^{a} \\ &= \int_a^b \frac{\partial}{\partial x}\left(K(x)\frac{\partial u}{\partial x}(t,x)\right)dx. \end{aligned} \tag{3.2}$$

[a] この符号の意味をしっかり説明する

となる．(3.1) と (3.2) から

$$\int_a^b c(x)\rho(x)\frac{\partial u}{\partial t}(t,x)dx = \int_a^b \frac{\partial}{\partial x}\left(K(x)\frac{\partial u}{\partial x}(t,x)\right)dx$$

故に

$$c(x)\rho(x)\frac{\partial u}{\partial t}(t,x) = \frac{\partial}{\partial x}\left(K(x)\frac{\partial u}{\partial x}(t,x)\right).$$

c, ρ, K が定数ならば

$$\frac{\partial u}{\partial t} = \frac{K}{c\rho}\frac{\partial^2 u}{\partial x^2}$$

となる．

3.2　拡散方程式の初期境界値問題の解の求め方

数値スキームの有効性を確認するための一つの方法として数値解と理論解を比較する方法がある．この節では変数分離法を用いて拡散方程式の理論解を求める．

現象の数学 I を参照

3.3　1 次元初期境界値問題 [3, 4]

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2}, \quad 0 < x < L,\ t > 0 \tag{3.3}$$

ただし，D は拡散係数と呼ばれる．

初期条件[1]：

$$u(x, 0) = u_0(x). \tag{3.4}$$

Dirichlet 境界条件

$$u(0, t) = \alpha, \quad u(L, t) = \beta. \tag{3.5}$$

Neumann 境界条件

$$\frac{\partial}{\partial x}u(t, 0) = \alpha, \quad \frac{\partial}{\partial x}u(t, L) = \beta. \tag{3.6}$$

周期境界条件

$$u(t, 0) = u(t, L), \quad \frac{\partial u}{\partial x}(t, 0) = \frac{\partial u}{\partial x}(t, L). \tag{3.7}$$

方程式の離散化

$$\frac{\partial u}{\partial t} \quad \to \quad \frac{u^{n+1} - u^n}{\Delta t}. \tag{3.8}$$

$$\frac{\partial^2 u}{\partial x^2} \quad \to \quad \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}. \tag{3.9}$$

(2.4) から時間離散化誤差は $O(\Delta t)$ である．空間離散化誤差は $O(\Delta x^2)$ である．

証明

$u(x + \Delta x)$ および $u(x - \Delta x)$ を $\Delta x = 0$ のまわりでテーラー展開する：

$$u(x + \Delta x) = u(x) + u_x(x)\Delta x + \frac{1}{2}u_{xx}(x)\Delta x^2 + \frac{1}{6}u_{xxx}(x)\Delta x^3 + \frac{1}{24}u_{xxxx}(x)\Delta x^4 + O(\Delta x^5). \tag{3.10}$$

$$u(x - \Delta x) = u(x) - u_x(x)\Delta x + \frac{1}{2}u_{xx}(x)\Delta x^2 - \frac{1}{6}u_{xxx}(x)\Delta x^3 + \frac{1}{24}u_{xxxx}(x)\Delta x^4 + O(\Delta x^5). \tag{3.11}$$

(3.10)+(3.11) より

$$u(x + \Delta x) + u(x - \Delta x) = 2u(x) + u_{xx}(x)\Delta x^2 + \frac{1}{12}u_{xxxx}(x)\Delta x^4 + O(\Delta x^5). \tag{3.12}$$

$$u_{xx}(x) = \frac{u(x + \Delta x) - 2u(x) + u(x - \Delta x)}{\Delta x^2} - \frac{1}{12}u_{xxxx}(x)\Delta x^2 + O(\Delta x^3). \tag{3.13}$$

[1] $u_0(x)$ は連続関数であれば特に問題ない

従って，離散化誤差は $O(\Delta x^2)$ となる．

境界条件の離散化

$$\text{Dirichlet 境界条件} \qquad u_0^n = \alpha, \quad u_N^n = \beta. \tag{3.14}$$

$$\text{Neumann 境界条件}^a \qquad u_1^n - u_{-1}^n = 2\Delta x\alpha, \quad u_{n+1}^n - u_{n-1}^n = 2\Delta x\beta. \tag{3.15}$$

$$\text{周期境界条件} \qquad u_0^n = u_n^n, \quad u_{-1}^n = u_{n-1}^n. \tag{3.16}$$

[a] 中心差分することによって，境界での差分誤差を $O(\Delta x^2)$ とすることができる．

3.3.1 陽解法

陽解法公式

$$u_i^{n+1} = Dru_{i-1}^n + (1-2Dr)u_i^n + Dru_{i+1}^n, \quad 0 \le i \le N. \tag{3.17}$$

ただし，$r = \Delta t/\Delta x^2$ である．
数値スキームの安定性 (A.1) より

$$\lambda_k = 1 - 4Dr\sin^2(\frac{k\Delta x\pi}{2}) \tag{3.18}$$

から

$$|\lambda_k| \le 1 \leftrightarrow D\frac{\Delta t}{\Delta x^2} \le \frac{1}{2}. \tag{3.19}$$

従って，(3.19) を満たさない場合は数値計算が不安定化する[2]．

安定性の直感的理解

図 3.1 のように u_i^{n+1} の値は $u_{i-1}^n, u_i^n, u_{i+1}^n$ の 3 点が重み $Dr, 1-2Dr, Dr$ で平均化されていることを意味しており，重みが正でなければ本当の平均ではなくなる．従って，$1-2Dr > 0$ が必要となる．

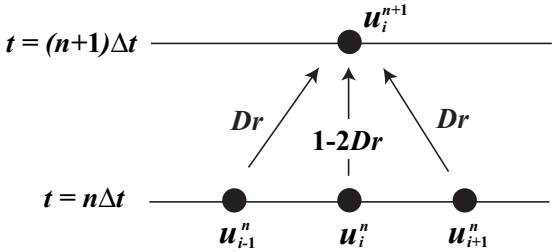

図 3.1: 陽解法の安定性の直感的理解

陽解法の誤差解析

$$\frac{\partial u}{\partial t} = \frac{u(x,t+\Delta t)-u(x,t)}{\Delta t} - \frac{\Delta t}{2}\frac{\partial^2 u}{\partial t^2} - \frac{\Delta t^2}{6}\frac{\partial^3 u}{\partial t^3} + O(\Delta t^3), \tag{3.20}$$

$$\frac{\partial^2 u}{\partial x^2} = \frac{u(x+\Delta x,t)-2u(x,t)+u(x-\Delta x,t)}{\Delta x^2} - \frac{\Delta x^2}{12}\frac{\partial^4 u}{\partial x^4} - O(\Delta x^4) \tag{3.21}$$

を (3.3) に代入すると

$$\frac{u_i^{n+1}-u_i^n}{\Delta t} = D\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} - D\frac{\Delta x^2}{12}\frac{\partial^4 u}{\partial x^4} + \frac{\Delta t}{2}\frac{\partial^2 u}{\partial t^2} + O(\Delta x^4, \Delta t^2), \tag{3.22}$$

[2] 注意：理論的には $D\frac{\Delta t}{\Delta x^2} \le \frac{1}{2}$ で数値計算法は安定であるが，数値計算による誤差によって (3.19) を満たしていても不安定化する場合があるので，$D\frac{\Delta t}{\Delta x^2} \le \frac{1}{6}$ 程度にしていた方が安全である．計算が途中で発散した場合は必ず $D\frac{\Delta t}{\Delta x^2}$ を小さくしてみよう．

ここで，

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial t} D \frac{\partial^2 u}{\partial x^2} = D^2 \frac{\partial^4 u}{\partial x^4} \tag{3.23}$$

より

$$\begin{aligned} \frac{u_i^{n+1} - u_i^n}{\Delta t} &= D \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} + \left(\frac{\Delta t}{2} D^2 - D \frac{\Delta x^2}{12} \right) \frac{\partial^4 u}{\partial x^4} + O(\Delta x^4, \Delta t^2), \\ &= D \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} + \frac{1}{2} D \Delta x^2 \left(D \frac{\Delta t}{\Delta x^2} - \frac{1}{6} \right) \frac{\partial^4 u}{\partial x^4} + O(\Delta x^4, \Delta t^2). \end{aligned} \tag{3.24}$$

故に

$$D \frac{\Delta t}{\Delta x^2} = \frac{1}{6} \tag{3.25}$$

のとき誤差は $O(\Delta x^4, \Delta t^2)$ となり，それ以外のときの誤差は $O(\Delta x^2, \Delta t)$ となる．しかしながら，境界条件の誤差から空間離散化の誤差は $O(\Delta x^2)$ となる．

3.3.2　陰解法

陰解法公式

$$u_i^{n+1} - u_i^n = D\theta(r u_{i-1}^{n+1} - 2r u_i^{n+1} + r u_{i+1}^{n+1}) + D(1-\theta)(r u_{i-1}^n - 2r u_i^n + r u_{i+1}^n), \quad 0 \le i \le N. \tag{3.26}$$

ここで，$\theta = 1/2$ のとき Crank-Nicolson 公式とよばれる．$\theta = 1$ のとき完全陰解法とよばれる．
陰解法数値スキームの安定性を調べる．

$$\lambda_n = \frac{1 - 4Dr(1-\theta)\sin^2(\frac{n\Delta x \pi}{2})}{1 + 4Dr\theta \sin^2(\frac{n\Delta x \pi}{2})}, \quad 0 \le n \le N. \tag{3.27}$$

から，$0 < \theta < 1/2$ ならば条件安定，$1/2 \le \theta \le 1$ ならば無条件安定であることが分かる陰解法は無条件安定なスキームであるが次のような連立一次方程式を解かなければならない．

$$A\boldsymbol{x} = \boldsymbol{b}. \tag{3.28}$$

ただし，A は三重対角行列で，その成分は $a_{i,i} = 1 + 2D\theta r$，$a_{i,i-1} = a_{i,i+1} = -D\theta r$ である．

3.3.2.1　LU 分解法

三重対角行列の連立一次方程式の計算は LU 分解法を用いて行う．

$$A = LU$$

ただし，境界条件が (3.14) あるいは (3.15) の場合

$$A = \begin{pmatrix} a_1 & c_1 & & & & \\ b_2 & a_2 & c_2 & & \mathbf{0} & \\ & \cdots & \cdots & \cdots & & \\ & & \cdots & \cdots & \cdots & \\ & \mathbf{0} & & b_{N-1} & a_{N-1} & c_{N-1} \\ & & & & b_N & a_N \end{pmatrix} \tag{3.29}$$

となる.

$$L=\begin{pmatrix} L_{11} & & & & 0 \\ L_{22} & L_{12} & & & \\ & \cdots & \cdots & & \\ & & \cdots & \cdots & \\ & & & L_{2N-1} & L_{1N-1} \\ & 0 & & & L_{2N} \quad L_{1N} \end{pmatrix},\quad U=\begin{pmatrix} 1 & U_1 & & & 0 \\ & 1 & U_2 & & \\ & & \cdots & \cdots & \\ & & & \cdots & \cdots \\ & & & & 1 \quad U_{N-1} \\ & 0 & & & 1 \end{pmatrix}. \tag{3.30}$$

と A を LU 分解し

$$L\boldsymbol{y}=\boldsymbol{b},$$

$$U\boldsymbol{x}=\boldsymbol{y}$$

を順に解くことで計算できる.

LU 分解のアルゴリズム

```
L[1][1] = a[1];
for (i = 2; i <= N ; i++) {
  L[2][i] = b[i];
}
for (i = 1; i <= N-1; i++)
{
  U[i] = c[i] /L[1][i];
  L[1][i+1] = a[i+1] - U[i]*L[2][i+1];
}
```

$L\boldsymbol{y}=\boldsymbol{b}$, $U\boldsymbol{x}=\boldsymbol{y}$ の解法アルゴリズム

```
y[1] = b[1]/L[1][1];
for (i = 2; i <= N; i++)
{
  y[i] = ( b[i] - L[2][i]*y[i-1])/L[1][i];
}
x[N] = y[N];
for (i = N-1; i >= 1; i - -)
{
  x[i] = y[i] - U[i]*x[i+1];
}
```

境界条件が (3.16) の場合，行列 A は次のようになる

$$A=\begin{pmatrix} a_1 & c_1 & & & & b_1 \\ b_2 & a_2 & c_2 & & 0 & \\ & \cdots & \cdots & \cdots & & \\ & & \cdots & \cdots & \cdots & \\ & 0 & & b_{N-1} & a_{N-1} & c_{N-1} \\ c_N & & & & b_N & a_N \end{pmatrix}. \tag{3.31}$$

このとき A を LU 分解すると行列 L, U は次のようになる.

$$L = \begin{pmatrix} L_{1,1} & & & & & \\ L_{2,2} & L_{1,2} & & & 0 & \\ & \cdots & \cdots & & & \\ & & \cdots & \cdots & & \\ & 0 & & L_{2,N-1} & L_{1,N-1} & \\ L_{3,1} & L_{3,2} & \cdots & \cdots & L_{3,N-1} & L_{3,N} \end{pmatrix}, \quad U = \begin{pmatrix} 1 & U_{1,1} & & & & U_{2,1} \\ & 1 & U_{1,2} & & 0 & U_{2,2} \\ & & \cdots & \cdots & & \cdots \\ & & & \cdots & U_{1,N-2} & \\ & & & & 1 & U_{2,N-1} \\ 0 & & & & & 1 \end{pmatrix}. \tag{3.32}$$

周期境界条件下での LU 分解のアルゴリズム

```
L[1][1] = a[1];
U[1][1] = c[1]/L[1][1];
U[2][1] = b[1]/L[1][1];
for (i = 2; i <= N-2; i++)
{
  L[2][i] = b[i];
  L[1][i] = a[i] - U[1][i-1]*L[2][i];
  U[1][i] = c[i]/L[1][i];
  U[2][i] = - L[2][i]*U[2][i-1]/L[1][i];
}
i = N-1;
L[2][i] = b[i];
L[1][i] = a[i] - U[1][i-1]*L[2][i];
U[2][i] = (c[i] - L[2][i]*U[2][i-1])/L[1][i];
L[3][1] = c[N];
for (i = 2; i <= N-2; i++)
{
  L[3][i] = -U[1][i-1] * L[3][i-1];
}
i = N-1;
L[3][i] = b[N] - U[1][i-1] * L[3][i-1];
sum = 0.0;
for (i = 1; i <= N-1; i++)
{
  sum = sum + L[3][i] * U[2][i];
}
L[3][N] = a[N] - sum;
```

周期境界条件下での $L\boldsymbol{y} = \boldsymbol{b}$, $U\boldsymbol{x} = \boldsymbol{y}$ の解法アルゴリズム

```
y[1] = b[1]/L[1][1];
for (i = 2; i <= N-1; i++)
{
  y[i] = ( b[i] - L[2][i] * y[i-1])/L[1][i];
}
sum = 0.0;
for (i = 1; i <= N-1; i++)
{
  sum = sum + L[3][i]*y[i];
}
y[N] = (b[N] - sum)/L[3][N];
x[N] = y[N];
x[N-1] = y[N-1] - U[2][N-1] * x[N];
for (i = N-2; i >= 1; i-)
{
  x[i] = y[i] - U[1][i] * x[i+1] - U[2][i] * x[N];
}
```

3.3.3 反復改良法 [6]

直接解法では数値計算中の誤差があるために，精度のよい数値解が得られない場合がある．このとき次のような改良をほどこすことによって精度のよい数値解を得ることができる場合がある．

A を LU 分解する：$A = LU$;
$L\boldsymbol{y} = \boldsymbol{b}, U\boldsymbol{x} = \boldsymbol{y}$ を解いて近似解 $\boldsymbol{x} = \boldsymbol{x}^{(0)}$ を求める;
$\boldsymbol{r}^{(0)} := \boldsymbol{b} - A\boldsymbol{x}^{(0)}$;
(反復改良)
For$m := 1, 2, \cdots$ **until** (終了条件) **do**
begin
$L\boldsymbol{y} = \boldsymbol{r}^{(m-1)}, U\boldsymbol{z} = \boldsymbol{y}$ を解いて修正量 $\boldsymbol{z} = \boldsymbol{z}^{(m)}$ を求める;
$\boldsymbol{x}^{(m)} := \boldsymbol{x}^{(m-1)} + z^{(m)}$; $\boldsymbol{r}^{(m)} := \boldsymbol{b} - A\boldsymbol{x}^{(m)}$
end

終了条件は，r_i の丸め誤差限界の評価値を採用する．

$$\delta r_i = (|J_i| + 1.1)\varepsilon_M \left(|b_i| + \sum_{j \in J_i} |a_{i,j}||x_j| \right) \qquad (i = 1, \cdots, N) \tag{3.33}$$

として，$|r_i| \leq \delta r_i \ \ (i = 1, \cdots, N)$ を採用する．ただし，J_i は A の第 i 行に非零要素のある行番号の集合 (はその要素数) を表し，ε_M はマシンイプシロンを表す[3]．といっても $||\boldsymbol{r}^{(m)}|| < 10^{-8}$ で十分だと思う．

3.3.3.1 SSI(Symmetrical Semi-Implicit) 法 [5]

無条件安定な差分にもかかわらず，行列計算が不必要な差分法である．

[3] 行列計算の精度に問題があるかどうか確認するためにも使うことができる．

$$
\begin{aligned}
u_i^{n+1} - u_i^n \quad = \quad & -D\frac{\Delta t}{\Delta x^2}\left(2u_i^{n+1} - \left(\frac{3}{2}u_{i+1}^n + u_i^n + \frac{3}{2}u_{i-1}^n\right)\right. \\
& \left. + \left(\frac{1}{2}u_{i+1}^{n-1} + u_i^{n-1} + \frac{1}{2}u_{i-1}^{n-1}\right)\right). \qquad (3.34)
\end{aligned}
$$

この方法には 1 つだけ明らかな問題点がある．それは u_i^1 を求めるとき u_i^{-1} が必要になり，そのようなデータを与えることはできないことである．陽解法あるいは陰解法を使って u_i^1 を計算し，(u_i^0, u_i^1) を用いて $n = 2$ 以降に SSI 法を使うことでこの問題を回避する必要がある．この方法がどの程度有効なのか確認していないので，誰か確認してほしい[4]．

3.3.4　問題

問題 1．熱方程式

$$
\begin{cases}
\dfrac{\partial u}{\partial t} = \dfrac{\partial^2 u}{\partial x^2}, \quad 0 < x < 1,\ t > 0, \\
u(0, x) = 4\sin(\pi x), \quad 0 \le x \le 1, \quad \text{（初期条件）}, \\
u(t, 0) = 0, \quad u(t, 1) = 0, \quad t > 0, \quad \text{（Dirichlet 境界条件）}.
\end{cases} \qquad (3.35)
$$

(3.35) の理論解は

$$
u(x, t) = 4\sin(\pi x)\exp(-\pi^2 t) \qquad (3.36)
$$

なっている．理論解と陽解法，陰解法，SSI 法の数値解を比較してみましょう．

[4] 一度試してみたことがあるが，あまり好ましい数値解を得られなかった（時間差分を大きく取ると数値解が波打つような解になった.）

3.4 2 次元長方形領域における拡散方程式の数値計算

$$\frac{\partial u}{\partial t} = D_x \frac{\partial^2 u}{\partial x^2} + D_y \frac{\partial^2 u}{\partial y^2}, \quad t > 0,\ (x, y) \in (0, L_x) \times (0, L_y). \tag{3.37}$$

3.4.1 陽解法 [4]

$$\frac{u_{i,j}^{n+1} - u_{i,j}^n}{\Delta t} = \frac{D_x}{\Delta x^2}(u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n) + \frac{D_y}{\Delta y^2}(u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n).$$

陽解法の安定性条件は

$$D_x \frac{\Delta t}{\Delta x^2} + D_y \frac{\Delta t}{\Delta y^2} \leq \frac{1}{2}. \tag{3.38}$$

従って，1 次元問題より Δt を小さく取る必要が生じる．実用上は使えないと思うだろうが実際には使うことがある．空間 2 次元になると数値計算の実行時間の問題により $\Delta x, \Delta y$ を大きくした計算をやらざるを得なくなり，(3.38) の条件を満足するからだ！

3.4.2 陰解法 [4]

$$\begin{aligned} \frac{u_{i,j}^{n+1} - u_{i,j}^n}{\Delta t} &= \frac{D_x}{\Delta x^2}(u_{i+1,j}^{n+1} - 2u_{i,j}^{n+1} + u_{i-1,j}^{n+1}) \\ &+ \frac{D_y}{\Delta y^2}(u_{i,j+1}^{n+1} - 2u_{i,j}^{n+1} + u_{i,j-1}^{n+1}). \end{aligned} \tag{3.39}$$

1 次元問題と同様にやはり安定な数値計算法であるが，次のような連立一次方程式を計算しなければならない：

$$A\boldsymbol{u}^{n+1} = \boldsymbol{u}^n. \tag{3.40}$$

2 次元領域の場合，行列 A は疎な 5 重対角行列となっており，直接解法（LU 分解法）を用いると，行列が密行列となってしまい，計算に要する時間が膨大になってしまう．この問題を解決する方法として，CG 法等の反復解法による連立一次方程式の数値計算法がある[5]．CG 法等を用いた数値計算法は発展編において解説する．

[5] 今回の講義では解説しない．

3.4.3　ADI 法 [5]

$$\begin{aligned}
\frac{u_{i,j}^{n+1/2}-u_{i,j}^{n}}{\Delta t/2} &= \frac{D_x}{\Delta x^2}(u_{i+1,j}^{n+1/2}-2u_{i,j}^{n+1/2}+u_{i-1,j}^{n+1/2})+\frac{D_y}{\Delta y^2}(u_{i,j+1}^{n}-2u_{i,j}^{n}+u_{i,j-1}^{n}),\\
\frac{u_{i,j}^{n+1}-u_{i,j}^{n+1/2}}{\Delta t/2} &= \frac{D_x}{\Delta x^2}(u_{i+1,j}^{n+1/2}-2u_{i,j}^{n+1/2}+u_{i-1,j}^{n+1/2})+\frac{D_y}{\Delta y^2}(u_{i,j+1}^{n+1}-2u_{i,j}^{n+1}+u_{i,j-1}^{n+1}).
\end{aligned}$$

ADI 法は無条件安定であることが知られている．証明が気になる方には桂田先生（明治大学）の資料が参考になる [13]．ADI 法は 1 次元の陰解法のアルゴリズムがそのまま使えるので，プログラムを作るのが大変楽である．

ADI 法のアルゴリズム
For $k := 1, 2, \cdots$ **until** $times$ **do**
begin
  For $j := 0, 1, 2, \cdots$ **until** N_y **do**
  begin
    $A_x \boldsymbol{u}_j^{n+1} = B_j \boldsymbol{u}_j^n$
  end
  For $i := 0, 1, 2, \cdots$ **until** N_x **do**
  begin
    $A_y \boldsymbol{u}_i^{n+1} = B_i \boldsymbol{u}_i^n$
  end
end

3.4.4　2 次元 SSI 法 [5]

$$\begin{aligned}
\left(1+2D\frac{\Delta t}{\Delta x^2}+2D\frac{\Delta t}{\Delta y^2}\right)u_{i,j}^{n+1} &= u_{i,j}^{n}+D\frac{\Delta t}{\Delta x^2}\left(\frac{3}{2}u_{i+1,j}^{n}+u_{i,j}^{n}+\frac{3}{2}u_{i-1,j}^{n}\right)\\
&- D\frac{\Delta t}{\Delta x^2}\left(\frac{1}{2}u_{i+1,j}^{n-1}+u_{i,j}^{n-1}+\frac{1}{2}u_{i-1,j}^{n-1}\right)\\
&+ D\frac{\Delta t}{\Delta y^2}\left(\frac{3}{2}u_{i,j+1}^{n}+u_{i,j}^{n}+\frac{3}{2}u_{i,j-1}^{n}\right)\\
&- D\frac{\Delta t}{\Delta y^2}\left(\frac{1}{2}u_{i,j+1}^{n-1}+u_{i,j}^{n-1}+\frac{1}{2}u_{i,j-1}^{n-1}\right). \qquad (3.41)
\end{aligned}$$

この解法がどこまで有効なのかわからない．一度試してみたい気がする．

3.5 3 次元直方体領域での数値計算法

$$\frac{\partial u}{\partial t} = D_x \frac{\partial^2 u}{\partial x^2} + D_y \frac{\partial^2 u}{\partial y^2} + D_z \frac{\partial^2 u}{\partial z^2}, \quad t > 0,\ (x, y, z) \in (0, L_x) \times (0, L_y) \times (0, L_z). \tag{3.42}$$

3.5.1 陽解法

3 次元問題だと使う場合が結構ある．私も 3 次元問題は陽解法を使うことがある．

$$\begin{aligned}
\frac{u_{i,j,k}^{n+1} - u_{i,j,k}^{n}}{\Delta t} &= \frac{D_x}{\Delta x^2}(u_{i+1,j,k}^{n} - 2u_{i,j,k}^{n} + u_{i-1,j,k}^{n}) \\
&+ \frac{D_y}{\Delta y^2}(u_{i,j+1,k}^{n} - 2u_{i,j,k}^{n} + u_{i,j-1,k}^{n}) \\
&+ \frac{D_z}{\Delta z^2}(u_{i,j,k+1}^{n} - 2u_{i,j,k}^{n} + u_{i,j,k+1}^{n}).
\end{aligned}$$

陽解法の安定性条件は

$$D_x \frac{\Delta t}{\Delta x^2} + D_y \frac{\Delta t}{\Delta y^2} + D_z \frac{\Delta t}{\Delta z^2} \le \frac{1}{2}. \tag{3.43}$$

従って，２次元問題より Δt を小さく取る必要が生じるが，２次元問題と同様の理由で使うことが多い．

3.5.2 ADI 法 [5]

無条件安定なので安心して使うことができる !!

$$\begin{aligned}
\frac{u_{i,j,k}^{n+1/3} - u_{i,j,k}^{n}}{\Delta t} &= \frac{D_x}{2\Delta x^2}(u_{i+1,j,k}^{n+1/3} - 2u_{i,j,k}^{n+1/3} + u_{i-1,j,k}^{n+1/3} + u_{i+1,j,k}^{n} - 2u_{i,j,k}^{n} + u_{i-1,j,k}^{n}) \\
&+ \frac{D_y}{\Delta y^2}(u_{i,j+1,k}^{n} - 2u_{i,j,k}^{n} + u_{i,j-1,k}^{n}) \\
&+ \frac{D_z}{\Delta z^2}(u_{i,j,k+1}^{n} - 2u_{i,j,k}^{n} + u_{i,j,k+1}^{n}), \\
\\
\frac{u_{i,j,k}^{n+2/3} - u_{i,j,k}^{n}}{\Delta t} &= \frac{D_x}{2\Delta x^2}(u_{i+1,j,k}^{n+1/3} - 2u_{i,j,k}^{n+1/3} + u_{i-1,j,k}^{n+1/3} + u_{i+1,j,k}^{n} - 2u_{i,j,k}^{n} + u_{i-1,j,k}^{n}) \\
&+ \frac{D_y}{2\Delta y^2}(u_{i,j+1,k}^{n+2/3} - 2u_{i,j,k}^{n+2/3} + u_{i,j-1,k}^{n+2/3} + u_{i,j+1,k}^{n} - 2u_{i,j,k}^{n} + u_{i,j-1,k}^{n}) \\
&+ \frac{D_z}{\Delta z^2}(u_{i,j,k+1}^{n} - 2u_{i,j,k}^{n} + u_{i,j,k+1}^{n}), \\
\\
\frac{u_{i,j,k}^{n+1} - u_{i,j,k}^{n}}{\Delta t} &= \frac{D_x}{2\Delta x^2}(u_{i+1,j,k}^{n+1/3} - 2u_{i,j,k}^{n+1/3} + u_{i-1,j,k}^{n+1/3} + u_{i+1,j,k}^{n} - 2u_{i,j,k}^{n} + u_{i-1,j,k}^{n}) \\
&+ \frac{D_y}{2\Delta y^2}(u_{i,j+1,k}^{n+2/3} - 2u_{i,j,k}^{n+2/3} + u_{i,j-1,k}^{n+2/3} + u_{i,j+1,k}^{n} - 2u_{i,j,k}^{n} + u_{i,j-1,k}^{n}) \\
&+ \frac{D_z}{2\Delta z^2}(u_{i,j,k+1}^{n+1} - 2u_{i,j,k}^{n+1} + u_{i,j,k-1}^{n+1} + u_{i,j,k+1}^{n} - 2u_{i,j,k}^{n} + u_{i,j,k-1}^{n}).
\end{aligned}$$

3.5.3　陰解法

$$
\begin{aligned}
\frac{u_{i,j,k}^{n+1}-u_{i,j,k}^{n}}{\Delta t} &= \frac{D_x}{\Delta x^2}(u_{i+1,j,k}^{n+1}-2u_{i,j,k}^{n+1}+u_{i-1,j,k}^{n+1}) \\
&+ \frac{D_y}{\Delta y^2}(u_{i,j+1,k}^{n+1}-2u_{i,j,k}^{n+1}+u_{i,j-1,k}^{n+1}) \\
&+ \frac{D_z}{\Delta z^2}(u_{i,j,k+1}^{n+1}-2u_{i,j,k}^{n+1}+u_{i,j,k+1}^{n+1}).
\end{aligned}
$$

この解法も無条件安定であることがわかるが，七重対角行列の数値計算を行なわなければならい．この場合はやはり CG 法等の反復法をを使わなければ計算速度が遅くなる．陰解法を用いた数値計算法は発展編において解説する．

第4章　反応拡散方程式の数値計算法（差分法）

本章では，次のような反応拡散系の数値計算方法を解説する：

$$\boldsymbol{u}_t = \boldsymbol{D}\Delta\boldsymbol{u} + \boldsymbol{f}(\boldsymbol{u}), \quad t > 0, \quad \boldsymbol{x} \in \boldsymbol{R}^n. \tag{4.1}$$

ただし，

$$\begin{aligned}
&\boldsymbol{x} = (x_1, x_2, \cdots, x_n), \quad \boldsymbol{u} = (u_1, u_2, \cdots, u_m), \quad \boldsymbol{f} = (f_1, f_2, \cdots, f_m),\\
&\boldsymbol{u} : \boldsymbol{R}_+ \times \boldsymbol{R}^n \longrightarrow \boldsymbol{R}^m,\\
&\boldsymbol{D} : \text{非負対角行列}, \quad \Delta = \sum_{k=1}^{n} \frac{\partial^2}{\partial x_k^2}.
\end{aligned}$$

(4.1) で表されるものを**反応拡散系** という．

例：FitzHugh-Nagumo 方程式 [25]

$$\begin{cases} u_t = u_{xx} + \dfrac{1}{\varepsilon}\left(u(1-u)(u-a) - v\right), \\ v_t = u - \gamma v. \end{cases} \tag{4.2}$$

ただし，$x \in \boldsymbol{R}$, $\quad u, v : \boldsymbol{R}_+ \times \boldsymbol{R} \longrightarrow \boldsymbol{R}$.

4.1　１次元反応拡散系の数値計算法

ここでは 1 次元反応拡散系の初期値境界値問題の数値計算法について解説する．

4.1.1　１次元反応拡散系の初期・境界値問題

$$\begin{cases} \dfrac{\partial \boldsymbol{u}}{\partial t} = \boldsymbol{D}\dfrac{\partial^2}{\partial x^2}\boldsymbol{u} + \boldsymbol{f}(\boldsymbol{u}), \quad t > 0, \quad x \in I = (0, L), \\ \boldsymbol{u}(0, x) = \boldsymbol{u}_0(x), \quad x \in \bar{I}, \\ \text{境界条件}. \end{cases} \tag{4.3}$$

4.1.1.1　境界条件

1. Dirichlet 条件

$$\boldsymbol{u}(t, 0) = \boldsymbol{a}, \quad \boldsymbol{u}(t, L) = \boldsymbol{b}. \tag{4.4}$$

   - 多くの場合，$\boldsymbol{a}, \boldsymbol{b}$ は $\boldsymbol{f}(\boldsymbol{u}) = 0$ の安定平衡点にとる．

2. Neumann 条件

$$\frac{\partial \boldsymbol{u}}{\partial x}(t, 0) = \boldsymbol{a}, \quad \frac{\partial \boldsymbol{u}}{\partial x}(t, L) = \boldsymbol{b}. \tag{4.5}$$

   - 多くの場合，$\boldsymbol{a} = \boldsymbol{b} = \boldsymbol{0}$.
   - その場合，流量なし条件とか断熱条件という．

3. 周期境界条件

$$\begin{cases} \boldsymbol{u}(t,0) = \boldsymbol{u}(t,L), \\ \dfrac{\partial \boldsymbol{u}}{\partial x}(t,0) = \dfrac{\partial \boldsymbol{u}}{\partial x}(t,L). \end{cases} \tag{4.6}$$

- 物理屋がよく使う.
- 進行波解（一定速度，一定波形）の性質を調べるときに用いる.

4. 第 3 種境界条件（滅多に使わない）

$$\begin{cases} \dfrac{\partial \boldsymbol{u}}{\partial x}(t,0) = -\alpha_1 \left(\boldsymbol{a} - \beta_1 \boldsymbol{u}(t,0) \right), \\ \dfrac{\partial \boldsymbol{u}}{\partial x}(t,L) = \alpha_2 \left(\boldsymbol{b} - \beta_2 \boldsymbol{u}(t,L) \right). \end{cases} \tag{4.7}$$

- $\alpha_i \to 0 \quad (i = 1, 2)$ ならば，Neumann 条件と同じになる.
- $\alpha_i \to \infty \quad (i = 1, 2)$ ならば，Dirichlet 条件と同じになる.

4.1.2　２変数反応拡散系の数値計算法 (半陰解法)

$$\begin{cases} u_t = d_u u_{xx} + f(u,v), \\ v_t = d_v v_{xx} + g(u,v). \end{cases} \tag{4.8}$$

を
初期条件　$u(0,x) = u_0(x), \quad v(0,x) = v_0(x)$
および
境界条件　上記，(4.4)-(4.7) のいずれか.
のものとで数値計算をする方法を説明する.

4.1.2.1　方程式の離散化 (差分化)

- 熱方程式の部分を陰解法
- 非線形項 f, g の部分は陽解法

設定：$\Delta x = \dfrac{L}{N}, \quad x_i = i \times \Delta x (i = 1, 2, \cdots, N), \quad t_n = n \times \Delta t$（適切な値）

$$\begin{aligned} & \frac{u(t_{n+1}, x_i) - u(t_n, x_i)}{\Delta t} \\ = \quad & d_u \frac{u(t_{n+1}, x_{i+1}) - 2u(t_{n+1}, x_i) + u(t_{n+1}, x_{i-1})}{Deltax^2} + f\big(u(t_n, x_i), v(t_n, x_i)\big). \end{aligned} \tag{4.9}$$

v に関しても同様である．ここで，次のように書くことにする：

$$u(t_n, x_i) = u_i^n, \quad v(t_n, x_i) = v_i^n.$$

このとき (4.8) の差分方程式は

$$\begin{cases} u_i^{n+1} - u_i^n = \dfrac{\Delta t}{\Delta x^2} d_u (u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}) + \Delta t f(u_i^n, v_i^n), \\ v_i^{n+1} - v_i^n = \dfrac{\Delta t}{\Delta x^2} d_v (v_{i+1}^{n+1} - 2v_i^{n+1} + v_{i-1}^{n+1}) + \Delta t g(u_i^n, v_i^n) \end{cases} \tag{4.10}$$

と書くことができる．ここで，$r = \dfrac{\Delta t}{\Delta x^2}$ とおくと

$$\Longrightarrow \begin{cases} -r d_u u_{i-1}^{n+1} + (1 + 2r d_u) u_i^{n+1} - r d_u u_{i+1}^{n+1} = u_i^n + \Delta t f(u_i^n, v_i^n), \\ -r d_u v_{i-1}^{n+1} + (1 + 2r d_u) v_i^{n+1} - r d_u v_{i+1}^{n+1} = v_i^n + \Delta t g(u_i^n, v_i^n) \end{cases} \tag{4.11}$$

となる．
$n = 0$ のとき

$$\begin{cases} u_i^0 = u(0, x_i) = u_0(x_i), \\ v_i^0 = v(0, x_i) = v_0(x_i) \end{cases} \tag{4.12}$$

であるから，初期条件より右辺の値は確定する．よって，(4.11) の数値計算をすることで，順次求まる．

4.1.2.2 境界条件の離散化

1. Dirichlet 条件

$$\begin{cases} u(t,0) = a_u, \quad u(t,L) = b_u, \\ v(t,0) = a_v, \quad v(t,L) = b_v. \end{cases}$$

$$\Longrightarrow \begin{cases} u_0^n = a_u, \quad u_N^n = b_u, \\ v_0^n = a_v, \quad v_N^n = b_v. \end{cases} \tag{4.13}$$

2. Neumann 条件 (4.5) から

$$\begin{cases} \dfrac{\partial u}{\partial x}(t,0) = a_u, \\ \dfrac{\partial u}{\partial x}(t,N) = b_u. \end{cases}$$

$$\Rightarrow \begin{cases} \dfrac{u_1^n - u_{-1}^n}{2\Delta x} + O(\Delta x^2) = a_u, \\ \dfrac{u_{N+1}^n - u_{N-1}^n}{2\Delta x} + O(\Delta x^2) = b_u. \end{cases}$$

従って

$$\begin{cases} u_{-1}^n = u_1^n - 2\Delta x a_u, \\ u_{N+1}^n = u_{N-1}^n - 2\Delta x b_u. \end{cases} \tag{4.14}$$

と離散化することができる．v についても同様に

$$\begin{cases} v_{-1}^n = v_1^n - 2\Delta x a_v, \\ v_{N+1}^n = v_{N-1}^n - 2\Delta x b_v \end{cases} \tag{4.15}$$

とできる．

3. 周期境界条件 (4.6) より

$$u(t,0) = u(t,L) \Longrightarrow u_0^n = u_N^n. \tag{4.16}$$

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

$\Longrightarrow$

$$\frac{u_1^n - u_{-1}^n}{2\Delta x} + O(\Delta x^2) = \frac{u_{N+1}^n - u_{N-1}^n}{2\Delta x} + O(\Delta x^2).$$

従って

$$u_{-1}^n = u_{N-1}^n$$

となる．以上をまとめると

$$u_0^n = u_N^n, \quad u_{-1}^n = u_{N-1}^n \tag{4.17}$$

となる．v についても同様に

$$v_0^n = v_N^n, \quad v_{-1}^n = v_{N-1}^n \tag{4.18}$$

となる．

4. 第 3 種境界条件

$$\begin{cases} \dfrac{\partial u}{\partial x}(t,0) = \alpha_{u1}\left(a_u - \beta_{u1} u(t,0)\right), \\ \dfrac{\partial u}{\partial x}(t,L) = \alpha_{u2}\left(b_u - \beta_{u2} u(t,L)\right). \end{cases}$$

$$\Longrightarrow \begin{cases} \dfrac{u_1^n - u_{-1}^n}{2\Delta x} + O(\Delta x^2) = \alpha_{u1}\left(a_u - \beta_{u1} u_0^n\right), \\ \dfrac{u_{N+1}^n - u_{N-1}^n}{2\Delta x} + O(\Delta x^2) = \alpha_{u1}\left(b_u - \beta_{u2} u_N^n\right). \end{cases}$$

従って

$$\begin{cases} u_{-1}^n = u_1^n - 2\Delta x \alpha_{u1}\left(a_u - \beta_{u1} u_0^n\right), \\ u_{N+1}^n = u_{N-1}^n + 2\Delta x \alpha_{u2}\left(b_u - \beta_{u2} u_N^n\right). \end{cases} \tag{4.19}$$

4.1.3 差分方程式の連立 1 次方程式表示

4.1.3.1 Dirichlet 条件のとき

$i = 0, N$ での値はすでに求まっているので，$u_0^{n+1}, u_N^{n+1}, v_0^{n+1}, v_N^{n+1}$ は解かなくてもよい．$i = 1, \cdots, N-1$ で (4.11) を解けばよい．

(4.13) より，$i = 1$ のとき

$$-rd_u a_u + (1 + 2rd_u)u_1^{n+1} - rd_u u_2^{n+1} = u_1^n + \Delta t f(u_1^n, v_1^n)$$

となる．

(4.13) より，$i = N - 1$ のとき

$$-rd_u u_{N-2}^{n+1} + (1 + 2rd_u)u_{N-1}^{n+1} - rd_u b_u = u_{N-1}^n + \Delta t f(u_{N-1}^n, v_{N-1}^n)$$

となる．$2 < i < N - 2$ のときは，(4.11) と同じ差分方程式となる．従って，計算すべき連立 1 次方程式は次のようになる：

$$\begin{pmatrix} 1+2rd_u & -rd_u & 0 & 0 & 0 & \dots & 0 \\ -rd_u & 1+2rd_u & -rd_u & 0 & 0 & \dots & 0 \\ 0 & -rd_u & 1+2rd_u & -rd_u & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & -rd_u & 1+2rd_u \end{pmatrix} \begin{pmatrix} u_1^{n+1} \\ u_2^{n+1} \\ \vdots \\ u_{N-2}^{n+1} \\ u_{N-1}^{n+1} \end{pmatrix}$$

$$= \begin{pmatrix} u_1^n + \Delta t f(u_1^n, v_1^n) + rd_u a_u \\ \vdots \\ u_i^n + \Delta t f(u_i^n, v_i^n) \\ \vdots \\ u_{N-1}^n + \Delta t f(u_{N-1}^n, v_{N-1}^n) + rd_u b_u \end{pmatrix}. \tag{4.20}$$

v についても同様である．

4.1.3.2　Neumann 条件のとき

$0 \leq i \leq N$ において，(4.11) を解く．
(4.14) より，$i = 0$ のとき

$$
\begin{aligned}
&-rd_u(u_1^{n+1} - 2\Delta x a_u) + (1 + 2rd_u)u_0^{n+1} - rd_u u_1^{n+1} = u_0^n + \Delta t f(u_0^n, v_0^n) \\
&\Longleftrightarrow \\
&(1 + 2rd_u)u_0^{n+1} - 2rd_u u_1^{n+1} = u_0^n + \Delta t f(u_0^n, v_0^n) - 2rd_u \Delta x a_u
\end{aligned} \tag{4.21}
$$

となる．
(4.14) より，$i = N$ のとき

$$
\begin{aligned}
&-rd_u u_{N-1}^n + (1 + 2rd_u)u_N^{n+1} - rd_u(u_{N-1}^n + 2\Delta x b_u) = u_N^n + \Delta t f(u_N^n, v_N^n) \\
&\Longleftrightarrow \\
&-2rd_u u_{N-1}^n + (1 + 2rd_u)u_N^{n+1} = u_N^n + \Delta t f(u_N^n, v_N^n) + 2rd_u \Delta x b_u
\end{aligned} \tag{4.22}
$$

となる．$i = 1, \cdots, N-1$ のときは，(4.11) と同じ差分方程式となる．従って，計算すべき連立 1 次方程式は次のようになる：

$$
\begin{pmatrix}
\frac{1}{2}(1 + 2rd_u) & -rd_u & 0 & 0 & 0 & \dots & 0 \\
-rd_u & 1 + 2rd_u & -rd_u & 0 & 0 & \dots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & 0 & 0 & 0 & -rd_u & 1 + 2rd_u & -rd_u \\
0 & 0 & 0 & 0 & 0 & -rd_u & \frac{1}{2}(1 + 2rd_u)
\end{pmatrix}
\begin{pmatrix}
u_0^{n+1} \\ u_1^{n+1} \\ \vdots \\ u_{N-1}^{n+1} \\ u_N^{n+1}
\end{pmatrix}
$$

$$
= \begin{pmatrix}
\frac{1}{2}\left(u_0^n + \Delta t f(u_0^n, v_0^n) - 2rd_u \Delta x a_u\right) \\
\vdots \\
u_i^n + \Delta t f(u_i^n, v_i^n) \\
\vdots \\
\frac{1}{2}\left(u_N^n + \Delta t f(u_N^n, v_N^n) + 2rd_u \Delta x b_u\right)
\end{pmatrix}. \tag{4.23}
$$

v についても同様である．

4.1.3.3 周期境界条件のとき

(4.17) から $u_0^{n+1} = u_N^{n+1}$ となるので，$u_i^{n+1} \quad (0 \le i \le N-1)$ において，(4.11) を解けばよい．
(4.17) より，$i = 0$ のとき

$$
\begin{aligned}
&-rd_u u_{N-1}^{n+1} + (1+2rd_u)u_0^{n+1} - rd_u u_1^{n+1} = u_0^n + \Delta t f(u_0^n, v_0^n) \\
\Longleftrightarrow \\
&(1+2rd_u)u_0^{n+1} - rd_u u_1^{n+1} - rd_u u_{N-1}^{n+1} = u_0^n + \Delta t f(u_0^n, v_0^n)
\end{aligned} \tag{4.24}
$$

となる．
(4.17) より，$i = N-1$ のとき

$$
\begin{aligned}
&-rd_u u_{N-2}^{n} + (1+2rd_u)u_{N-1}^{n+1} - rd_u u_1^{n} = u_N^n + \Delta t f(u_{N-1}^n, v_{N-1}^n) \\
\Longleftrightarrow \\
&-rd_u u_1^{n} - rd_u u_{N-2}^{n} + (1+2rd_u)u_{N-1}^{n+1} = u_N^n + \Delta t f(u_{N-1}^n, v_{N-1}^n)
\end{aligned} \tag{4.25}
$$

となる．$i = 1, \cdots, N-2$ のときは，(4.11) と同じ差分方程式となる．従って，計算すべき連立 1 次方程式は次のようになる：

$$
\begin{pmatrix}
1+2rd_u & -rd_u & 0 & 0 & \cdots & 0 & -rd_u \\
-rd_u & 1+2rd_u & -rd_u & 0 & \cdots & 0 & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & -rd_u & 1+2rd_u & -rd_u \\
-rd_u & 0 & \cdots & 0 & 0 & -rd_u & 1+2rd_u
\end{pmatrix}
\begin{pmatrix}
u_0^{n+1} \\ u_1^{n+1} \\ \vdots \\ \vdots \\ u_{N-2}^{n+1} \\ u_{N-1}^{n+1}
\end{pmatrix}
$$

$$
= \begin{pmatrix}
u_0^n + \Delta t f(u_0^n, v_0^n) \\
\vdots \\
u_i^n + \Delta t f(u_i^n, v_i^n) \\
\vdots \\
\vdots \\
u_{N-1}^n + \Delta t f(u_{N-1}^n, v_{N-1}^n)
\end{pmatrix}. \tag{4.26}
$$

v についても同様である．

4.1.3.4 問題

問題 1．神経繊維上の電位の伝播モデル (FitzHugh-Nagumo 方程式)[25]

$$
\begin{cases}
\dfrac{du}{dt} = d_u u_{xx} + \dfrac{1}{\varepsilon}\left(u(1-u)(u-a) - v\right), \quad t > 0,\ 0 < x < L \\
\dfrac{dv}{dt} = u - \gamma v.
\end{cases} \tag{4.27}
$$

$\gamma = 1.0, \varepsilon = 0.001, a = 0.125, d_u = 1.0, d_v = 0.1, L = 20.0$ とおいて数値計算を行ってみなさい．ただし，境界条件は Neumann 境界条件

$$
\frac{\partial u}{\partial x}(t, 0) = \frac{\partial u}{\partial x}(t, L) = 0, \quad t > 0
$$

と周期境界条件

$$u(t,0) = u(t,L), \quad \frac{\partial u}{\partial x}(t,0) = \frac{\partial u}{\partial x}(t,L), \quad t > 0,$$

の両方の場合に行なってみなさい．そして d_v を大きくとると (例えば，$d_v = 10.0$) どうなるか数値実験してみなさい．

4.1.3.5　注意

陰解法は 数値スキームが安定なので，Δt を大きく取ることができる．しかし,「スキームが安定なこと」と「モデル方程式の解挙動を近似すること」は同じ意味ではないことを忘れてはいけない．Δt を大きくしたのではゴースト解[1]を見てしまう恐れがある．また，陰解法が最も有効な場合は解の漸近挙動 (時間が十分経過した後) を見ることである．

[1]差分方程式の解であって連続方程式の解でないもの，$\Delta t \to 0, \Delta x \to 0 (\Delta t/\Delta x^2$は一定) とすると見えなくなってしまう

4.2 ２次元反応拡散系の数値計算法

ここでは無条件安定な数値計算法の一つである ADI 法 [5] を用いた数値計算法を解説する．

4.2.1 方程式の離散化

次の２変数反応拡散方程式系を考える：

$$\begin{cases} u_t = d_u \Delta u + f(u,v), \\ v_t = d_v \Delta v + g(u,v), \end{cases} \quad t > 0, \quad \boldsymbol{x} \in \Omega. \tag{4.28}$$

ただし，Ω は長方形領域 $\Omega = (0, L_x) \times (0, L_y)$ とし

$$\begin{cases} u = u(t, \boldsymbol{x}) = u(t, x, y), \\ v = v(t, \boldsymbol{x}) = v(t, x, y), \end{cases} \quad \boldsymbol{x} \in \bar{\Omega}. \tag{4.29}$$

と書く．
初期条件は

$$\begin{cases} u(0, x, y) = u_0(x, y), \\ \\ v(0, x, y) = v_0(x, y), \end{cases} \quad (x, y) \in \bar{\Omega}. \tag{4.30}$$

とする．ここで

$$\Delta x = \frac{L_x}{Nx}, \quad \Delta y = \frac{L_y}{Ny} (\Delta x = \Delta y), \quad x_i = i \times \Delta x, \quad y_j = j \times \Delta y$$

とおき，Δt を与えて

$$t_n = n \times \Delta t$$

とおく．以降，$u(t_n, x_i, y_j) = u^n_{i,j}$ と書くことにする．(4.28) の離散化を行なう．２階微分を次の中心差分で離散化を行なう：

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2} = \dfrac{u(t, x_i + \Delta x, y_j) - 2u(t, x_i, y_j) + u(t, x_i - \Delta x, y_j)}{\Delta x^2} + O(\Delta x^2), \\ \dfrac{\partial^2 u}{\partial y^2} = \dfrac{u(t, x_i, y_j + \Delta y) - 2u(t, x_i, y_j) + u(t, x_i, y_j - \Delta y)}{\Delta y^2} + O(\Delta y^2). \end{cases} \tag{4.31}$$

さらに，時間に関しては次のように前進差分で離散化する：

$$\begin{aligned} \frac{\partial u}{\partial t} &= \frac{u(t_n + \Delta t, x_i, y_j) - u(t_n, x_i, y_j)}{\Delta t} + O(\Delta t) \\ &\approx \frac{u^{n+1}_{i,j} - u^n_{i,j}}{\Delta t}. \end{aligned}$$

- 陽解法：(4.31) において，$t = t_n$ とする．このとき，数値解法の安定性より

  $$\Delta t \Big(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} \Big) \leq \frac{1}{2}$$

  の条件が必要である．

- 陰解法：(4.31) において，$t = t_{n+1}$ とする．このとき，無条件安定であるが，$(Nx+1)(Ny+1) \times (Nx+1)(Ny+1)$ 行列の５重対角連立１次方程式の数値解法が必要になってくる．

4.2.2　非線形項の離散化

$$\begin{cases} f(u,v) = f(u_{i,j}^n, v_{i,j}^n), \\ \\ g(u,v) = g(u_{i,j}^n, v_{i,j}^n). \end{cases} \tag{4.32}$$

4.2.3　初期条件の離散化

$$\begin{cases} u_{i,j}^0 = u_0(x_i, y_j), \\ \\ v_{i,j}^0 = v_0(x_i, y_j). \end{cases} \tag{4.33}$$

4.2.4　境界条件の離散化

4.2.4.1　斉次 Neumann 境界条件

ここで，境界条件は次のような斉次 Neumann 境界条件を与える：

$$\frac{\partial u}{\partial n} = \frac{\partial v}{\partial n} = 0, \quad t > 0, \quad \boldsymbol{x} \in \partial\Omega \tag{4.34}$$

とするただし，n は $\partial\Omega$ の外向き法線ベクトルとする．$\Omega = (0, L_x) \times (0, L_y)$ のとき，式 (4.34) は

$$\begin{cases} \dfrac{\partial u}{\partial y}(x,0) = \dfrac{\partial u}{\partial y}(x, L_y) = 0, \quad x \in (0, L_x), \\ \dfrac{\partial u}{\partial x}(0,y) = \dfrac{\partial u}{\partial x}(L_x, y) = 0, \quad y \in (0, L_y) \end{cases} \tag{4.35}$$

と書ける．
このとき，境界条件の離散化は

$$\begin{cases} \dfrac{\partial u}{\partial x}(t,0,y) \approx \dfrac{u(t,x_1,y_j) - u(t,x_{-1},y_j)}{2\Delta x}, \quad (0 \le j \le Ny), \\ \dfrac{\partial u}{\partial x}(t,L_x,y) \approx \dfrac{u(t,x_{Nx+1},y_j) - u(t,x_{Nx-1},y_j)}{2\Delta x}, \quad (0 \le j \le Ny), \\ \dfrac{\partial u}{\partial y}(t,x,0) \approx \dfrac{u(t,x_i,y_1) - u(t,x_i,y_{-1})}{2\Delta y}, \quad (0 \le i \le Nx), \\ \dfrac{\partial u}{\partial y}(t,x,L_y) \approx \dfrac{u(t,x_i,y_{Ny+1}) - u(t,x_i,y_{Ny-1})}{2\Delta y}, \quad (0 \le i \le Nx) \end{cases} \tag{4.36}$$

となる．(4.36) より

$$\begin{cases} u_{-1,j}^n = u_{1,j}^n, \quad u_{N_x+1,j}^n = u_{N_x-1,j}^n, \quad (0 \le j \le Ny), \\ \\ u_{i,-1}^n = u_{i,1}^n, \quad u_{i,N_y+1}^n = u_{i,N_y-1}^n, \quad (0 \le i \le Nx) \end{cases} \tag{4.37}$$

となる．

4.2.4.2 周期境界条件

境界条件が周期境界条件ならば

$$\begin{cases} u(0,y) = u(L_x,y), \quad u(x,0) = u(x,L_y), \\ \dfrac{\partial u}{\partial x}(0,y) = \dfrac{\partial u}{\partial x}(L_x,y), \quad \dfrac{\partial u}{\partial y}(x,0) = \dfrac{\partial u}{\partial y}(x,L_y) \end{cases} \tag{4.38}$$

となり，その離散化は

$$\begin{cases} u(t,x_0,y_j) = u(t,x_{Nx},y_j) \Longrightarrow u^n_{0,j} = u^n_{Nx,j}, \quad (0 \le j \le Ny), \\ u(t,x_i,y_0) = u(t,x_i,y_{Ny}) \Longrightarrow u^n_{i,0} = u^n_{i,Ny}, \quad (0 \le i \le Nx), \\ u(t,x_{-1},y_j) = u(t,x_{Nx-1},y_j) \Longrightarrow u^n_{-1,j} = u^n_{Nx-1,j}, \quad (0 \le j \le Ny), \\ u(t,x_i,y_{-1}) = u(t,x_i,y_{Ny-1}) \Longrightarrow u^n_{i,-1} = u^n_{i,Ny-1}, (0 \le i \le Nx) \end{cases} \tag{4.39}$$

となる.

4.2.5 ADI 法

ここでは，無条件安定な計算法かつ空間 1 次元の陰解法の応用で数値計算可能な ADI 法を用いて数値計算する方法を説明する.

4.2.5.1 斉次 Neumann 境界条件の場合

1 段目

$$\frac{u^{n+\frac{1}{2}}_{i,j} - u^n}{\left(\frac{\Delta t}{2}\right)}$$

$$= d_u \left(\frac{u^{n+\frac{1}{2}}_{i+1,j} - 2u^{n+\frac{1}{2}}_{i,j} + u^{n+\frac{1}{2}}_{i-1,j}}{\Delta x^2} \right) + d_u \left(\frac{u^n_{i,j+1} - 2u^n_{i,j} + u^n_{i,j-1}}{\Delta y^2} \right) + f\left(u^n_{i,j}, v^n_{i,j}\right).$$

2 段目

$$\frac{u^{n+1}_{i,j} - u^{n+\frac{1}{2}}_{i,j}}{\left(\frac{\Delta t}{2}\right)}$$

$$= d_u \left(\frac{u^{n+\frac{1}{2}}_{i+1,j} - 2u^{n+\frac{1}{2}}_{i,j} + u^{n+\frac{1}{2}}_{i-1,j}}{\Delta x^2} \right) + d_u \left(\frac{u^{n+1}_{i,j+1} - 2u^{n+1}_{i,j} + u^{n+1}_{i,j-1}}{\Delta y^2} \right) + f\left(u^{n+\frac{1}{2}}_{i,j}, v^{n+\frac{1}{2}}_{i,j}\right).$$

1 段目に関して

$$r_x = \frac{1}{\Delta x^2} \times \left(\frac{\Delta t}{2}\right), \quad r_y = \frac{1}{\Delta y^2} \times \left(\frac{\Delta t}{2}\right)$$

とおくと

$$-d_u r_x u^{n+\frac{1}{2}}_{i+1,j} + (1 + 2d_u r_x) u^{n+\frac{1}{2}}_{i,j} - d_u r_x u^{n+\frac{1}{2}}_{i-1,j}$$

$$= u^n_{i,j} + d_u r_y \left(u^n_{i,j+1} - 2u^n_{i,j} + u^n_{i,j-1}\right) + \frac{\Delta t}{2} f\left(u^n_{i,j}, v^n_{i,j}\right)$$

と書ける．ここで

$$A=\begin{pmatrix}\frac{1}{2}(1+2d_ur_x) & -d_ur_x & 0 & 0 & 0 & 0\\ -d_ur_x & 1+2d_ur_x & -d_ur_x & 0 & 0 & 0\\ 0 & -d_ur_x & 1+2d_ur_x & -d_ur_x & 0 & 0\\ 0 & \ddots & \ddots & \ddots & \ddots & 0\\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots\\ 0 & 0 & 0 & -d_ur_x & 1+2d_ur_x & -d_ur_x\\ 0 & 0 & 0 & 0 & -d_ur_x & \frac{1}{2}(1+2d_ur_x)\end{pmatrix}$$

とすると，この式は各 j に対して

$$A\boldsymbol{u}_j^{n+\frac{1}{2}}=\boldsymbol{b}_j,\quad 0\le j\le Ny \tag{4.40}$$

という連立一次方程式を解く問題に帰着される．ただし，$\boldsymbol{b}_j$ は斉次 Neumann 境界条件の離散化 (4.37) より

$$\begin{cases}\boldsymbol{b}_0=\frac{1}{2}\left(u_{i,0}^n+d_ur_y(2u_{i,1}^n-2u_{i,0}^n)+\frac{\Delta t}{2}f(u_{i,0}^n,v_{i,0}^n)\right),\quad (0\le i\le N_x),\\ \boldsymbol{b}_j=\left(u_{i,j}^n+d_ur_y(u_{i,j+1}^n-2u_{i,j}^n+u_{i,j-1}^n)+\frac{\Delta t}{2}f(u_{i,j}^n,v_{i,j}^n)\right),\quad (0\le i\le N_x),\\ \boldsymbol{b}_{N_y}=\frac{1}{2}\left(u_{i,N_y}^n+d_ur_y(2u_{i,N_y-1}^n-2u_{i,N_y}^n)+\frac{\Delta t}{2}f(u_{i,N_y}^n,v_{i,N_y}^n)\right),\quad (0\le i\le N_x)\end{cases} \tag{4.41}$$

となる．各 j に対して，(4.40) を解くと

$$u_{i,j}^{n+\frac{1}{2}},\quad (0\le i\le Nx,\quad 0\le j\le Ny)$$

が求まる．
2 段目に関しては

$$\begin{aligned}&-d_ur_yu_{i,j+1}^{n+1}+(1+2d_ur_)u_{i,j}^{n+1}-d_ur_yu_{i,j-1}^{n+1}\\ &\qquad=d_ur_x\left(u_{i+1,j}^{n+\frac{1}{2}}-2u_{i,j}^{n+\frac{1}{2}}+u_{i-1,j}^{n+\frac{1}{2}}\right)+\frac{\Delta t}{2}f\left(u_{i,j}^{n+\frac{1}{2}},v_{i,j}^{n+\frac{1}{2}}\right)\end{aligned}$$

より

$$B=\begin{pmatrix}\frac{1}{2}(1+2d_ur_y) & -d_ur_y & 0 & 0 & 0 & 0\\ -d_ur_y & 1+2d_ur_y & -d_ur_y & 0 & 0 & 0\\ 0 & -d_ur_y & 1+2d_ur_y & -d_ur_y & 0 & 0\\ 0 & \ddots & \ddots & \ddots & \ddots & 0\\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots\\ 0 & 0 & 0 & -d_ur_y & 1+2d_ur_y & -d_ur_y\\ 0 & 0 & 0 & 0 & -d_ur_y & \frac{1}{2}(1+2d_ur_y)\end{pmatrix}$$

とすると，この式は各 i に対して

$$B\boldsymbol{u}_i^{n+1}=\boldsymbol{b}_i,\quad (0\le i\le Nx) \tag{4.42}$$

という連立一次方程式を解く問題に帰着される．ただし，$\boldsymbol{b}_i$ は斉次 Neumann 境界条件の離散化 (4.37)

より

$$\begin{cases} \boldsymbol{b}_0 = \dfrac{1}{2}\left(u_{0,j}^{n+\frac{1}{2}} + d_u r_x(2u_{1,j}^{n+\frac{1}{2}} - 2u_{0,j}^{n+\frac{1}{2}}) + \dfrac{\Delta t}{2} f(u_{0,j}^{n+\frac{1}{2}}, v_{0,j}^{n+\frac{1}{2}})\right), \quad (0 \le j \le N_y), \\ \boldsymbol{b}_i = \left(u_{i,j}^{n+\frac{1}{2}} + d_u r_x(u_{i+1,j}^{n+\frac{1}{2}} - 2u_{i,j}^{n+\frac{1}{2}} + u_{i-1,j}^{n+\frac{1}{2}}) + \dfrac{\Delta t}{2} f(u_{i,j}^{n+\frac{1}{2}}, v_{i,j}^{n+\frac{1}{2}})\right), \quad (0 \le j \le N_y), \\ \boldsymbol{b}_{N_x} = \dfrac{1}{2}\left(u_{N_x,j}^{n+\frac{1}{2}} + d_u r_x(2u_{N_x-1,j}^{n+\frac{1}{2}} - 2u_{N_x,j}^{n+\frac{1}{2}}) + \dfrac{\Delta t}{2} f(u_{N_x,j}^{n+\frac{1}{2}}, v_{N_x,j}^{n+\frac{1}{2}})\right), \quad (0 \le j \le N_y) \end{cases} \tag{4.43}$$

となる．各 i に対して，(4.42) を解くと

$$u_{i,j}^{n+1}, \quad (0 \le i \le Nx, \quad 0 \le j \le Ny)$$

が求まる．

4.2.5.2 周期境界条件の場合

1段目に関しては

$$A = \begin{pmatrix} 1+2d_u r_x & -d_u r_x & 0 & 0 & 0 & -d_u r_x \\ -d_u r_x & 1+2d_u r_x & -d_u r_x & 0 & 0 & 0 \\ 0 & -d_u r_x & 1+2d_u r_x & -d_u r_x & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & -d_u r_x & 1+2d_u r_x & -d_u r_x \\ -d_u r_x & 0 & 0 & 0 & -d_u r_x & 1+2d_u r_x \end{pmatrix}$$

となる．ただし A は $(N_x-1)\times(N_x-1)$ の正方行列である．この式は各 j に対して

$$A\boldsymbol{u}_j^{n+\frac{1}{2}} = \boldsymbol{b}_j, \quad 0 \le j \le Ny-1 \tag{4.44}$$

という連立一次方程式を解く問題に帰着される．ただし，$\boldsymbol{b}_j$ は (4.39) より

$$\begin{cases} \boldsymbol{b}_0 = \left(u_{i,0}^n + d_u r_y(u_{i,1}^n - 2u_{i,0}^n + u_{i,N_y-1}^n) + \dfrac{\Delta t}{2} f(u_{i,0}^n, v_{i,0}^n)\right), \\ \boldsymbol{b}_j = \left(u_{i,j}^n + d_u r_y(u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n) + \dfrac{\Delta t}{2} f(u_{i,j}^n, v_{i,j}^n)\right), \\ \boldsymbol{b}_{N_y-1} = \left(u_{i,N_y-1}^n + d_u r_y(u_{i,N_y-2}^n - 2u_{i,N_y-1}^n + u_{i,0}^n) + \dfrac{\Delta t}{2} f(u_{i,N_y-1}^n, v_{i,N_y-1}^n)\right) \end{cases} \tag{4.45}$$

となる．ただし，$0 \le j \le N_x - 1$ である．(4.45) を解くと

$$u_{i,j}^{n+\frac{1}{2}}, \quad (0 \le i \le Nx-1, \quad 0 \le j \le Ny-1)$$

が求まる．

2段目に関しては

$$B = \begin{pmatrix} 1+2d_u r_y & -d_u r_y & 0 & 0 & 0 & -d_u r_y \\ -d_u r_y & 1+2d_u r_y & -d_u r_y & 0 & 0 & 0 \\ 0 & -d_u r_y & 1+2d_u r_y & -d_u r_y & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & -d_u r_y & 1+2d_u r_y & -d_u r_y \\ -d_u r_y & 0 & 0 & 0 & -d_u r_y & 1+2d_u r_y \end{pmatrix}$$

とする．ただし B は $(N_y-1)\times(N_y-1)$ の正方行列である．この式は各 i に対して

$$B\boldsymbol{u}_i^{n+1}=\boldsymbol{b}_i,\quad (0\le i\le Nx-1) \tag{4.46}$$

という連立一次方程式を解く問題に帰着される．ただし，$\boldsymbol{b}_i$ は (4.39) より

$$\begin{cases}\boldsymbol{b}_0=\left(u_{0,j}^{n+\frac{1}{2}}+d_u r_x(u_{1,j}^{n+\frac{1}{2}}-2u_{0,j}^{n+\frac{1}{2}}+u_{N_x-1,j}^{n+\frac{1}{2}})+\dfrac{\Delta t}{2}f(u_{0,j}^{n+\frac{1}{2}},v_{0,j}^{n+\frac{1}{2}})\right),\\ \boldsymbol{b}_i=\left(u_{i,j}^{n+\frac{1}{2}}+d_u r_x(u_{i+1,j}^{n+\frac{1}{2}}-2u_{i,j}^{n+\frac{1}{2}}+u_{i-1,j}^{n+\frac{1}{2}})+\dfrac{\Delta t}{2}f(u_{i,j}^{n+\frac{1}{2}},v_{i,j}^{n+\frac{1}{2}})\right),\\ \boldsymbol{b}_{N_x-1}=\left(u_{N_x-1,j}^{n+\frac{1}{2}}+d_u r_x(u_{N_x-2,j}^{n+\frac{1}{2}}-2u_{N_x-1,j}^{n+\frac{1}{2}}+u_{0,j}^{n+\frac{1}{2}})+\dfrac{\Delta t}{2}f(u_{N_x-1,j}^{n+\frac{1}{2}},v_{N_x-1,j}^{n+\frac{1}{2}})\right)\end{cases} \tag{4.47}$$

となる．ただし，$0\le j\le N_y-1$ である．(4.47) を解くと

$$u_{i,j}^{n+1},\quad (0\le i\le Nx-1,\quad 0\le j\le Ny-1)$$

が求まる．

反応拡散系の数値計算上の注意

拡散項の数値計算法に陰解法を使っているからといって，Δt を大きく取れるわけではない．反応項の数値計算は Euler 法で計算しているので Δt を大きく取ると Euler 法によって発散してしまう !!

4.2.5.3　問題

問題１．拡張した FitzHugh-Nagumo 方程式 [25]

$$\begin{cases}\dfrac{\partial u}{\partial t}=d_u\Delta u+\dfrac{1}{\varepsilon}\left(u(1-u)(u-a)-v\right),\\ \qquad\qquad t>0,\ x\in(0,L)\times(0,L)\equiv\Omega\\ \dfrac{\partial v}{\partial t}=d_v\Delta v+u-\gamma v.\end{cases} \tag{4.48}$$

ただし，$0<a<1/2$.

$\gamma=1.0,\varepsilon=0.001,a=0.125,d_u=1.0,d_v=0.1,L=20.0$ とおいて数値計算を行い，スパイラル波を見つけなさい．そして d_v を大きくとると (例えば，$d_v=10.0$) どうなるか数値実験してみなさい．ただし，境界条件は斉次 Neumann 境界条件

$$\frac{\partial u}{\partial x}(t,x,y)=\frac{\partial u}{\partial y}(t,x,y)=0,\frac{\partial v}{\partial x}(t,x,y)=\frac{\partial v}{\partial y}(t,x,y)=0,\quad t>0,x\in\partial\Omega$$

と周期境界条件の両方の場合とする．

第5章　可視化

数値計算結果を可視化することは数値シミュレーションを行う上でも，方程式に対するイメージを掴むためにも非常に重要です．結果をすばやく正確に可視化できれば数値シミュレーションの効率もグッと上昇するでしょう！

5.1　数値データの可視化

(1) できあいのプログラムを用いる

- GNUPLOT → フリーソフト，最近ではグラデーションも表示できる．
- Mathematica → 有料でしかも高価.
- AVS → 有料（欲しいけど買えない！）.

長所：定型的なグラフならば美しくかつ楽に出せる．
短所：シミュレーションを走らせながらモニターするのには向かない[1]．楽な反面，かゆいところに手がとどかない.

(2)Graphic Library を用いる

- OpenGL, MesaGL 系（[18],[19]）+ glut [20].
- GLSC 系 (手作り，広島大学の小林亮先生らが作ったもの)([21, 22]).

長所：シミュレーションプログラムを走らせながらモニターするのに便利[2]．細かいところまで自分の思うようにグラフが出せる．
短所：プログラムを書く手間がかかる．

以下の解説には **GUNPLOT ver.4.0** 以上と **ImageMagick** がインストールされていなければなりません．入っていない場合は前もってインストールしてください．cygwin はすべて入っています．Linux だと GNUPLOT はパッケージインストールする必要があります.

5.2　必要なソフトウエアのインストール

フルパッケージの cygwin をインストールしてもインストールされないソフトウエアをインストールする手順を解説します．以下のソフトウエアは Linux にもインストールする必要があります.

[1] GNUPLOT を使えばシミュレーション結果をモニターすることがお手軽にできるが，２次元の描画は時間コストが高い．１次元グラフなら GNUPLOT を用いるのが一番かもしれない
[2] 表示する時間コストが少ないという点で有利であるという意味

5.2.1　mpeg_encode のインストール

cygwin 上でのインストールは次にように行う.

```
$ tar -xvzf mpeg_encode_cygwin_src.tar.gz
$ cd mpeg_encode/jpeg
$ make
$ cd ../
$ make
warning が一杯出るが，mpeg_encode.exe はできている.
$ cp  mpeg_encode.exe /usr/local/bin/.
としてインストール終わり.
```

5.2.2　mpeg_play のインストール

cygwin 上でのインストールは次のように行う.

```
$ tar -xvzf mpeg_play_cygwin_src.tar.gz
$ cd mpeg_play
$ make
warning が一杯出るが，mpeg_play.exe はできている.
$ cp  mpeg_play.exe /usr/local/bin/.
としてインストール終わり.
```

5.3　必要になるかもしれないソフトウエアのインストール

5.3.1　OpenGL のインストール

cygwin に OpenGL がインストールされていない場合や glut がインストールされていない場合は自分でインストールしてみよう！

```
$ tar -xvzf MesaLib-7.2.tar.gz
$ tar -xvzf MesaGLUT-7.2.tar.gz
$ tar -xvzf MesaDemos-7.2.tar.gz
$ cd Mesa-7.2
$ ./configure
$ make
最後に Err が出て終わるが，OpenGL のコンパイルは出来ている.
少なくとも私が使った範囲では動いてくれた.
$ make install
としてインストール終わり.
最後に Err が出たら確認のため
$ cd lib
$ cp * /usr/local/lib/.
とすればインストールが出来ているはず.
```

5.3.2 GLSC のインストール

私は主に GLSC は鳥瞰図を論文に載せたいときに使っている[3].
cygwin 上でのインストールは次のように行う.

```
$ tar -xvzf glsc-3.5-patch.tar.gz
```
このソースファイルは正規の GLSC から鳥瞰図を白塗りに変更したものである.
```
$ cd glsc-3.5
$ make
```
最後に Err が出て終わるが，GLSC のコンパイルは出来ている.
```
$ make install
```
でインストール完了.
確認したい場合は
```
$ cd test
$ ./ctestd.exe
```
としてみよう．GLSC が実行されるはずです.

5.4 GNUPLOT による数値計算結果の可視化 [15]

我々の業界では，数値計算結果を分かりやすく表現するも重要である．また，解の遷移過程や漸近挙動のパラメータ依存性をすばやく調べるためには数値計算と同時に可視化することが望ましい．そのためには数値計算プログラム以外のプログラムを組むことが必要となり，初心者には困難な作業と必要なる．そこで GNUPLOT を使うことによって可視化プログラム作成コストを少なくする．数値計算中の途中結果データを GNUPLOT で表示できるように出力することで，ある程度計算過程での数値計算結果を調べることができる．GNUPLOT の詳しい使い方は [15] を見て下さい．例えば，数値計算データが $(t, u(t))$ の順番にならんでいたら，次のようにすればよい：

```
% gnuplot
gnuplot> plot 'sol_ODE.dat' using 1:2 with lines
```

数値データの形式が $(t, x, u(t, x))$ のとき

```
% gnuplot
gnuplot> splot 'sol_1DRD.dat' using 1:2:3 with lines
```

5.4.1 プリンターへの出力

プリンターへの出力は例えば次のようにする.

[3]計算しながら結果を表示するためなら GNUPLOT で十分である.

```
% gnuplot
    # gnuplot の起動
gnuplot> plot 'sol_ODE.dat' using 1:2 with lines
    # この時点まではデータの可視化と同じ.
gnuplot> set term postscript
    # この時点でウインドウが消える
gnuplot> set output 'fig01.ps'
    # fig01.ps というファイルを開く
gnuplot> replot
    # plot 'sol_ODE.dat' using 1:2 with lines と同じ
    # sol_ODE.dat に対するグラフの postscript データが fig01.ps に出力される
gnuplot> quit
    # gnuplot の終了
% lpr fig01.ps
    # fig01.ps をプリンターに出力する
```

上記の作業を繰り返し行う場合には，次のようなスクリプトファイル (g_plot) を作りそれを繰り返し使う方が効率がよい.

```
plot 'sol_ODE.dat' using 1:2 with lines
pause -1 "Hit Return  key"
    # 正しくデータがグラフ表示されているか確認する
    # リターンキーを押せば次に進む
set nokey
    # グラフの中のタイトルを消去する
    # タイトルが不要の場合は書いおけばよい
set term postscript
set output 'fig01.ps'
replot
```

そして

```
% gnuplot g_plot
```

とすれば，fig01.ps というファイルができて gnuplot が自動的に終了するので，満足いくグラフならプリンターに出力すればよい．また，fig01.ps の内容を確認したいなら以下のようにすればよい.

```
% gs fig01.ps
```

5.4.2　GNUPLOT を使ってパラパラ動画を作ろう

数値データの構造が

時間　空間　u の値　v の値　$\cdots$

となっている数値データ (ファイル名：sample_data.dat) に対して，各時間ステップごとの数値データファイルに分割する perl スクリプト（ファイル名：datacut）を作る：

```
#! /usr/bin/perl

# データを分割するディレクトリ
$data_dir = 'cut';

$i = 0;
open(GA, ">$data_dir/$i");
open(HA, ">./plot1d");

print HA "set yrange[0:20]\n";
print HA "plot './$data_dir/$i' using 1:2 with lines,\\
'./$data_dir/$i' using 1:3 with lines \n";

$j = 0;
while(<>) {
    $a = $&.$' if /\d/;
    @b = split(/\s+/, $a);
    if($b[0] != $i){
        $i = $b[0];
        $j += 1;
        print HA "plot './$data_dir/$j' using 1:2 with lines,\\
         './$data_dir/$j' using 1:3 with lines \n";
        close(GA);
        open(GA, ">$data_dir/$j");
        print GA " $b[1] $b[2] $b[3]\n";
    }
    else {
        print GA " $b[1] $b[2] $b[3]\n";
    }
    $i = $b[0];
}
print HA "pause -1 'Hit return key !!' \n";
close(GA); close(HA);
```

このスクリプトを用いて次の操作を行うことでパラパラ動画を作りことができます[4].

```
% ./datacut < sample_data.dat
    # ./cut の中に分割された数値データファイルが作られる.
    # ./ に plot1d という gnuplot のスクリプトファイルが作られる.
% gnuplot plot1d
    # 動画のように見えますよね !?
```

5.4.3 GNUPLOT を使って JPEG 形式の画像ファイルを作る

GNUPLOT のバージョンが 4.0 以上なら GNUPLOT で表示された画像を JPEG フォーマットのファイルとして保存することができる.

[4]事前に cut というディレクトリを作っておかなければなりません.

```
% gnuplot
    # gnuplot の起動
gnuplot> plot 'sol_ODE.dat' using 1:2 with lines lw 3
    # この時点まではデータの可視化と同じ.
    # lw は線の太さを変更するオプション．デフォルトは１.
gnuplot> set term jpeg
    # この時点でウインドウが消える
gnuplot> set output 'fig01.jpg'
    # fig01.jpg というファイルを開く
gnuplot> replot
    # plot 'sol_ODE.dat' using 1:2 with lines と同じ
    # sol_ODE.dat に対する jpeg フォーマットの画像が fig01.jpg に出力される
gnuplot> quit
    # gnuplot の終了
% display fig01.jpg
    # fig01.jpg が表示される.
```

JPEG ファイルを表示して見るとわかるが，JPEG 形式の画像ファイルは劣化する．線画ではこの劣化が致命傷となる場合があるので，JPEG 形式の画像ファイルは 2 次元以上の数値データに対するグラデーション表示に使うべきである．どうしても線画を JPEG 形式で保存したい場合は線幅を太くするとよい．上記の例では lw オプションで線を太くしている．表示する線を太くするには次のように使えばよい：

```
% gnuplot
    # gnuplot の起動
gnuplot> plot 'sol_ODE.dat' using 1:2 with lines lw 3
    # lw オプションは線の太さを変更するデフォルトは 1 である.
```

5.4.4　GNUPLOT を使って GIF 形式の画像ファイルを作る

前節と同様にバージョン 4.0 以上の GNUPLOT で表示された画像を gif 形式のファイルとして保存することができる．ただし，Fedora Core5 の GNUPLOT ver4.0 では GNUPLOT をインストールし直す必要があります.

```
% gnuplot
    # gnuplot の起動
gnuplot> plot 'sol_ODE.dat' using 1:2 with lines
    # この時点まではデータの可視化と同じ.
    # GIF 形式にする場合は
gnuplot> set term gif
    # この時点でウインドウが消える
gnuplot> set output 'fig01.gif'
    # fig01.gif というファイルを開く
gnuplot> replot
    # plot 'sol_ODE.dat' using 1:2 with lines と同じ
    # sol_ODE.dat に対する gif フォーマットの画像が fig01.gif に出力される
gnuplot> quit
    # gnuplot の終了
% display fig01.gif
    # fig01.gif が表示される.
```

線画を JPEG 形式で圧縮するとどうしても解像度が落ちるので GIF 形式で保存する方がよい．後から PPT に貼り付けたり動画化するのにも便利です.

5.4.5 GNUPLOT を使って PNG 形式の画像ファイルを作る

ライセンスの関係で GIF 形式で保存できない場合は，PNG 形式で保存するとよい．この場合も線画が綺麗に保存できる.

```
% gnuplot
    # gnuplot の起動
gnuplot> plot 'sol_ODE.dat' using 1:2 with lines
    # この時点まではデータの可視化と同じ.
# PNG 形式にする場合は
gnuplot> set term png
    # この時点でウインドウが消える
gnuplot> set output 'fig01.png'
    # fig01.png というファイルを開く
gnuplot> replot
    # plot 'sol_ODE.dat' using 1:2 with lines と同じ
    # sol_ODE.dat に対する PNG フォーマットの画像が fig01.png に出力される
gnuplot> quit
    # gnuplot の終了
% display fig01.png
    # fig01.png が表示される.
```

5.5　GNUPLOT と ImageMagick による動画の作成

5.5.1　アニメーション GIF の作成

この節では，5.4.4,5.4.5 で作った GIF 画像や PNG 画像の時系列画像ファイルを用いてアニメーション GIF ファイルの作成を行う．第 5.4.2 節で用いた perl スクリプトを改良して次のようなスクリプト (data2image) を作る：

```
#! /usr/bin/perl

# データを分割するディレクトリ
$data_dir = 'cut';

# 画像を保存するディレクトリ
$image_dir = 'image';

# -> png とすれば PNG 形式の画像となる.
$format = 'gif';

$i = 0;

open(GA, ">$data_dir/$i");
open(HA, ">./plot_$format");
print HA "set yrange[0:20]\n";
print HA "set term $format\n";
print HA "set output './$image_dir/fig0000.$format' \n";
print HA "plot './$data_dir/$i' using 1:2 with lines,\\
  './$data_dir/$i' using 1:3 with lines \n";
$j = 0;
while(<>) {
  $a = $&.$' if /\d/;
  @b = split(/\s+/, $a);
  if($b[0] != $i){
    $i = $b[0];
    $j += 1;
    if($j <= 9){
      print HA "set output './$image_dir/fig000$j.$format'\n";
    }
    elsif($j >= 10 && $j <= 99){
      print HA "set output './$image_dir/fig00$j.$format'\n";
    }
    else{
      print HA "set output './$image_dir/fig0$j.$format'\n";
    }
    print HA "plot './$data_dir/$j' using 1:2 with lines,\\
      './$data_dir/$j' using 1:3 with lines \n";
    close(GA);
    open(GA, ">$data_dir/$j");
    print GA " $b[1] $b[2] $b[3]\n";
  }
  else {
    print GA " $b[1] $b[2] $b[3]\n";
  }
  $i = $b[0];
}
close(GA);
close(HA);
```

このスクリプトを用いて[5]

```
% ./data2image < sample_data.dat
    # ./cut の中に分割された数値データファイルが作られる.
    # ./ に plot_gif という gnuplot のスクリプトファイルが作られる.
% gnuplot plot_gif
    # ./image の中に fig0000.gif〜fig0xxx.gif の GIF ファイルができる
```

もしも PNG 形式の画像を作るのだったら data2image ファイルにおいて

```
$format = 'gif';
を
$format = 'png';
と書き換えればよい.
```

以下のように実行すると PNG ファイルができる：

```
% gnuplot plot_png
    # ./image の中に fig0000.png〜fig0xxx.png の PNG ファイルができる
```

ImageMagick がインストールされているならば，次のようにすればアニメーション GIF を作ることができます：

```
% convert -delay 20 ./image/fig*.gif animation.gif
    # -delay は再生速度を調整するオプションです.
```

PNG ファイルならば

```
% convert -delay 20 ./image/fig*.png animation.gif
    # -delay は再生速度を調整するオプションです.
```

とすることでアニメーション GIF を作ることができます.

できた animation.gif は animate を使って再生することができます[6]：

```
% animate animation.gif
```

もし，animate が見つからない場合は ImageMagick をインストールしてください.

5.5.2 MPEG 動画の作成

前節のスクリプトファイル (data2image) を次のように書き換える：

```
#! /usr/bin/perl

# データを分割するディレクトリ
$data_dir = 'cut';

# 画像を保存するディレクトリ
$image_dir = 'image';
```

[5]事前に cut と image というディレクトリを作っておかなければなりません.

[6]Windows で再生する場合は再生ソフトをインストールする必要がある．例えば QuickTime があれば再生できます.

```
# 修正したのはこの部分
$format = 'jpeg';

$i = 0;

open(GA, ">$data_dir/$i");
open(HA, ">./plot_format");
print HA "set yrange[0:20]\n";
print HA "set term $format\n";
print HA "set output './$image_dir/fig0000.$format' \n";
print HA "plot './$data_dir/$i' using 1:2 with lines,\\
  './$data_dir/$i' using 1:3 with lines \n";
$j = 0;
while(<>) {
  $a = $&.$' if /\d/;
  @b = split(/\s+/, $a);
  if($b[0] != $i){
    $i = $b[0];
    $j += 1;
    if($j <= 9){
      print HA "set output './$image_dir/fig000$j.$format'\n";
    }
    elsif($j >= 10 && $j <= 99){
      print HA "set output './$image_dir/fig00$j.$format'\n";
    }
    else{
      print HA "set output './$image_dir/fig0$j.$format'\n";
    }
    print HA "plot './$data_dir/$j' using 1:2 with lines,\\
      './$data_dir/$j' using 1:3 with lines \n";
    close(GA);
    open(GA, ">$data_dir/$j");
    print GA " $b[1] $b[2] $b[3]\n";
  }
  else {
    print GA " $b[1] $b[2] $b[3]\n";
  }
  $i = $b[0];
}
close(GA);
close(HA);
```

この perl スクリプトを用いて

```
% ./data2image < sample_data.dat
    # ./cut の中に分割された数値データファイルが作られる.
    # ./ に plot_jpeg という gnuplot のスクリプトファイルが作られる.
% gnuplot plot_jpeg
    # ./image の中に fig0000.jpeg〜fig0xxx.jpeg の JPEG ファイルができる
```

5.5.3 MPEG 動画作成 その１

このうようにして作られた時系列 JPEG 画像ファイルをまとめて MPEG の動画ファイルにする方法を解説する．MPEG 化するためにパラメーターファイル (mpeg_encode_para) を適切に書くことによって，次のように実行すれば MPEG の動画ファイルが作成される：

```
% mpeg_encode mpeg_encode_para
```

ここでは，sample.mpg が作成されたとすると，その動画を再生するには

```
% mpeg_play sample.mpg
```

とするだけでよい[7]．パラメーターファイルに関しては，
“http://www.not-enough.org/abe/manual/command/mpeg_encode.html”
を参照のこと!!

5.5.3.1 パラメータファイル (mpeg_encode_para) のサンプル

```
PATTERN         IBBPBBPBBPBBPBBP
# this must be one of {YUV, PPM, PNM}
BASE_FILE_FORMAT        PPM

#INPUT_CONVERT   convert -size 320x240 * ppm:-
INPUT_CONVERT   convert -size 640x480 * ppm:-
# 動画のサイズを決定する.

GOP_SIZE        30
SLICES_PER_FRAME        1

# 出力するファイル名を指定する
OUTPUT          sample.mpeg

# 画像ファイルの存在するディレクトリーを指定する
INPUT_DIR       image

# 画像ファイルの記述始まり
INPUT
# 拡張子が jpg なら   Image*.jpg と書き換えること
Image*.jpeg      [0000-0100]
# Image0001.jpeg...Image0100.jpeg という画像ファイルがエンコードされる
# 画像ファイルの記述始まり
END_INPUT

# this must be one of {FULL, HALF}
PIXEL           HALF
RANGE           10
# this must be one of {EXHAUSTIVE, SUBSAMPLE, LOGARITHMIC}
PSEARCH_ALG     LOGARITHMIC
# this must be one of {SIMPLE, CROSS2, EXHAUSTIVE}
BSEARCH_ALG     CROSS2

IQSCALE         8
PQSCALE         10
```

[7] ここで作成した sample.mpg は Windows 上でも再生できるので，PowerPoint 等に貼り付けて講演するときに有効である．

```
BQSCALE         25

# this must be ORIGINAL or DECODED
REFERENCE_FRAME ORIGINAL
FRAME_RATE 30
```

5.5.4　MPEG 動画作成 その 2

ImageMagick を使って MPEG 動画を作ることも可能です．このときは事前に mpeg2vidcodec をインストールする必要があります[8]

cygwin 上では

```
% rpm  -ivh mpeg2vidcodec-1.2-5.src.rpm
```

Fedora8 上では

```
% rpm  -ivh mpeg2vidcodec-1.2-1.i386.rpm
```

でインストールできます．
インストールができたら次のように入力してください．MPEG ファイルを作ることができます．

```
% convert *.jpeg test.mpg
```

この方法で MPEG ファイルを作るときは，静止画が BMP でも PPM でも何でも大丈夫です．

この場合も簡単に動画を作ることができますが，cygwin 上で実行したところ，MPEG ファイルの画質の劣化がはっきりと見てとれました．

5.6　GNUPLOT を C 言語から呼び出す [16, 17]

Fortran から呼び出して使う場合，いちいち GNUPLOT を終了させなければならないので画面がチラつく．ちらつきが嫌な場合は，メインルーチン部分も C 言語で書いた方がよい．具体的にはパイプ機能を用いて行います．次の例は [16] に記載されていたサンプルを改良したものです (sample_gnuplot01.c)．

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

int main(int argc, char **argv)
{
    FILE *gpid;
    int i;
    double x;

/* gnuplot を開く */
    gpid = popen("gnuplot","w");
```

[8] mpeg2vidcodec は rmp 形式で配布されているのでインストールは簡単です．mpeg_encode や mpeg_play をインストールするよりは簡単なので，mpeg_encod をインストールすることができない人にはこちらがお勧めです．

```
/* gnuplot が見つからない場合のエラー処理  */
    if(gpid == NULL){
        printf("I can't find gnuplot.\n");
        exit(1);
}
/* gnuplot でグラフを書く */
    for(i=1; i < 100; i++){
        fprintf(gpid, "set xrange [0:4]\n");
        x = 0.03*i;
        fprintf(gpid, "plot sin(x - %f*pi)\n", x);
    }

/* 最後の画面を１０秒静止する  */
/* C 言語から呼び出すときは pause -1 が効かないので注意すること */
    fprintf(gpid, "pause 10\n");

    pclose(gpid);
}
```

サンプルプログラムのコンパイル方法と実行方法 (run_sample.sh を参照してください)

```
$ gcc -O3 -Wall sample_gnuplot01.c -o a.out -lm
$ ./a.out # 実行
```

上記のプログラムを改造すれば，数値計算を実行しながら GNUPLOT で計算結果を可視化することができる．詳しくは [16] を参考にしてください．以下のサンプルプログラムは熱方程式の近似解を陽解法により求めながら，GNUPLOT を用いて数値解を表示するプログラムである (sample_gnuplot02.c)：

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

#define dt 0.0005
#define N 10
#define T 1000

int main()
{
  double u[N+1], new_u[N+1];
  int i, j;
  double dx = 1. / (float)N;
  double r = dt / (dx * dx);
  FILE *gpid;

  /* gnuplot を開く */
  if((gpid =  popen("gnuplot -persist", "w")) == NULL) {
```

```
    printf("I can't find gnuplot.\n");
    exit(1);
  }

  /* y 方向の描画領域を固定 */
  fprintf(gpid, "set yrange[-4:4]\n");

  for(i = 0; i <= N; i++){
    u[i] = 4 * cos(M_PI * i * dx);
  }

  for(j = 0; j < T; j++){
    /* 熱方程式を陽解法で解く*/
    for(i = 1; i < N; i++){
      new_u[i] = r * u[i - 1] + (1. - 2. * r) * u[i] + r * u[i + 1];
    }
    new_u[0] = 2. * r * u[1] + (1. - 2. * r) * u[0];
    new_u[N] = 2. * r * u[N - 1] + (1. - 2. * r) * u[N];

    for(i = 0; i <= N; i++){
      u[i] = new_u[i];
    }

    /* gnuplot でグラフを描く */
    fprintf(gpid, "plot '-' w l t '%d'\n", j);
    for(i = 0; i <= N; i++){
      fprintf(gpid, "%lf %lf\n", i * dx, u[i]);
    }
    fprintf(gpid, "e\n");
    fflush(gpid);
  }

  /* 最後の画面を 10 秒静止する */
  /* C 言語から呼び出すときは pause -1 が効かないので注意 */
  fprintf(gpid, "pause 10\n");

  /* gnuplot を閉じる */
  pclose(gpid);

  return 0;
}
```

サンプルプログラムのコンパイル方法と実行方法 (run_sample.sh を参照してください)

```
$ gcc -O3 -Wall sample_gnuplot01.c -o a.out -lm
$ ./a.out # 実行
```

5.7 OpenGL によるオフスクリーンレンダリング

大事なおまじないは次の行である.

```
...
#include "GL/osmesa.h"
...
int main()
{
  OSMesaContext ctx;
  void *buffer;
  ...
  /* Create an RGBA-mode context */
  ctx = OSMesaCreateContext( GL_RGBA, NULL );
  /* Allocate the image buffer */
  buffer = malloc( WIDTH * HEIGHT * 4 );
  /* Bind the buffer to the context and make it current */
  OSMesaMakeCurrent( ctx, buffer, GL_UNSIGNED_BYTE, WIDTH, HEIGHT );
  ...(#1)
  OSMesaDestroyContext(ctx);
}
```

オフスクリーンレンダリングはバックグランドで数値計算を行いながら静止画の時系列画像を作成するのに便利です[9]. 従って, 画像をファイルにする必要があります. その関数は次のように記述します. ここでは PPM ファイル形式で出力します.

```
save_file(FILE *f, void *buffer){
  if (f) {
    int i, x, y;
    GLubyte *ptr;
    ptr = buffer;
    fprintf(f,"P6\n");
    fprintf(f,"# ppm-file created\n");
    fprintf(f,"%i %i\n", Width,Height);
    fprintf(f,"255\n");
    for (y=Height-1; y>=0; y--) {
      for (x=0; x<Width; x++) {
        i = (y*Width + x) * 4;
        fputc(ptr[i], f);   /* write red */
        fputc(ptr[i+1], f); /* write green */
        fputc(ptr[i+2], f); /* write blue */
      }
    }
    fclose(f);
  }
```

[9] 大規模計算では数値データだけを出力して, あとで可視化することをお勧めします.

```
  printf("all done\n");
}
```

PPM ファイルはファイルサイズが大きくなるので ImageMagick を用いて JPEG ファイルに変換します．次のような関数を用意します．

```
void change_image_format(char *in, char *out)
{
   char string[300];
   sprintf(string, "convert %s %s", in, out);
   system(string);
}
```

そして main 関数の中で (#1) の部分のところに次のように書き込みます．

```
  ...
  /* PPM ファイル名の決定 */
  sprintf(ppmfile, "%s/%s%04d.ppm",image_dir,image_file,fn);
  /* JPEG ファイル名の決定 */
  sprintf(jpegfile,"%s/%s%04d.jpg",image_dir,image_file,fn);
  if ((fp = fopen(ppmfile, "w")) == NULL){
    perror(ppmfile);
    exit(1);
  }
  /* PPM ファイルに保存 */
  save_file(fp, buffer);
  /* PPM file -> JPEG file */
  change_image_format(ppmfile,jpegfile);
  /* PPM ファイルを消去 */
  unlink(ppmfile);
  /* image buffer の開放*/
  free(buffer);
```

詳しくはサンプルプログラム (5.8.2.1) を参照してください．ここで作られた時系列 JPEG 画像データは 5.5.2 で解説した方法で MPEG 動画に変換することができます．

5.8　可視化の参考プログラム

この節に掲載しているサンプルプログラムの実行方法は CD-ROM の Sample_Program 中の run_script, run_glse を参考にしてください．

5.8.1　GLSC による鳥瞰図の参考プログラム

このプログラム (GLSC_sample01.c) は上山大信氏（明治大学）によるプログラムに手を加えたもの．論文に掲載する鳥瞰図を作るために使っている．

```
#include "/usr/local/include/glsc.h"
#include <stdio.h>
#define Imax    (251) /* x 軸の分点数 */
#define Jmax    (100) /* y 軸の分点数 */
#define Xleft   (0.0)
#define Xright  (1.0)
#define Ybottom (0.0)  /* グラフ化時の最小値の調節 */
#define Ytop    (30.0) /* グラフ化時の最大値の調節 */
#define Jstep   (1)
#define Istep   (Imax)
#define X (100.0)
#define Y (200.0)
#define Xmar (10.0)
#define Ymar (10.0)
#define Xwid (80.0)
#define Ywid (180.0)
void main(int argc, char **argv)
{
    int i, j;
    int imax = Imax;
    int jmax = Jmax;
    int istep = Istep;
    int jstep = Jstep;
    FILE *fp = stdin;
    double *u, *v;

    fp = fopen(argv[1], "r");
    /* Memory allocation */
    u = (double *)malloc((imax)*(jmax)*sizeof(double));
    if(u == NULL)
    {
        fprintf(stderr, "Can't allocate memory.\n", argv[0]);
        exit(1);
    }
    /* Read the data */
    for(j = 0; j < jmax; j++)
    {
        for(i = 0; i < imax; i++)
        {
            fscanf(fp, "%lf", &u[i*jmax + j]);
        }
    }
    g_init("Graph", X, Y);
    g_device(G_BOTH);
    g_hidden(10.0, 15.0, 10.0, Ybottom, Ytop,
             500.0, 0.0, 80.0,
             Xmar, Ymar, Xwid, Ywid,
             u, imax, jmax, 1,
             1, istep, jstep);
   g_sleep(G_STOP);
}
```

このときの Makefile(Makefile_GLSC_sample01) は次のようになる.

```
#make
```

```
LIB = -lglscd -lX11 -lm
DIR = -I/usr/X11R6/include -I/usr/local/include \
-L/usr/X11R6/lib -L/usr/local/lib
OBJECT = GLSC_sample01.c
CC = gcc -g
all : main01
main01 : $(OBJECT)
$(CC) -o a.out $(OBJECT) $(DIR) $(LIB)
```

コンパイル方法と実行方法

```
$ make -f Makefile_GLSC_sample01
$ ./a.out sample_data_glsc01.dat
data_file_name の中は 1 列に並んだデータ
x 軸データが Imax と一致し，y 軸データが Jmax と一致している必要がある．
例えば，Sample_Program 内の sol01.dat を使うときは
Imax = 501，Jmax = 100 として make をやり直す．
$ make -f Makefile_GLSC_sample01
$ ./a.out sol01.dat
とすると，鳥瞰図が出力される．鳥瞰図上でマウスの左ボタンを押せば終了．
最後に次のコマンドを実行．
$ g_out -v Graph
これで Graph.ps という PS ファイルができる．
```

これらの作業を自動で行うためのスクリプトファイル (run_glsc) を用意しておくと便利．

```
#!/bin/sh
make -f Makefile_GLSC_sample01
./a.out sample_data_glsc01.dat
g_out -v Graph
```

5.8.2　OSMesa(OpenGL) のプログラム

OpenGL の osmesa と ImageMagick の convert を使って計算しながらグラフをファイルに落すためのプログラムの例．数値計算プログラムの中からは次のように呼び出す．

C から呼び出す場合

```
    mk_color_full(u, rr, gg, bb, mod, umax, umin, nx0, nx, ny0, ny);
    draw_(ii, Mgx, Mgy, dgx, dgy, rr, gg, bb, "u", "image");
```

次のプログラムは上記の 2 つのプログラムを用いて熱方程式の数値結果を画像にして保存するサンプルプログラム (heat2d_ADI.c) です：

```
#include <stdio.h>
#include <math.h>
void mk_color_full(double (*)[], double[], double[], double[],
int, double, double, int, int, int, int);
void draw_(int, int, int, double, double, double *, double *,
double *, char *, char *);
```

```
#define TIME 500   /* 計算回数  */
#define CUT 10    /* 何回毎に出力するか  */
#define DT 0.00001     /* Δt  */
#define NX 100    /* 要素の個数  */
#define NY 50    /* 要素の個数 */

#define LENGTH_X 1    /* 長さ  */
#define LENGTH_Y 0.5    /* 長さ  */
#define DX 1   /* 拡散係数  */
#define DY 1  /* 拡散係数  */

/*OpenGL*/
#define nx0 0
#define ny0 0
#define mod 1

void initial_u(double u[NX + 1][NY + 1],
       double dx, double dy, int nx, int ny)
{
  int i, j;
  double pi = acos(-1.);
  double x, y;

  for(i = 0; i <= nx; i++) {
    x = dx * i;
    for(j = 0; j <= ny; j++) {
      y = dy * j;
      u[i][j] = 4 * cos(3 * pi * x) * cos(2 * pi * y);
    }
  }
}

/* LU 分解のプログラム */
void lu_divide(double a[], double b[], double c[],
 double l[][2], double u[], int n)
{
  int i;

  l[0][0] = a[0];
  for(i = 1; i <= n; i++) {
    l[i][1] = b[i];
  }
  for(i = 0; i <= n - 1; i++) {
    u[i] = c[i] / l[i][0];
    l[i + 1][0] = a[i + 1] - u[i] * l[i + 1][1];
  }
}

/* LU x = b の解法*/
void lu_solve(double y[], double b[], double l[][2], double u[],
double x[], int n)
{
  int i;

  y[0] = b[0] / l[0][0];
  for(i = 1; i <= n; i++) {
    y[i] = (b[i] - l[i][1] * y[i - 1]) / l[i][0];
```

```
  }
  x[n] = y[n];
  for(i = n - 1; i >= 0; i--) {
    x[i] = y[i] - u[i] * x[i + 1];
  }
}

int main()
{
  int i, j, k;
  int time = TIME;
  int nx = NX;
  int ny = NY;
  int cut = CUT;
  double Dx = DX;
  double Dy = DY;
  double length_x = LENGTH_X;
  double length_y = LENGTH_Y;
  double dt = DT;
  double dx = length_x / (double)nx;
  double dy = length_y / (double)ny;
  double rx = dt / (dx * dx);
  double ry = dt / (dy * dy);
  double old_u[NX + 1][NY + 1], u[NX + 1][NY + 1];
  double um[NX + 1][NY + 1];
  double umx[NY + 1][NX + 1];
  double ax[NX + 1], bx[NX + 1], cx[NX + 1];
  double ay[NY + 1], by[NY + 1], cy[NY + 1];
  double Lx[NX + 1][2], Ux[NX], Bx[NX + 1];
  double Ly[NY + 1][2], Uy[NY], By[NY + 1];
  double Yx[NX + 1], Yy[NY + 1];
  double drx = Dx * rx;
  double dry = Dy * ry;

  /*OpenGL*/
  int Mgx, Mgy;
  double dgx, dgy;
  double rr[(NX + 1) * (NY + 1)];
  double gg[(NX + 1) * (NY + 1)];
  double bb[(NX + 1) * (NY + 1)];
  double umax = 1.;
  double umin = -1.;

  /*OpenGL*/
  Mgx = NX / mod;
  Mgy = NY / mod;
  dgx = dx * (double)mod;
  dgy = dy * (double)mod;

  for(i = 0; i <= nx; i++) {
    ax[i] = 2 * (1 + drx);
    bx[i] = - drx;
    cx[i] = - drx;
  }
  cx[0] = - 2 * drx;
  bx[nx] = - 2 * drx;
```

```
for(i = 0; i <= ny; i++) {
  ay[i] = 2 * (1 + dry);
  by[i] = - dry;
  cy[i] = - dry;
}
cy[0] = - 2 * dry;
by[ny] = - 2 * dry;

initial_u(old_u, dx, dy, nx, ny);
initial_u(u, dx, dy, nx, ny);

/*OpenGL*/
/* 最大値と最小値を求める */
umax = u[0][0];
umin = u[0][0];
for(i=0; i<=NX; i+=mod){
  for(j=0; j<=NY; j+=mod){
    if(umax < u[i][j])     umax = u[i][j];
    if(umin > u[i][j])     umin = u[i][j];
  }
}
mk_color_full(u, rr, gg, bb, mod, umax, umin, nx0, nx, ny0, ny);
draw_(0, Mgx, Mgy, dgx, dgy, rr, gg, bb, "u", "image");

lu_divide(ax, bx, cx, Lx, Ux, nx);
lu_divide(ay, by, cy, Ly, Uy, ny);
for(i = 1; i < time; i++) {
  for(k = 0; k <= nx; k++) {
    Bx[k] = 2 * (1 - dry) * u[k][0] + 2 * dry * u[k][1];
  }
  lu_solve(Yx, Bx, Lx, Ux, umx[0], nx);

  for(k = 0; k <= nx; k++) {
    Bx[k] = 2 * (1 - dry) * u[k][ny] + 2 * dry * u[k][ny - 1];
  }
  lu_solve(Yx, Bx, Lx, Ux, umx[ny], nx);

  for(j = 1; j <= ny - 1; j++) {
    for(k = 0; k <= nx; k++) {
      Bx[k] = dry * u[k][j - 1] + 2 * (1 - dry) * u[k][j] + dry * u[k][j + 1];
    }
    lu_solve(Yx, Bx, Lx, Ux, umx[j], nx);
  }
  for(j = 0; j <= ny; j++) {
    for(k = 0; k <= nx; k++) {
      um[k][j] = umx[j][k];
    }
  }

  for(k = 0; k <= ny; k++) {
    By[k] = 2 * (1 - drx) * um[0][k] + 2 * drx * um[1][k];
  }
  lu_solve(Yy, By, Ly, Uy, u[0], ny);

  for(k = 0; k <= ny; k++) {
    By[k] = 2 * (1 - drx) * um[nx][k] + 2 * drx * um[nx - 1][k];
  }
```

```
    lu_solve(Yy, By, Ly, Uy, u[nx], ny);

    for(j = 1; j <= nx - 1; j++) {
      for(k = 0; k <= ny; k++) {
        By[k] = drx * um[j - 1][k] + 2 * (1 - drx) * um[j][k] + drx * um[j + 1][k];
      }
      lu_solve(Yy, By, Ly, Uy, u[j], ny);
    }

    if(i % cut == 0) {
      /*OpenGL*/
      mk_color_full(u, rr, gg, bb, mod, umax, umin, nx0, nx, ny0, ny);
      draw_(i / cut, Mgx, Mgy, dgx, dgy, rr, gg, bb, "u", "image");
    }
  }
  return 0;
}
```

5.8.2.1 オフスクリーンレンダリングのプログラム（Draw_OSMesa2.c）

このサンプルは空間2次元の数値データを可視化するためのもの関数である.

```
/* Draw_OSMesa2.c */
/* Reference osdemo.c */
/* Demo of off-screen Mesa rendering */
/*
 * See Mesa/include/GL/osmesa.h for documentation of the OSMesa functions.
 *
 * If you want to render BIG images you'll probably have to increase
 * MAX_WIDTH and MAX_HEIGHT in src/config.h.
 *
 * This program is in the public domain.
 *
 * Brian Paul
 *
 * PPM output provided by Joerg Schmalzl.
 */

#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include "GL/osmesa.h"
#include "GL/glu.h"

#define PIXEL 400
#define MS 260

int Width = PIXEL;
int Height = PIXEL;

/* argument */
int Mgx,Mgy;
int fn;
double dgx,dgy;
double *u,*v,*w;
```

```
double x_range,y_range,xy_range;

static void render_image(void)
{
  int i,j;
  double *p,*q,*r,*pp,*qq,*rr;

  p = u; q = v; r = w;
  pp = u + (Mgx + 1); qq = v + (Mgx + 1); rr = w + (Mgx + 1);
  for (j=0;j<Mgy;j++){
    for (i=0;i<Mgx;i++){
      glBegin(GL_POLYGON);
      glColor3f(*p,*q,*r);
      glVertex2f(dgx*i,dgy*j);
      glColor3f(*(p+1), *(q+1), *(r+1));
      glVertex2f(dgx*(i+1),dgy*j);
      glColor3f(*(pp+1), *(qq+1), *(rr+1));
      glVertex2f(dgx*(i+1),dgy*(j+1));
      glColor3f(*pp,*qq,*rr);
      glVertex2f(dgx*i,dgy*(j+1));
      glEnd();
      p++;q++;r++;pp++;qq++;rr++;
    }
    p++;q++;r++;pp++;qq++;rr++;
  }
  glBegin(GL_LINE_LOOP);
  glColor3f(0.0,0.0,0.0);
  glVertex2f(0.0,0.0);
  glVertex2f(x_range,0.0);
  glVertex2f(x_range,y_range);
  glVertex2f(0.0,y_range);
  glEnd();
}
void set_view(){
  glClearColor(1.0, 1.0, 1.0, 1.0);
  glClear(GL_COLOR_BUFFER_BIT);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef(-0.5*x_range,-0.5*y_range,0.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(-0.5*x_range-0.1*xy_range,0.5*x_range+0.1*xy_range,
     -0.5*y_range-0.1*xy_range,0.5*y_range+0.1*xy_range);
  glRectf(-0.1*x_range,-0.1*y_range,1.1*x_range,1.1*y_range);
}
void save_file(FILE *f, void *buffer){
  /* write PPM file */
  if (f) {
    int i, x, y;
    GLubyte *ptr;
    ptr = buffer;
    fprintf(f,"P6\n");
    fprintf(f,"# ppm-file created\n");
    fprintf(f,"%i %i\n", Width,Height);
    fprintf(f,"255\n");
    for (y=Height-1; y>=0; y--) {
      for (x=0; x<Width; x++) {
```

```
        i = (y*Width + x) * 4;
        fputc(ptr[i], f);   /* write red */
        fputc(ptr[i+1], f); /* write green */
        fputc(ptr[i+2], f); /* write blue */
      }
    }
    fclose(f);
  }
  printf("all done\n");
}
void change_image_format(char *in, char *out)
{
   char string[300];
   sprintf(string, "convert %s %s", in, out);
   system(string);
}

/* For Fortran */
/* void draw_( int *p_fn, int *p_Mgx, int *p_Mgy, double *p_dgx, double *p_dgy,
  double *p, double *q, double *r, char *image_file, char *image_dir)
*/
/* For C */
void draw_( int p_fn, int p_Mgx, int p_Mgy, double p_dgx, double p_dgy,
          double *p, double *q, double *r, char *image_file, char *image_dir)

{
  char ppmfile[BUFSIZ],jpegfile[BUFSIZ];
  FILE *fp;
  OSMesaContext ctx;
  void *buffer;


  fn = p_fn; Mgx = p_Mgx; Mgy = p_Mgy; dgx = p_dgx; dgy = p_dgy;
  u = p; v = q; w = r;
  printf("%d %d %d %lf %lf\n", fn, Mgx, Mgy, dgx, dgy);

  x_range = dgx*Mgx;y_range=dgy*Mgy;
  xy_range = ((x_range > y_range) ? x_range : y_range);
  Width  = (int)((x_range / xy_range + 0.2) / 1.2 * PIXEL);
  Height = (int)((y_range / xy_range + 0.2) / 1.2 * PIXEL);

  /* Create an RGBA-mode context */
  ctx = OSMesaCreateContext( GL_RGBA, NULL );

  /* Allocate the image buffer */
  buffer = malloc( Width * Height * 4 );

  /* Bind the buffer to the context and make it current */
  OSMesaMakeCurrent( ctx, buffer, GL_UNSIGNED_BYTE, Width, Height );

  set_view();
  render_image();
  glFlush();

  sprintf(ppmfile, "%s/%s%04d.ppm",image_dir,image_file,fn);
  sprintf(jpegfile,"%s/%s%04d.jpg",image_dir,image_file,fn);
```

```
  if ((fp = fopen(ppmfile, "w")) == NULL){
    perror(ppmfile);
    exit(1);
  }
  save_file(fp, buffer);
  /*fclose(fp); */
  change_image_format(ppmfile,jpegfile);
  unlink(ppmfile);
  /* free the image buffer */
  free(buffer);
  /* destroy the context */
  OSMesaDestroyContext(ctx);
/*  return 0; */
}
```

5.8.2.2 mk_color_full.c

mk_color_full は数値データから RGB 値を決める関数である.

```
#include<stdio.h>
#define Nx         100
#define Ny         50
void mk_color_full(double x[][Ny+1], double R[], double G[], double B[], \
   int mod, double xmax, double xmin, \
   int nx0, int nx, int ny0, int ny)
{
  int i, j, n;
  double color;
  n = 0;
  for(j=ny0; j<=ny; j+=mod){
    for(i=nx0; i<=nx; i+=mod){
      color = (x[i][j]-xmin)/(xmax - xmin);
      if(color < 0.0){
        R[n] = 0.0;
        G[n] = 0.0;
        B[n] = 1.0;
      }else if(color >= 1.0){
        R[n] = 1.0;
        G[n] = 0.0;
        B[n] = 0.0;
      }else if( (color >= 0.0) && (color < 0.250)){
        R[n] = 0.0;
        G[n] = 4.0*color;
        B[n] = 1.0;
      }else if( (color >= 0.250) && (color < 0.50)){
        R[n] = 0.0;
        G[n] = 1.0;
        B[n] = 2.0 - 4.0*color;
      }else if( (color >= 0.50) && (color < 0.750)){
        R[n] = 4.0*color - 2.0;
        G[n] = 1.0;
        B[n] = 0.0;
      }else if( (color >= 0.750) && (color < 1.0)){
        R[n] = 1.0;
        G[n] = 4.0 - 4.0*color;
```

```
        B[n] = 0.0;
      }
      n += 1;
    }
  }
  printf("n = %d\n", n-1);
}
```

5.8.2.3　Makefile (C 言語用)

上記の Draw_OSMesa2.c と mk_color_full.c を C 言語の数値計算プログラム heat1d.c に挿入して使用するための Makefile のサンプル (Makefile_C_OSMesa)：

```
#make
DIR1 = /usr
DIR2 = /usr/X11R6
DIR3 = /usr/local
#  Linux
INCDIR = -I$(DIR2)/include -I$(DIR1)/include -I$(DIR3)/include
LIBDIRS = -L$(DIR1)/lib -L$(DIR2)/lib -L$(DIR3)/lib
XLIBS = -lX11 -lXpm -lXext
GLUTLIBS = -lXmu -lXi
#
CC = gcc -O3 -Wall
#
OSMESA = -lOSMesa -lGLU -lGL -lm $(XLIBS) $(GLUTLIBS) $(F2CLIBS)
#
#
#
SRC_OBJ = heat1d.c
COLOR_OBJ = mk_color_full.c

main03 : $(SRC_OBJ) $(COLOR_OBJ) draw_osmesa
$(CC) -o a.out  $(SRC_OBJ) \
$(COLOR_OBJ) $(INCDIR) $(LIBDIRS) Draw_OSMesa2.o $(OSMESA)
clean :
rm -f *.o
#
draw_osmesa: Draw_OSMesa2.c
$(CC) $(INCDIR) $(CFLAGS) -c Draw_OSMesa2.c
```

サンプルプログラムのコンパイル方法と実行方法

```
$ make -f Makefile_C_OSMesa
```

実行する前に

```
$ mkdir image
```

として，画像ファイルが保存されるディレクトリを作っておく．

```
$ ./a.out # 実行
```

5.8.2.4 数値計算をおこないながら結果をモニターする（OpenGL 版）

以下のサンプル (OpenGL_Disp01.c) は数値計算を行いながら可視化し，アニメーションに必要な画像 (JPEG ファイル) をつくるためのものである．

```
#include <stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<GL/glut.h>
#include<GL/glu.h>
#include<GL/gl.h>

/* 画像のピクセル数 */
#define WIDTH 500
#define HEIGHT 500

/* 数値計算に必要なパラメータ */
/* 当然数値計算の内容によって書き換える必要がある */
#define MAXTIME 10
#define DT 0.0005
#define E 0.8
#define G 9.8

/* 必要な宣言 */
GLuint startList;
void display(double y);

/* 画像を PPM ファイルとして保存する */
void save_file(FILE *f){
  if (f) {
      int i, x, y;
      GLubyte *ptr;
      ptr = malloc( WIDTH * HEIGHT * 4 );

      glReadPixels(0, 0, WIDTH, HEIGHT, GL_RGBA,
                GL_UNSIGNED_BYTE, ptr);
      fprintf(f,"P6\n");
      fprintf(f,"# ppm-file created\n");
      fprintf(f,"%i %i\n", WIDTH,HEIGHT);
      fprintf(f,"255\n");
      for (y = HEIGHT-1; y >= 0; y--) {
        for (x = 0; x < WIDTH; x++) {
          i = (y*WIDTH + x) * 4;
          fputc(ptr[i], f);   /* write red */
          fputc(ptr[i+1], f); /* write green */
          fputc(ptr[i+2], f); /* write blue */
        }
      }
    free(ptr);
  }
}
/* PPM ファイル → JPEG ファイル：ImageMagick を使っている*/
void change_image_format(char *in, char *out)
{
   char string[300];
   sprintf(string, "convert %s %s", in, out);
   system(string);
}
```

```
/* 数値計算する部分 */
double f1(double t, double x, double v)
{
  return - G;
}

double f2(double t, double x, double v)
{
  return v;
}
void rakka()
{
  int i;
  double x, v;
  double new_x, new_v;
  double t;
  double dt = DT;
  double e = E;
  double maxtime = MAXTIME;
  int n = (int)(maxtime / dt);

  char ppmfile[BUFSIZ],jpegfile[BUFSIZ];
  FILE *fp;

  x = 0.;
  v = 4.9;

  for(i = 1; i <= n; i++) {
    t = dt * i;
    new_x = x + dt * f2(t, x, v);
    new_v = v + dt * f1(t, x, v);
    x = new_x;
    v = new_v;
    //printf("%lf %lf\n", t, x);
    display(x);
    if(x <= 0) {
      v = - e * v;
      x = 0.;
    }

    if((i-1)%50 == 0){
      sprintf(ppmfile, "./image/Image%04d.ppm",(i-1)/50);
      sprintf(jpegfile,"./image/Image%04d.jpg",(i-1)/50);
      if ((fp = fopen(ppmfile, "w")) == NULL){
        perror(ppmfile);
        exit(1);
      }
      save_file(fp);
      fclose(fp);
      change_image_format(ppmfile,jpegfile);
      unlink(ppmfile);
    }
  }
  return;
}

/* エラー処理 */
```

```
void errorCallback(GLenum errorCode)
{
  const GLubyte *estring;

  estring = gluErrorString(errorCode);
  fprintf(stderr,"Quadric Error: %s\n",estring);
  exit(0);
}

/* 画面の初期化 */
void init(void)
{
  GLUquadricObj *qobj;
  GLfloat mat_ambient[] = { 1.0, 0.0, 0.0,1.0 };
  GLfloat mat_specular[] = { 1.0, 1.0, 1.0,1.0 };
  GLfloat mat_shininess[] = { 50.0 };
  GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0 };
  GLfloat model_ambient[] = { 0.5, 0.5, 0.5, 1.0 };

  glClearColor(1.0, 1.0, 1.0, 1.0);

  glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
  glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT, model_ambient);

  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glEnable(GL_DEPTH_TEST);

  startList = glGenLists(1);
  qobj = gluNewQuadric();
  /* gluQuadricCallback(qobj, GLU_ERROR, errorCallback); */
  gluQuadricDrawStyle(qobj,GLU_FILL);
  gluQuadricNormals(qobj, GLU_SMOOTH);
  glNewList(startList, GL_COMPILE);
  gluSphere(qobj, 0.1, 10, 10);
  glEndList();
}

/* 可視化部分：一番大事 */
void display(double y)
{
  glClear(GL_COLOR_BUFFER_BIT| GL_DEPTH_BUFFER_BIT);
  glPushMatrix();
  glEnable(GL_LIGHTING);
  glShadeModel(GL_SMOOTH);
  glTranslatef(0.0, y, 0.0);
  glCallList(startList);
  glPopMatrix();
  glFlush();
}

void reshape(int w, int h)
{
  glViewport(0, 0, (GLsizei) w, (GLsizei) h);
```

```
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

  if( w <= h ){

    glOrtho(-2.5, 2.5, -2.5*(GLfloat)h/(GLfloat)w, 2.5*(GLfloat)h/(GLfloat)w,-10.0,10.0);
  }else{
    glOrtho(-2.5*(GLfloat)h/(GLfloat)w, 2.5*(GLfloat)h/(GLfloat)w,-2.5, 2.5,-10.0,10.0);
  }

  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

/* キーボードからの入力 */
void keyboard(unsigned char key, int x, int y)
{
  switch( key ){
  case 27:
    exit(0);
    break;
  }
}

/* メインループ */
int main(int argc, char *argv[])
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE| GLUT_RGB| GLUT_DEPTH);
  glutInitWindowSize(500, 500);
  glutInitWindowPosition(200,200);
  glutCreateWindow(argv[0]);
  init();
  glutDisplayFunc(rakka);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```

サンプルプログラムのコンパイル方法と実行方法

```
$ gcc -O3 -Wall OpenGL_Disp01.c -o a.out -I/usr/X11R6/include -L/usr/X11R6/lib
 -lglut -lGLU -lGL -lXmu -lXi -lXext -lX11 -lm
```

実際は一行で書くこと！

実行する前に

```
$ mkdir image
```

として，画像ファイルが保存されるディレクトリを作っておく．

```
$ ./a.out # 実行
```

付 録A　数値スキームの安定性

A.1　拡散方程式の数値スキームの安定性解析 その１

拡散方程式の数値スキームの安定性に関しては [11] を参照するとよい.

A.1.1　陽解法 (Explicit Scheme) の安定性解析

$$u_j^{n+1} = \mu u_{j+1}^n + (1-2\mu)u_j^n + \mu u_{j-1}^n \tag{A.1}$$

ただし，$\mu = D\Delta x^2/\Delta t$，$x_j = j\Delta x$ である．波数 k の正弦波成分の成長を見るために

$$u_j^n = \lambda^n \exp(ikx_j)$$

とおく．もしある波数 k に対して $|\lambda| > 1$ であるならば，ノイズの中の波数 k の成分が指数的に成長してしまう → 数値的不安定性
よって全ての波数 k に対し $|\lambda| \leq 1$ でなければならない.

$$\begin{aligned}
\lambda^{n+1}\exp(ikx_j) &= \mu\lambda^n\exp(ikx_{j+1}) + (1-2\mu)\lambda^n\exp(ikx_j) + \mu\lambda^n\exp(ikx_{j-1}) \\
\lambda &= \mu\exp(ik\Delta x) + (1-2\mu) + \mu\exp(-ik\Delta x) \\
&= 1 - 2\mu(1-\cos k\Delta x) = 1 - 4\mu\sin^2\frac{k\Delta x}{2}
\end{aligned}$$

よって $\lambda \leq 1$ は常に成立するので $\lambda \geq -1$ をチェック

$$1 - 4\mu\sin^2\frac{k\Delta x}{2} \geq -1 \quad \text{for all} \quad k,$$

$$1 - 4\mu \geq -1.$$

故に

$$D\frac{\Delta t}{\Delta x^2} \leq \frac{1}{2}.$$

A.1.2　完全陰解法 (Implicit Scheme) の安定性解析

$$-\mu u_{j+1}^{n+1} + (1+2\mu)u_j^{n+1} - \mu u_{j-1}^{n+1} = u_j^n. \tag{A.2}$$

ただし，$\mu = D\Delta x^2/\Delta t$，$x_j = j\Delta x$ である.

$$\begin{aligned}
-\mu\lambda^{n+1}\exp(ikx_{j+1}) + (1+2\mu)\lambda^{n+1}\exp(ikx_j) - \mu\lambda^{n+1}\exp(ikx_{j-1}) &= \lambda^n\exp(ikx_j), \\
\lambda(-\mu\exp(ik\Delta x) + (1+2\mu) - \mu\exp(-ik\Delta x)) &= 1, \\
\lambda\{1 + 2\mu(1-\cos k\Delta x)\} &= 1, \\
\lambda &= \frac{1}{1 + 4\mu\sin^2\frac{k\Delta x}{2}}.
\end{aligned}$$

故に，任意の k に対して $|\lambda| \leq 1$ となるので陰解法は無条件安定である.

A.2 拡散方程式の数値スキームの安定性解析 その２

A.2.1 差分方程式の行列表示

$$\frac{\partial^2 u}{\partial x^2} \cong \frac{u(t,(j+1)\Delta x) - 2u(t,j\Delta x) + u(t,(j-1)\Delta x)}{\Delta x^2} + O(\Delta x^2) \tag{A.3}$$

に対し, $t = (k+1)\Delta t,\ \Delta t$ を混合して考える.

$$\begin{aligned}\text{(A.3) の右辺} \cong & \theta \frac{u((k+1)\Delta t,(j+1)\Delta x) - 2u((k+1)\Delta t, j\Delta x) + u((k+1)\Delta t,(j-1)\Delta x)}{\Delta x^2} \\ & + (1-\theta)\frac{u(k\Delta t,(j+1)\Delta x) - 2u(k\Delta t, j\Delta x) + u(k\Delta t,(j-1)\Delta x)}{\Delta x^2}\end{aligned} \tag{A.4}$$

と近似する．このとき

1. $\theta = 0$ のとき **完全陰解法**.
2. $\theta = 1/2$ のとき **クランク · ニコルソン法**.
3. $\theta = 1$ のとき **陽解法**.

となる.

(A.4) を用いると熱方程式の差分方程式は

$$\frac{u_j^{k+1} - u_j^k}{\Delta t} = \theta \frac{u_{j+1}^{k+1} - 2u_j^{k+1} + u_{j-1}^{k+1}}{\Delta x^2} + (1-\theta)\frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{\Delta x^2}$$

となる．ここで，$r = \Delta t/\Delta x^2$ とおくと

$$-r\theta u_{j-1}^{k+1} + (1+2\theta r)u_j^{k+1} - \theta r u_{j+1}^{k+1} = (1-\theta)r u_{j-1}^k + (1-2(1-\theta)r)u_j^k + (1-\theta)r u_{j+1}^k. \tag{A.5}$$

と書ける．境界条件 $u_0^k = a \quad u_N^k = b$ より

1. $j = 1$ のとき
$$(1+2\theta r)u_1^{k+1} - \theta r u_2^{k+1} = ar + (1-2(1-\theta)r)u_1^k + (1-\theta)r u_2^k. \tag{A.6}$$
2. $j = N-1$ のとき
$$-r\theta u_{N-2}^{k+1} + (1+2\theta r)u_{N-1}^{k+1} = (1-\theta)r u_{N-2}^k + (1-2(1-\theta)r)u_{N-1}^k + rb. \tag{A.7}$$

(A.5)-(A.7) より

$$\begin{aligned}&\begin{pmatrix} 1+2\theta r & -\theta r & 0 & 0 & \cdots & 0 \\ -\theta r & 1+2\theta r & -\theta r & 0 & \cdots & 0 \\ 0 & -\theta r & 1+2\theta r & -\theta r & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\theta r & 1+2\theta r & -\theta r \\ 0 & \cdots & 0 & 0 & -\theta r & 1+2\theta r \end{pmatrix} \begin{pmatrix} u_1^{k+1} \\ u_2^{k+1} \\ \vdots \\ \vdots \\ u_{N-2}^{k+1} \\ u_{N-1}^{k+1} \end{pmatrix} \\ &= \begin{pmatrix} ra + (1-2(1-\theta)r)u_1^k + (1-\theta)r u_2^k \\ \vdots \\ (1-\theta)r u_{j-1}^k + (1-2(1-\theta)r)u_j^k + (1-\theta)r u_{j+1}^k \\ \vdots \\ (1-\theta)r u_{N-2}^k + (1-2(1-\theta)r)u_{N-1}^k + rb \end{pmatrix}\end{aligned} \tag{A.8}$$

となり，これは

$$A=\begin{pmatrix} 1+2\theta r & -\theta r & 0 & 0 & \cdots & 0 \\ -\theta r & 1+2\theta r & -\theta r & 0 & \cdots & 0 \\ 0 & -\theta r & 1+2\theta r & -\theta r & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\theta r & 1+2\theta r & -\theta r \\ 0 & \cdots & 0 & 0 & -\theta r & 1+2\theta r \end{pmatrix},\ \boldsymbol{u}^{k+1}=\begin{pmatrix} u_1^{k+1} \\ u_2^{k+1} \\ \vdots \\ \vdots \\ u_{N-2}^{k+1} \\ u_{N-1}^{k+1} \end{pmatrix},$$

$$\boldsymbol{b}^k=\begin{pmatrix} ra+(1-2(1-\theta)r)u_1^k+(1-\theta)ru_2^k \\ \vdots \\ (1-\theta)ru_{j-1}^k+(1-2(1-\theta)r)u_j^k+(1-\theta)ru_{j+1}^k \\ \vdots \\ (1-\theta)ru_{N-2}^k+(1-2(1-\theta)r)u_{N-1}^k+rb \end{pmatrix}$$

とすると，差分方程式の解は連立一次方程式

$$A\boldsymbol{u}^{k+1}=\boldsymbol{b}^k$$

を解くことによって求められる．ところで

$$\text{(A.8) の右辺}=\begin{pmatrix} 1-2(1-\theta)r & (1-\theta)r & & & & 0 \\ (1-\theta)r & 1-2(1-\theta)r & (1-\theta)r & & & \\ & \ddots & \ddots & \ddots & & \\ & & (1-\theta)r & 1-2(1-\theta)r & (1-\theta)r & \\ 0 & & & (1-\theta)r & 1-2(1-\theta)r & \end{pmatrix}\begin{pmatrix} u_1^k \\ u_2^k \\ \vdots \\ u_{N-2}^k \\ u_{N-1}^k \end{pmatrix}+\begin{pmatrix} ra \\ 0 \\ \vdots \\ 0 \\ rb \end{pmatrix}.$$

$$\stackrel{\text{def}}{=} B\boldsymbol{u}^k+\boldsymbol{b}.$$

と書けるので，連立一次方程式は次のように書き下すことができる．

$$A\boldsymbol{u}^{k+1}=B\boldsymbol{u}^k+\boldsymbol{b}.$$

ここで

$$M=\begin{pmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{pmatrix}$$

とすると

$$A=I-\theta rM,$$

$$B=I+(1-\theta)rM$$

と書ける．ただし，I は単位行列である．

A.2.2 差分方程式の安定性

A.2.2.1 陽解法の安定性

陽解法 ($\theta=0$ のとき) の安定性は

$$\begin{cases} A=I, \\ B=I+rM \end{cases}$$

となるので

$$\boldsymbol{u}^{k+1} = B\boldsymbol{u}^k + \boldsymbol{b}$$

と書ける．従って

$$\begin{aligned}\mathbf{u}^{k+1} &= B(B\boldsymbol{u}^{k-1} + \boldsymbol{b}) + \mathbf{b} \\ &\vdots \\ &= B^{k+1}\boldsymbol{u}^0 + \sum_{l=0}^{k} B^l \boldsymbol{b}.\end{aligned}$$

今，$\rho(B) = \max\limits_{1\le i\le N-1}|\lambda_i|$ （λ_i は B の固有値とする) とおくと

$$\rho(B) < 1 \quad \Longleftrightarrow \quad \lim_{k\to\infty} B^k = 0,$$

$$\rho(B) > 1 \quad \Longleftrightarrow \quad \lim_{k\to\infty} B^k = \infty.$$

差分方程式が安定であるためには（発散しないためには），$\lim\limits_{k\to\infty} B^k = 0$ でなければならない．つまり $\rho(B) < 1$ を満たさなければならない．従って，B の固有値を調べればよい．今，M の固有値を λ，固有空間を $\boldsymbol{x}$ とすると

$$B\boldsymbol{x} = (I + rM)\boldsymbol{x} = (1 + r\lambda)\boldsymbol{x}$$

より，B の固有値は $(1 + r\lambda)$ となる. 従って M の固有値を調べる.

$$M\boldsymbol{x} = \lambda\boldsymbol{x}$$

とする.

$$\begin{pmatrix} \lambda+2 & -1 & & & 0 \\ -1 & \lambda+2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & \lambda+2 & -1 \\ 0 & & & -1 & \lambda+2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-2} \\ x_{N-1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$

より

$$\begin{aligned} (\lambda+2)x_1 - x_2 &= 0, \\ -x_{j-1} + (\lambda+2)x_j - x_{j+1} &= 0, \qquad (j = 2, 3, \cdots, N-2), \\ -x_{N-2} + (\lambda+2)x_{N-1} &= 0. \end{aligned}$$

を得る. ここで,

第二種チェビシェフ多項式について

$$-U_{j-1}(x) + 2xU_j(x) - U_{j+1}(x) = 0 \qquad (\#)$$

の解は

$$U_j(x) = \sin(n\theta), \qquad x = \cos\theta$$

と書ける.

故に，θ を任意として $x_j(\lambda) = \sin(j\theta)$．ただし，$\lambda/2 + 1 = \cos\theta$ となる．$-x_{N-2} + (\lambda+2)x_{N-1} = 0$ より

$$-\sin(N-2)\theta + 2\cos\theta\sin(N-1)\theta = 0. \tag{A.9}$$

$\sin j\theta$ は $j = N-1$ において漸化式 (#) をみたすので

$$-\sin(N-2)\theta + 2\cos\theta\sin(N-1)\theta - \sin N\theta = 0. \tag{A.10}$$

(A.9) かつ (A.10) をみたすような θ を決める.

$$\begin{aligned}\sin N\theta &= 0,\\ N\theta &= m\pi, \quad (1 \le m \le N-1),\\ \therefore \theta_m &= \frac{m\pi}{N}.\end{aligned}$$

故に

$$\begin{aligned}\lambda_m &= -2(1-\cos\theta),\\ &= -2(1-\cos\frac{m\pi}{N}), \quad (1 \le m \le N-1),\\ &= -4\sin^2\frac{m\pi}{2N}.\end{aligned} \tag{A.11}$$

固有空間 $\boldsymbol{x}^m$ は

$$\boldsymbol{x}^m = \begin{pmatrix} \vdots \\ \sin\frac{jm\pi}{N} \\ \vdots \end{pmatrix}_{1\le j\le N-1}.$$

(A.11) より, B の固有値は $1-4r\sin^2\dfrac{m\pi}{2N}$, $\quad 1 \le m \le N-1$. 従って, $-1 < 1-4r\sin^2\dfrac{m\pi}{2N} < 1$ のとき, $\lim\limits_{k\to\infty} B^k = 0$ となり，差分方程式は安定となる．従って

$$\begin{aligned}-1 &< 1-4r,\\ r &< \frac{1}{2}.\end{aligned}$$

A.2.2.2 陰解法の安定性

完全陰解法 ($\theta = 1$ のとき) の安定性

$$\begin{cases} A = I - rM, \\ B = I \end{cases}$$

となるので，差分方程式は $A\boldsymbol{u}^{k+1} = \boldsymbol{u}^k$ となる.
ここで M の固有値 λ_m と固有空間 $\boldsymbol{v}_m$ に対して

$$A\boldsymbol{v}_m = (I - rM)\boldsymbol{v}_m,$$

$$1 - r\lambda_m = 1 + 4r\sin^2\frac{m\pi}{2N} \ge 1$$

より，A は逆行列 A^{-1} を持つ．従って

$$\boldsymbol{u}^{k+1} = A^{-1}\boldsymbol{u} = \cdots = (A^{-1})^{k+1}\boldsymbol{u}_0.$$

安定性は $\rho(A^{-1}) < 1$ で決まる.
A^{-1} の固有値は A の固有値 $\lambda_A = 1 - r\lambda_m$ の逆数より，A^{-1} の固有値は

$$\begin{aligned}\lambda_{A^{-1}} &= \frac{1}{1-r\lambda_m},\\ &= \frac{1}{1+4r\sin^2\frac{m\pi}{2N}}.\end{aligned}$$

ここで

$$\Big|\frac{1}{1+4r\sin^2\frac{m\pi}{2N}}\Big| \le 1$$

より，任意の r に対して $\rho(A^{-1}) < 1$ となる．〃$r = \Delta t/\Delta x^2$ に制約無し〃という意味で，**無条件安定** であるという．

A.2.2.3 クランクーニコルソン法の安定性

クランクーニコルソン法 ($\theta = 1/2$ のとき) の安定性.

$$\begin{cases} A = I - \dfrac{r}{2}M, \\ B = I + \dfrac{r}{2}M. \end{cases}$$

なので, 差分方程式は $A\boldsymbol{u}^{k+1} = B\boldsymbol{u}^k$ となる．これまで同様 A の固有値を調べる．A の固有値は

$$1 + 2r\sin^2\frac{m\pi}{2N} \ge 1$$

より，A は逆行列を持つ．従って

$$\boldsymbol{u}^{k+1} = A^{-1}B\boldsymbol{u}^k = \cdots = (A^{-1}B)^{k+1}\boldsymbol{u}_0.$$

安定性は $\rho(A^{-1} \le 1)$ で決まる．ただし，数値計算上では $\rho(A^{-1}) < 1$ としなければならない.
M の固有値 λ_m と固有空間 $\boldsymbol{v}_m$ に対して,

$$\begin{aligned} B\boldsymbol{v}_m &= (I + \frac{r}{2}M)\boldsymbol{v}_m = (1 + \frac{r}{2}\lambda_m)\boldsymbol{v}_m, \\ A\boldsymbol{v}_m &= (I - \frac{r}{2}M)\boldsymbol{v}_m = (1 - \frac{r}{2}\lambda_m)\boldsymbol{v}_m. \end{aligned}$$

従って

$$A^{-1}\boldsymbol{v}_m = \frac{1}{(1 - \frac{r}{2}\lambda_m)}\boldsymbol{v}_m$$

となるので

$$\begin{aligned} A^{-1}B\boldsymbol{v}_m &= A^{-1}(1 + \frac{r}{2}\lambda_m)\boldsymbol{v}_m, \\ &= (1 + \frac{r}{2}\lambda_m)A^{-1}\boldsymbol{v}_m, \\ &= \frac{(1 + \frac{r}{2}\lambda_m)}{(1 - \frac{r}{2}\lambda_m)}\boldsymbol{v}_m. \end{aligned}$$

$A^{-1}B$ の固有値 $\lambda_{A^{-1}B}$ は

$$\begin{aligned} \lambda_{A^{-1}B} &= \frac{(1 + \frac{r}{2}\lambda_m)}{(1 - \frac{r}{2}\lambda_m)}, \\ &= \frac{1 - 2r\sin^2\frac{m\pi}{2N}}{1 + 2r\sin^2 m\pi 2N}. \end{aligned}$$

従って，安定であるためには

$$-1 \le \frac{1 - 2r\sin^2\frac{m\pi}{2N}}{1 + 2r\sin^2 m\pi 2N} \le 1$$

となればよい. $\theta = 1/2$ のとき，この不等式は自動的に成立するので無条件に安定である.

問題；混合解法

$$u_i^{n+1} - u_i^n = D\theta(ru_{i-1}^{n+1} - 2ru_i^{n+1} + ru_{i+1}^{n+1}) + D(1-\theta)(ru_{i-1}^n - 2ru_i^n + ru_{i+1}^n), \quad 0 \le i \le N. \text{ (A.12)}$$

について安定性条件を求めよ.

関連図書

[1] “やさしく学べる C 言語入門 - 基礎から数値計算入門まで - ”，皆本晃弥，サイエンス社

[2] “C 言語による数値計算入門 - 解法・アルゴリズム・プログラム - ”，皆本晃弥，サイエンス社

[3] “偏微分方程式の数値シミュレーション”，登坂 宣好，大西和榮，東京大学出版会.

[4] “コンピュータによる偏微分方程式の解法［新訂版］”，G.D. スミス，サイエンス社.

[5] “Fortran & C 言語によるシミュレーション技法入門”，矢部 孝，井門 俊治，日刊工業新聞社.

[6] “岩波講座 応用数学「線形計算」”，森 正武，杉原 正顯，室田 一雄，岩波書店.

[7] “Fortran77 による数値計算ソフトウェア”，渡部 力，名取 亮，小国 力，丸善.

[8] “行列計算ソフトウェア”，小国 力編著，丸善.

[9] “NUMERICAL RECIPES in C”，技術評論社.

[10] “数値解析入門”，山本 哲朗，サイエンス社.

[11] “数値解析の基礎”，篠原 能材，日新出版.

[12] “数値計算の常識”，伊理正夫，藤野和建，共立出版.

[13] “ http://www.math.meiji.ac.jp/ mk/labo/text/heat-fdm-1.pdf”.

[14] “コンピュータ流体力学”，フレッチャー，シュプリンガーフェアーラーク東京.

[15] “使いこなす GNUPLOT Ver4.0 対応”，大竹 敢，テクノプレス.

[16] “http://ayapin.film.s.dendai.ac.jp/ matuda/TeX/PDF/40th.pdf”

[17] “http://www-sens.sys.es.osaka-u.ac.jp/wakate/tutorial/group3/gnuplot/”

[18] “OpenGL プログラミングガイド”，OpenGL ARB，アジソンウェスレイ.

[19] “OpenGL リファレンスマニュアル”，OpenGL ARB，アジソンウェスレイ.

[20] “http://www.wakayama-u.ac.jp/ tokoi/opengl/libglut.html”.

[21] “ftp://ftp.ryukoku.ac.jp/Ryukoku/software/math/glsc-3.5.tar.Z”.

[22] “ftp://ftp.ryukoku.ac.jp/Ryukoku/software/math/glsc-3.3.man.tar.Z”.

[23] “非平衡系の科学 I”，北原 和夫，吉川 研一，講談社.

[24] “非平衡系の科学 III”，三池 秀敏，森 義仁，山口 智彦，講談社.

[25] “非線形科学”，吉川 研一，学会出版センター.