

“私にとっての”
反応拡散系数値計算入門
基礎編

2021 年度数値計算チュートリアル

2024 年 3 月 : OpenMP を使った ADI 法の数値計算法を追加しました.

長山 雅晴

北海道大学電子科学研究所 教授

E-mail : nagayama@es.hokudai.ac.jp

目次

第 1 章	はじめに	7
1.1	数値シミュレーションとは	7
1.2	数値シミュレーションプログラムのあらすじ	8
1.3	入門編の目標	8
第 2 章	常微分方程式の数値計算法	11
2.1	Euler 法 [9, 10, 11]	11
2.2	Runge-Kutta 法 [10]	11
2.3	問題	13
2.3.1	問題 1	13
2.3.2	問題 2	14
2.3.3	問題 3	14
第 3 章	拡散方程式の数値計算法 (差分法)	15
3.1	熱方程式 (拡散方程式) の導出 [3]	15
3.2	1 次元初期境界値問題 [3, 4]	16
3.2.1	陽解法	17
3.2.2	陰解法	18
3.2.3	反復改良法 [6]	21
3.2.4	問題	22
3.3	2 次元長方形領域における拡散方程式の数値計算	22
3.3.1	陽解法 [4]	22
3.3.2	陰解法 [4]	23
3.3.3	ADI 法 [5]	23
3.3.4	2 次元 SSI 法 [5]	24
3.4	3 次元直方体領域での数値計算法	24
3.4.1	陽解法	24
3.4.2	ADI 法 [5]	24
3.4.3	陰解法	25
第 4 章	反応拡散方程式の数値計算法 (差分法)	27
4.1	1 次元反応拡散系の数値計算法	27
4.1.1	1 次元反応拡散系の初期・境界値問題	27
4.1.2	2 変数反応拡散系の数値計算法 (半陰解法)	28
4.1.3	差分方程式の連立 1 次方程式表示	31
4.2	2 次元反応拡散系の数値計算法	35
4.2.1	方程式の離散化	35
4.2.2	非線形項の離散化	36
4.2.3	初期条件の離散化	36
4.2.4	境界条件の離散化	36

4.2.5	ADI 法	37
第 5 章	数値計算環境の構築と計算結果の可視化	43
5.1	数値データの可視化	43
5.2	数値計算と結果の可視化に必要なソフトウェアのインストール	43
5.2.1	VMWare	44
5.2.2	macOS 上で数値計算環境を作る	45
5.2.3	Linux で必要なソフトのインストール	46
5.3	必要になるかもしれないソフトウェアのインストール	47
5.3.1	GLSC のインストール	47
5.3.2	vlc のインストール	48
5.4	GNU PLOT による数値計算結果の可視化 [15]	48
5.4.1	一般的な注意	49
5.4.2	画像ファイルに出力する	49
5.4.3	Gnuplot を C 言語から呼び出す [16, 17]	50
5.5	Gnuplot を用いた 2 次元データの可視化	52
5.5.1	動画作成 その 1	55
5.5.2	動画作成 (MPEG1) その 2	57
5.6	OpenGL によるオフスクリーンレンダリング	58
5.7	可視化の参考プログラム	60
5.7.1	GLSC による鳥瞰図の参考プログラム	60
5.7.2	OSMesa(OpenGL) のプログラム	62
付 録 A	数値スキームの安定性	75
A.1	拡散方程式の数値スキームの安定性解析 その 1	75
A.1.1	陽解法 (Explicit Scheme) の安定性解析	75
A.1.2	完全陰解法 (Implicit Scheme) の安定性解析	75
A.2	拡散方程式の数値スキームの安定性解析 その 2	76
A.2.1	差分方程式の行列表示	76
A.2.2	差分方程式の安定性	77
A.3	差分スキームの収束性	81
付 録 B	発展編の完成部分 (連立 1 次方程式の反復解法)	83
B.1	Jacobi 法, Gauss-Seidel 法, SOR 法	83
B.1.1	Jacobi 法	84
B.1.2	Gauss-Seidel 法	84
B.1.3	SOR 法	85
B.1.4	反復法の収束定理	85
B.2	共役勾配法 (Conjugate Gradient method)	86
B.2.1	CG 法について	87
B.2.2	アルゴリズム	88
B.2.3	原型版と実用版の違いについて	89
B.2.4	PCG 法	89
B.2.5	ICCG 法	92
B.3	CR 法, PCR 法, ICCR 法	97
B.3.1	CR 法について	97
B.3.2	ICCR 法について	99

付録 C 発展編の完成部分 (有限体積法, その他)	101
C.1 軸対称問題の数値計算法 (有限体積近似)	101
C.2 2次元円板領域での数値計算法	103
C.3 円柱領域での数値計算法	104
C.4 移流拡散方程式の数値計算法	105
C.4.1 1次の風上差分	105
C.4.2 3次の風上差分	105
C.5 非線形拡散方程式の離散化 (差分法)	106
付録 D 並列化編の完成部分	109
D.1 ADI 法	109
D.1.1 並列化された ADI 法を用いたプログラムの例	110

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を発見するために重要です。数理モデルの妥当性等を調べる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します。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ます [8]。

- (1) モデル方程式に対する理解：方程式の持つ1つ1つの項の意味をしっかりと理解する
- (2) モデル方程式を数値的に解く：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なのか?この問題に対する明確な回答はまだないと思われる。一つの回答は、解析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かつ定量的に比較することではないか。

(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のだが、このときモデルの妥当性や数値計算結果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、次のようなことではないだろうか²：

- (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) 数値計算法の特徴を知ること(理論+実践)
- (3) 数学的な理解(例えば、分岐理論的理解)

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と³：

¹この解説書では(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、数値計算結果から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。

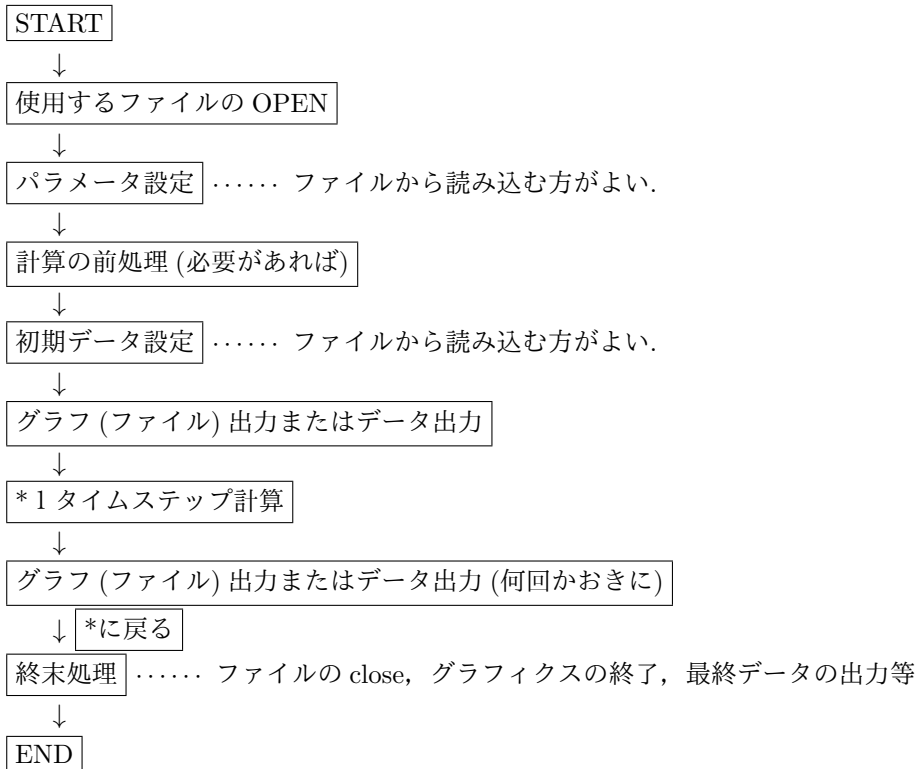
^a力学的理解や分岐理論的理解を用いることも重要

^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

数値計算プログラムは大体次のようなあらすじで書くことができます。

— 数値計算プログラムの流れ —



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、面倒でもプログラムにドキュメント書いておくことが重要です。何年すると後から何をしているのかわからなくなります。

1.3 入門編の目標

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：

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

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)[30]

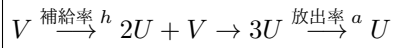
$$\begin{cases} \frac{\partial u}{\partial t} = du_{xx} + \frac{1}{\varepsilon} (u(1-u)(u-a) - v), \\ \frac{\partial v}{\partial t} = u - \gamma v. \end{cases} \quad (1.2)$$

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 [28, 29, 30]

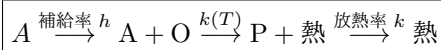
$$\begin{cases} \frac{\partial u}{\partial t} = d_u \Delta u + \frac{1}{\varepsilon} \left(u(1-u) - fw \frac{u-q}{u+q} \right), \\ \frac{\partial w}{\partial t} = d_w \Delta w + u - \gamma w. \end{cases} \quad (1.3)$$

Gray-Scott モデル [29]



$$\begin{cases} \frac{\partial u}{\partial t} = d_u \Delta u - ku + vu^2, \\ \frac{\partial v}{\partial t} = d_v \Delta v + h(1-v) - vu^2. \end{cases} \quad (1.4)$$

発熱反応モデル



$$\begin{cases} \frac{\partial T}{\partial t} = d_T \Delta T + \frac{1}{\varepsilon} \left(-kT + a \exp \left(\frac{T}{1+T/c} \right) \right), \\ \frac{\partial a}{\partial t} = d_a \Delta a + h(a_0 - a) - a \exp \left(\frac{T}{1+T/c} \right). \end{cases} \quad (1.5)$$

ここで, T は温度, a 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

$$\frac{du}{dt} = \mathbf{f}(u) \quad (1.6)$$

と拡散項に対する数値計算

$$\frac{\partial u}{\partial t} = d \Delta u \quad (1.7)$$

の二つの部分に分けて解説する.

反応拡散方程式の数値計算は, 反応項の性質を調べることから始まるので, 常微分方程式系の数値計算法を知っておくことは必要である. また, 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．

$$\begin{cases} \frac{d\mathbf{u}}{dt} &= \mathbf{f}(t, \mathbf{u}), \\ \mathbf{u}(t_0) &= \mathbf{u}_0. \end{cases} \quad (2.1)$$

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

$$\mathbf{u}(t_n) = \mathbf{u}_n, \quad (2.2)$$

と書く．数値計算法を説明するために関数 $\mathbf{g}(x)$ を $x = x_0$ の周りでのテイラー展開を書いておく．

$$\mathbf{g}(x) = \mathbf{g}(x_0) + \frac{d}{dx}\mathbf{g}(x_0)(x - x_0) + \frac{1}{2} \frac{d^2}{dx^2}\mathbf{g}(x_0)(x - x_0)^2 + \frac{1}{3!} \frac{d^3}{dx^3}\mathbf{g}(x_0)(x - x_0)^3 + O((x - x_0)^4). \quad (2.3)$$

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

$$\mathbf{u}_0 = \mathbf{u}(t_0), \quad (2.4)$$

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \Delta t \mathbf{f}(t_n, \mathbf{u}_n). \quad (2.5)$$

このスキームは $O(\Delta t)$ の誤差を持つ．

簡単な証明

$\mathbf{u}(t + \Delta t)$ を t のまわりでテイラー展開すると¹

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

故に

$$\frac{d}{dt}\mathbf{u}(t) = \frac{\mathbf{u}(t + \Delta t) - \mathbf{u}(t)}{\Delta t} - \frac{1}{2} \frac{d^2}{dt^2}\mathbf{u}(t)\Delta t + O(\Delta t^2). \quad (2.6)$$

2.2 Runge-Kutta 法 [10]

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

$$\mathbf{u}(t + \Delta t) = \mathbf{u}(t) + \Delta t \Phi(t, \mathbf{u}(t)) \quad (2.7)$$

という形式を仮定する．このとき

$$\Phi(t, \mathbf{u}(t)) = \alpha k_1 + \beta k_2, \quad (2.8)$$

$$k_1 = \mathbf{f}(t, \mathbf{u}(t)), \quad (2.9)$$

$$k_2 = \mathbf{f}(t + p\Delta t, \mathbf{u}(t) + q\Delta t k_1) \quad (2.10)$$

¹(2.3) において $x = t + \Delta t$, $x_0 = t$ して代入すると求めることができる．

修正 Euler 法 (通常版)

$$\mathbf{u}_0 = \mathbf{u}(t_0), \quad (2.20)$$

$$\mathbf{k} = \mathbf{u}_n + \frac{\Delta t}{2} \mathbf{f}(t_n, \mathbf{u}_n), \quad (2.21)$$

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \Delta t \mathbf{f}\left(t_n + \frac{\Delta t}{2}, \mathbf{k}\right). \quad (2.22)$$

このスキームは $O(\Delta t^2)$ の誤差を持つ。また,

$$\alpha = \beta = \frac{1}{2}, \quad p = q = 1 \quad (2.23)$$

とおくと, Heun 法と呼ばれる。

Heun 法

$$\mathbf{u}_0 = \mathbf{u}(t_0), \quad (2.24)$$

$$\mathbf{k}_1 = \mathbf{f}(\mathbf{u}_n, t_n), \quad (2.25)$$

$$\mathbf{k}_2 = \mathbf{f}(t_n + \Delta t, \mathbf{u}_n + \Delta t \mathbf{k}_1), \quad (2.26)$$

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \frac{\Delta t}{2} (\mathbf{k}_1 + \mathbf{k}_2). \quad (2.27)$$

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

4 次の Runge-Kutta 法

$$\mathbf{u}_0 = \mathbf{u}(t_0), \quad (2.28)$$

$$\mathbf{k}_1 = \mathbf{f}(t_n, \mathbf{u}_n), \quad (2.29)$$

$$\mathbf{k}_2 = \mathbf{f}\left(t_n + \frac{1}{2}\Delta t, \mathbf{u}_n + \frac{1}{2}\Delta t \mathbf{k}_1\right), \quad (2.30)$$

$$\mathbf{k}_3 = \mathbf{f}\left(t_n + \frac{1}{2}\Delta t, \mathbf{u}_n + \frac{1}{2}\Delta t \mathbf{k}_2\right), \quad (2.31)$$

$$\mathbf{k}_4 = \mathbf{f}(t_n + \Delta t, \mathbf{u}_n + \Delta t \mathbf{k}_3), \quad (2.32)$$

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \frac{\Delta t}{6} (\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4). \quad (2.33)$$

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

ロジスティック方程式 [30]

$$\frac{du}{dt} = au(1-u), \quad u(0) = u_0. \quad (2.34)$$

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³。

(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 (2.34) の解を求めなさい。

2.3.2 問題 2

興奮現象モデル

$$\begin{cases} \frac{du}{dt} = \frac{1}{\varepsilon}(u(1-u)(u-a) - v) \equiv \frac{1}{\varepsilon}f(u, v), \\ \frac{dv}{dt} = u - \gamma v \equiv g(u, v), \\ (u, v)(0) = (u_0, v_0). \end{cases} \quad t > 0, \quad (2.35)$$

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 $f(\bar{u}, \bar{v}) = 0, g(\bar{u}, \bar{v}) = 0$ 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

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]

$$\begin{cases} \frac{d^2x}{dt^2} = -\frac{x}{(x^2 + y^2)^{3/2}}, \\ \frac{d^2y}{dt^2} = -\frac{y}{(x^2 + y^2)^{3/2}}, \\ x(0) = 1 - e, \quad \dot{x}(0) = 0, \quad y(0) = 0, \quad \dot{y}(0) = \sqrt{(1+e)/(1-e)}, \quad e = 0.9. \end{cases} \quad t > 0, \quad (2.36)$$

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 (2.36) を解き, (x, y) 平面に表示しなさい。

³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）

「拡散方程式の数値計算は行列計算」と言っても過言ではありません.. いかにも速く精度よく連立一次方程式を計算することが重要になってきます。

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

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

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

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

^aこの符号の意味をしっかりと説明する

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 1次元初期境界値問題 [3, 4]

$$\frac{\partial u}{\partial t} = d \frac{\partial^2 u}{\partial x^2}, \quad 0 < x < L, \quad t > 0 \quad (3.3)$$

ただし, d は拡散係数であり, 正定数である.

初期条件¹ :

$$u(x, 0) = u_0(x). \quad (3.4)$$

Dirichlet 境界条件

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

Neumann 境界条件

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

周期境界条件

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

方程式の離散化

$$\frac{\partial u}{\partial t} = \frac{u^{n+1} - u^n}{\Delta t} + O(\Delta t). \quad (3.8)$$

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} + O(\Delta x^2). \quad (3.9)$$

(2.5) 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 x のまわりでテーラー展開する² :

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

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

(3.10)+(3.11) より

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

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

従って, 離散化誤差は $O(\Delta x^2)$ となる.

¹ $u_0(x)$ は連続関数であれば特に問題ない

²(2.3) において $x = x + \Delta x, x_0 = x$ して代入すると求めることができる.

境界条件の離散化

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

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

$$\text{周期境界条件} \quad u_0^n = u_N^n, \quad u_{-1}^n = u_{N-1}^n. \quad (3.16)$$

^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。

3.2.1 陽解法

陽解法公式

$$u_i^{n+1} = dr u_{i-1}^n + (1 - 2dr)u_i^n + dr u_{i+1}^n, \quad 0 \leq i \leq N. \quad (3.17)$$

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

$$\lambda_k = 1 - 4dr \sin^2\left(\frac{k\Delta x\pi}{2}\right) \quad (3.18)$$

から

$$|\lambda_k| \leq 1 \Leftrightarrow d \frac{\Delta t}{\Delta x^2} \leq \frac{1}{2}. \quad (3.19)$$

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³。

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。

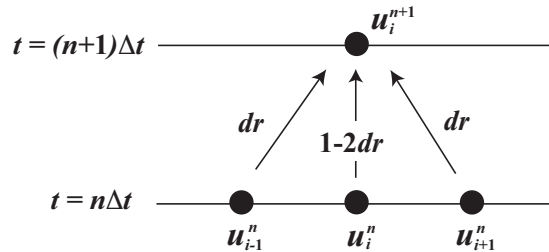


図 3.1: 陽解法の安定性の直感的理解

陽解法の誤差解析

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

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

を (3.3) に代入すると

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

³注意: 理論的には $d \frac{\Delta t}{\Delta x^2} \leq \frac{1}{2}$ 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りますので、 $d \frac{\Delta t}{\Delta x^2} \leq \frac{1}{6}$ 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,

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial t} d \frac{\partial^2 u}{\partial x^2} = d^2 \frac{\partial^4 u}{\partial x^4} \quad (3.23)$$

より

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

故に

$$d \frac{\Delta t}{\Delta x^2} = \frac{1}{6} \quad (3.25)$$

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.

3.2.2 陰解法

陰解法公式

$$u_i^{n+1} - u_i^n = d\theta(ru_{i-1}^{n+1} - 2ru_i^{n+1} + ru_{i+1}^{n+1}) + d(1-\theta)(ru_{i-1}^n - 2ru_i^n + ru_{i+1}^n), \quad 0 \leq i \leq N. \quad (3.26)$$

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.

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

から, $0 < \theta < 1/2$ ならば条件安定, $1/2 \leq \theta \leq 1$ 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.

$$A\mathbf{x} = \mathbf{p}. \quad (3.28)$$

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.

3.2.2.1 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

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

となる.

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

と A を LU 分解し

$$L\mathbf{y} = \mathbf{p},$$

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

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];
}
```

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

```
y[1] = p[1]/L[1][1];
for (i = 2; i <= N; i++)
{
    y[i] = (p[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

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

このとき A を LU 分解すると行列 L, U は次のようになる.

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

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 $Ly = p$, $Ux = y$ の解法アルゴリズム

```

y[1] = p[1]/L[1][1];
for (i = 2; i <= N-1; i++)
{
    y[i] = (p[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] = (p[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];
}

```

3.2.3 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。

反復改良法

```

A を LU 分解する :  $A = LU$ ;
 $Ly = b, Ux = y$  を解いて近似解  $x = x^{(0)}$  を求める;
 $r^{(0)} := b - Ax^{(0)}$ ;
(反復改良)
Form := 1, 2, ... until (終了条件) do
begin
     $Ly = r^{(m-1)}, Uz = y$  を解いて修正量  $z = z^{(m)}$  を求める;
     $x^{(m)} := x^{(m-1)} + z^{(m)}$ ;  $r^{(m)} := b - Ax^{(m)}$ 
end

```

終了条件は、 r_i の丸め誤差限界の評価値を採用する。

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

として、 $|r_i| \leq \delta r_i$ ($i = 1, \dots, N$) 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⁴。といっても $\|r^{(m)}\| < 10^{-8}$ で十分だと思う。

3.2.3.1 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。

⁴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。

SSI 法

$$u_i^{n+1} - u_i^n = -d \frac{\Delta t}{\Delta x^2} \left(2u_i^{n+1} - \left(\frac{3}{2}u_{i+1}^n + u_i^n + \frac{3}{2}u_{i-1}^n \right) + \left(\frac{1}{2}u_{i+1}^{n-1} + u_i^{n-1} + \frac{1}{2}u_{i-1}^{n-1} \right) \right). \quad (3.34)$$

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⁵。

3.2.4 問題

問題1. 熱方程式の初期値境界値問題

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

(3.35) の理論解は

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

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。

3.3 2次元長方形領域における拡散方程式の数値計算

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

3.3.1 陽解法 [4]

陽解法公式

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

陽解法の安定性条件は

$$d_x \frac{\Delta t}{\Delta x^2} + d_y \frac{\Delta t}{\Delta y^2} \leq \frac{1}{2} \quad (3.38)$$

となります。従って、1次元問題より Δt を小さく取る必要があります。実用上は使えないと思うかもしれませんが、実際には使うことが時々あります。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) の条件を満足することがあるからです。

⁵一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。)

3.3.2 陰解法 [4]

陰解法公式

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

1次元問題と同様にやはり安定な数値計算法であるが⁶、次のような連立一次方程式を計算しなければならない：

$$A\mathbf{u}^{n+1} = \mathbf{u}^n. \quad (3.40)$$

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う、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⁶。CG 法等を用いた数値計算法は発展編において解説する予定です。

3.3.3 ADI 法 [5]

ADI 法

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

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  $k := 1, 2, \dots$  until  $times$  do
  begin
    For  $j := 0, 1, 2, \dots$  until  $N_y$  do
      begin
         $A_x \mathbf{u}_j^{n+1} = B_j \mathbf{u}_j^n$ 
      end
    For  $i := 0, 1, 2, \dots$  until  $N_x$  do
      begin
         $A_y \mathbf{u}_i^{n+1} = B_i \mathbf{u}_i^n$ 
      end
    end
  end

```

⁶ 今回の講義では解説しない。

3.3.4 2次元 SSI 法 [5]

2次元 SSI 法

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

この解法がどこまで有効なのか使ったことがないのでわかりません。一度試してみたい気がする。

3.4 3次元直方体領域での数値計算法

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

3.4.1 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。

陽解法公式

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

陽解法の安定性条件は

$$d_x \frac{\Delta t}{\Delta x^2} + d_y \frac{\Delta t}{\Delta y^2} + d_z \frac{\Delta t}{\Delta z^2} \leq \frac{1}{2} \quad (3.43)$$

となります。従って、2次元問題より Δt を小さく取る必要が生じますが、2次元問題と同様の理由で使うことが多いです。

3.4.2 ADI 法 [5]

無条件安定なので安心して使うことができる!!

3. 周期境界条件

$$\begin{cases} \mathbf{u}(t, 0) = \mathbf{u}(t, L), \\ \frac{\partial \mathbf{u}}{\partial x}(t, 0) = \frac{\partial \mathbf{u}}{\partial x}(t, L). \end{cases} \quad (4.6)$$

- 物理でよく使う境界条件 (境界に影響しないパターンダイナミクスを考えるとときに有効).
- 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)

$$\begin{cases} \frac{\partial \mathbf{u}}{\partial x}(t, 0) = -\alpha_1 (\mathbf{a} - \beta_1 \mathbf{u}(t, 0)), \\ \frac{\partial \mathbf{u}}{\partial x}(t, L) = \alpha_2 (\mathbf{b} - \beta_2 \mathbf{u}(t, L)). \end{cases} \quad (4.7)$$

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

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

を

初期条件: $u(0, x) = u_0(x)$, $v(0, x) = v_0(x)$ および 境界条件: (4.4)-(4.7) のいずれか.
のものとして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: $\Delta x = \frac{L}{N}$, $x_i = i \times \Delta x$ ($i = 1, 2, \dots, N$), $t_n = n \times \Delta t$ (適切な値)

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

v についても同様である. 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

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

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

$$\implies \begin{cases} -rd_u u_{i-1}^{n+1} + (1 + 2rd_u)u_i^{n+1} - rd_u u_{i+1}^{n+1} = u_i^n + \Delta t f(u_i^n, v_i^n), \\ -rd_v v_{i-1}^{n+1} + (1 + 2rd_v)v_i^{n+1} - rd_v v_{i+1}^{n+1} = v_i^n + \Delta t g(u_i^n, v_i^n) \end{cases} \quad (4.11)$$

となる.

$n = 0$ のとき

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

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

$$\begin{cases} u(t, 0) = a_u, & u(t, L) = b_u, \\ v(t, 0) = a_v, & v(t, L) = b_v. \end{cases} \implies \begin{cases} u_0^n = a_u, & u_N^n = b_u, \\ v_0^n = a_v, & v_N^n = b_v. \end{cases} \quad (4.13)$$

2. Neumann 条件 (4.5) から

$$\begin{cases} \frac{\partial u}{\partial x}(t, 0) = a_u, \\ \frac{\partial u}{\partial x}(t, N) = b_u. \end{cases} \implies \begin{cases} \frac{u_1^n - u_{-1}^n}{2\Delta x} + O(\Delta x^2) = a_u, \\ \frac{u_{N+1}^n - u_{N-1}^n}{2\Delta x} + O(\Delta x^2) = b_u. \end{cases}$$

従って

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

と離散化することができる. v についても同様に

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

とできる.

3. 周期境界条件 (4.6) より

$$u(t, 0) = u(t, L) \implies u_0^n = u_N^n. \quad (4.16)$$

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

\Rightarrow

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

従って

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

となる。以上をまとめると

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

となる。 v についても同様に

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

となる。

4. 第3種境界条件

$$\begin{aligned} & \begin{cases} \frac{\partial u}{\partial x}(t, 0) = \alpha_{u1} (a_u - \beta_{u1}u(t, 0)), \\ \frac{\partial u}{\partial x}(t, L) = \alpha_{u2} (b_u - \beta_{u2}u(t, L)). \end{cases} \\ \Rightarrow & \begin{cases} \frac{u_1^n - u_{-1}^n}{2\Delta x} + O(\Delta x^2) = \alpha_{u1} (a_u - \beta_{u1}u_0^n), \\ \frac{u_{N+1}^n - u_{N-1}^n}{2\Delta x} + O(\Delta x^2) = \alpha_{u2} (b_u - \beta_{u2}u_N^n). \end{cases} \end{aligned}$$

従って

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

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. $i = 1, \dots, N-1$ 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:

$$\begin{pmatrix} 1 + 2rd_u & -rd_u & 0 & 0 & 0 & \dots & 0 \\ -rd_u & 1 + 2rd_u & -rd_u & 0 & 0 & \dots & 0 \\ 0 & -rd_u & 1 + 2rd_u & -rd_u & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & -rd_u & 1 + 2rd_u \end{pmatrix} \begin{pmatrix} u_1^{n+1} \\ u_2^{n+1} \\ \vdots \\ u_{N-2}^{n+1} \\ u_{N-1}^{n+1} \end{pmatrix} = \begin{pmatrix} u_1^n + \Delta t f(u_1^n, v_1^n) + rd_u a_u \\ \vdots \\ u_i^n + \Delta t f(u_i^n, v_i^n) \\ \vdots \\ u_{N-1}^n + \Delta t f(u_{N-1}^n, v_{N-1}^n) + rd_u b_u \end{pmatrix}. \quad (4.20)$$

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} ($0 \leq i \leq N-1$) において, (4.11) を解けばよい.
 (4.17) より, $i=0$ のとき

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

となる.

(4.17) より, $i=N-1$ のとき

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

となる. $i=1, \dots, N-2$ 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:

$$\begin{aligned} & \begin{pmatrix} 1+2rd_u & -rd_u & 0 & 0 & \cdots & 0 & -rd_u \\ -rd_u & 1+2rd_u & -rd_u & 0 & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & -rd_u & 1+2rd_u & -rd_u \\ -rd_u & 0 & \cdots & 0 & 0 & -rd_u & 1+2rd_u \end{pmatrix} \begin{pmatrix} u_0^{n+1} \\ u_1^{n+1} \\ \vdots \\ \vdots \\ u_{N-2}^{n+1} \\ u_{N-1}^{n+1} \end{pmatrix} \\ & = \begin{pmatrix} u_0^n + \Delta t f(u_0^n, v_0^n) \\ \vdots \\ u_i^n + \Delta t f(u_i^n, v_i^n) \\ \vdots \\ \vdots \\ u_{N-1}^n + \Delta t f(u_{N-1}^n, v_{N-1}^n) \end{pmatrix}. \end{aligned} \quad (4.26)$$

v についても同様である.

4.1.3.4 問題

神経繊維上の電位の伝播モデル (FitzHugh-Nagumo 方程式)[30]

$$\begin{cases} \frac{\partial u}{\partial t} = d_u u_{xx} + \frac{1}{\varepsilon} (u(1-u)(u-a) - v), & t > 0, 0 < x < L \\ \frac{\partial v}{\partial t} = u - \gamma v. \end{cases} \quad (4.27)$$

$\gamma = 1.0, \varepsilon = 0.001, a = 0.125, d_u = 1.0, d_v = 0.1, L = 20.0$ に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, $\Delta t \rightarrow 0, \Delta x \rightarrow 0 (\Delta t / \Delta x^2 \text{は一定})$ 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：

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

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

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

と書く。

初期条件は

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

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、 $u(t_n, x_i, y_j) = u_{i,j}^n$ 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：

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

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：

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

- 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

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

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

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

4.2.3 初期条件の離散化

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

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：

$$\frac{\partial u}{\partial n} = \frac{\partial v}{\partial n} = 0, \quad t > 0, \quad \mathbf{x} \in \partial\Omega \quad (4.34)$$

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

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

と書ける。

このとき、境界条件の離散化は

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

となる。(4.36) より

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

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

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

となり, その離散化は

$$\begin{cases} u(t, x_0, y_j) = u(t, x_{Nx}, y_j) \implies u_{0,j}^n = u_{Nx,j}^n, & (0 \leq j \leq Ny), \\ u(t, x_i, y_0) = u(t, x_i, y_{Ny}) \implies u_{i,0}^n = u_{i,Ny}^n, & (0 \leq i \leq Nx), \\ u(t, x_{-1}, y_j) = u(t, x_{Nx-1}, y_j) \implies u_{-1,j}^n = u_{Nx-1,j}^n, & (0 \leq j \leq Ny), \\ u(t, x_i, y_{-1}) = u(t, x_i, y_{Ny-1}) \implies u_{i,-1}^n = u_{i,Ny-1}^n, & (0 \leq i \leq Nx) \end{cases} \quad (4.39)$$

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

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

2 段目

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

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

$$\begin{aligned} & -d_u r_x u_{i+1,j}^{n+\frac{1}{2}} + (1 + 2d_u r_x) u_{i,j}^{n+\frac{1}{2}} - d_u r_x u_{i-1,j}^{n+\frac{1}{2}} \\ &= u_{i,j}^n + d_u r_y (u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n) + \frac{\Delta t}{2} f(u_{i,j}^n, v_{i,j}^n) \end{aligned}$$

と書ける. ここで

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

とすると, この式は各 j に対して

$$A u_j^{n+\frac{1}{2}} = b_j, \quad 0 \leq j \leq N_y \quad (4.40)$$

と書くことができる連立一次方程式を解く問題に帰着される. ただし, 斉次 Neumann 境界条件の離散化 (4.37) より, b_j の各要素は

$$\mathbf{b}_0 = \begin{cases} b_0 = \frac{1}{2} \left(u_{0,0}^n + d_u r_y (2u_{0,1}^n - 2u_{0,0}^n) + \frac{\Delta t}{2} f(u_{0,0}^n, v_{0,0}^n) \right), \\ b_i = \left(u_{i,0}^n + d_u r_y (2u_{i,1}^n - 2u_{i,0}^n) + \frac{\Delta t}{2} f(u_{i,0}^n, v_{i,0}^n) \right), \quad (1 \leq i \leq N_x - 1), \\ b_{N_x} = \frac{1}{2} \left(u_{N_x,0}^n + d_u r_y (2u_{N_x,1}^n - 2u_{N_x,0}^n) + \frac{\Delta t}{2} f(u_{N_x,0}^n, v_{N_x,0}^n) \right), \end{cases} \quad (4.41)$$

$$\mathbf{b}_j = \begin{cases} b_0 = \frac{1}{2} \left(u_{0,j}^n + d_u r_y (u_{0,j+1}^n - 2u_{0,j}^n + u_{0,j-1}^n) + \frac{\Delta t}{2} f(u_{0,j}^n, v_{0,j}^n) \right), \\ b_i = \left(u_{i,j}^n + d_u r_y (u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n) + \frac{\Delta t}{2} f(u_{i,j}^n, v_{i,j}^n) \right), \quad (1 \leq i \leq N_x - 1), \\ b_{N_x} = \frac{1}{2} \left(u_{N_x,j}^n + d_u r_y (u_{N_x,j+1}^n - 2u_{N_x,j}^n + u_{N_x,j-1}^n) + \frac{\Delta t}{2} f(u_{N_x,j}^n, v_{N_x,j}^n) \right), \end{cases} \quad (4.42)$$

$$\mathbf{b}_{N_y} = \begin{cases} b_0 = \frac{1}{2} \left(u_{0,N_y}^n + d_u r_y (2u_{0,N_y-1}^n - 2u_{0,N_y}^n) + \frac{\Delta t}{2} f(u_{0,N_y}^n, v_{0,N_y}^n) \right), \\ b_i = \left(u_{i,N_y}^n + d_u r_y (2u_{i,N_y-1}^n - 2u_{i,N_y}^n) + \frac{\Delta t}{2} f(u_{i,N_y}^n, v_{i,N_y}^n) \right), \quad (1 \leq i \leq N_x - 1), \\ b_{N_x} = \frac{1}{2} \left(u_{N_x,N_y}^n + d_u r_y (2u_{N_x,N_y-1}^n - 2u_{N_x,N_y}^n) + \frac{\Delta t}{2} f(u_{N_x,N_y}^n, v_{N_x,N_y}^n) \right) \end{cases} \quad (4.43)$$

となる. 各 j に対して, (4.40) を解くと

$$u_{i,j}^{n+\frac{1}{2}}, \quad (0 \leq i \leq N_x, \quad 0 \leq j \leq N_y)$$

が求まる.

2 段目に関しては

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

より

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

とすると、この式は各 i に対して

$$B\mathbf{u}_i^{n+1} = \mathbf{b}_i, \quad (0 \leq i \leq Nx) \quad (4.44)$$

と書くことができる連立一次方程式を解く問題に帰着される。 \mathbf{b}_i の各要素は

$$\mathbf{b}_0 = \begin{cases} b_0 = \frac{1}{2} \left(u_{0,0}^{n+\frac{1}{2}} + d_u r_x (2u_{1,0}^{n+\frac{1}{2}} - 2u_{0,0}^{n+\frac{1}{2}}) + \frac{\Delta t}{2} f(u_{0,0}^{n+\frac{1}{2}}, v_{0,0}^{n+\frac{1}{2}}) \right), \\ b_j = \left(u_{0,j}^{n+\frac{1}{2}} + d_u r_x (2u_{1,j}^{n+\frac{1}{2}} - 2u_{0,j}^{n+\frac{1}{2}}) + \frac{\Delta t}{2} f(u_{0,j}^{n+\frac{1}{2}}, v_{0,j}^{n+\frac{1}{2}}) \right), \quad (1 \leq j \leq N_y - 1), \\ b_{N_y} = \frac{1}{2} \left(u_{0,N_y}^{n+\frac{1}{2}} + d_u r_x (2u_{1,N_y}^{n+\frac{1}{2}} - 2u_{0,N_y}^{n+\frac{1}{2}}) + \frac{\Delta t}{2} f(u_{0,N_y}^{n+\frac{1}{2}}, v_{0,N_y}^{n+\frac{1}{2}}) \right) \end{cases} \quad (4.45)$$

$$\mathbf{b}_i = \begin{cases} b_0 = \frac{1}{2} \left(u_{i,0}^{n+\frac{1}{2}} + d_u r_x (u_{i+1,0}^{n+\frac{1}{2}} - 2u_{i,0}^{n+\frac{1}{2}} + u_{i-1,0}^{n+\frac{1}{2}}) + \frac{\Delta t}{2} f(u_{i,0}^{n+\frac{1}{2}}, v_{i,0}^{n+\frac{1}{2}}) \right), \\ b_j = \left(u_{i,j}^{n+\frac{1}{2}} + d_u r_x (u_{i+1,j}^{n+\frac{1}{2}} - 2u_{i,j}^{n+\frac{1}{2}} + u_{i-1,j}^{n+\frac{1}{2}}) + \frac{\Delta t}{2} f(u_{i,j}^{n+\frac{1}{2}}, v_{i,j}^{n+\frac{1}{2}}) \right), \quad (1 \leq j \leq N_y - 1), \\ b_{N_y} = \frac{1}{2} \left(u_{i,N_y}^{n+\frac{1}{2}} + d_u r_x (u_{i+1,N_y}^{n+\frac{1}{2}} - 2u_{i,N_y}^{n+\frac{1}{2}} + u_{i-1,N_y}^{n+\frac{1}{2}}) + \frac{\Delta t}{2} f(u_{i,N_y}^{n+\frac{1}{2}}, v_{i,N_y}^{n+\frac{1}{2}}) \right) \end{cases} \quad (4.46)$$

$$\mathbf{b}_{N_x} = \begin{cases} b_0 = \frac{1}{2} \left(u_{N_x,0}^{n+\frac{1}{2}} + d_u r_x (2u_{N_x-1,0}^{n+\frac{1}{2}} - 2u_{N_x,0}^{n+\frac{1}{2}}) + \frac{\Delta t}{2} f(u_{N_x,0}^{n+\frac{1}{2}}, v_{N_x,0}^{n+\frac{1}{2}}) \right), \\ b_j = \left(u_{N_x,j}^{n+\frac{1}{2}} + d_u r_x (2u_{N_x-1,j}^{n+\frac{1}{2}} - 2u_{N_x,j}^{n+\frac{1}{2}}) + \frac{\Delta t}{2} f(u_{N_x,j}^{n+\frac{1}{2}}, v_{N_x,j}^{n+\frac{1}{2}}) \right), \quad (1 \leq j \leq N_y - 1), \\ b_{N_y} = \frac{1}{2} \left(u_{N_x,N_y}^{n+\frac{1}{2}} + d_u r_x (2u_{N_x-1,N_y}^{n+\frac{1}{2}} - 2u_{N_x,N_y}^{n+\frac{1}{2}}) + \frac{\Delta t}{2} f(u_{N_x,N_y}^{n+\frac{1}{2}}, v_{N_x,N_y}^{n+\frac{1}{2}}) \right) \end{cases} \quad (4.47)$$

となる。各 i に対して、(4.44) を解くと

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

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 $N_x \times N_x$ の正方行列である。この式は各 j に対して

$$A \mathbf{u}_j^{n+\frac{1}{2}} = \mathbf{b}_j, \quad 0 \leq j \leq N_y - 1 \quad (4.48)$$

という連立一次方程式を解く問題に帰着される。ただし、 \mathbf{b}_j は (4.39) より \mathbf{b}_i の各要素は

$$\mathbf{b}_0 = \begin{cases} b_0 = \left(u_{0,0}^n + d_u r_y (u_{0,1}^n - 2u_{0,0}^n + u_{0,N_y-1}^n) + \frac{\Delta t}{2} f(u_{0,0}^n, v_{0,0}^n) \right), \\ b_i = \left(u_{i,0}^n + d_u r_y (u_{i,1}^n - 2u_{i,0}^n + u_{i,N_y-1}^n) + \frac{\Delta t}{2} f(u_{i,0}^n, v_{i,0}^n) \right), \quad (1 \leq i \leq N_x - 2), \\ b_{N_x-1} = \left(u_{N_x-1,0}^n + d_u r_y (u_{N_x-1,1}^n - 2u_{N_x-1,0}^n + u_{N_x-1,N_y-1}^n) + \frac{\Delta t}{2} f(u_{N_x-1,0}^n, v_{N_x-1,0}^n) \right), \end{cases} \quad (4.49)$$

$$\mathbf{b}_j = \begin{cases} b_0 = \left(u_{0,j}^n + d_u r_y (u_{0,j+1}^n - 2u_{0,j}^n + u_{0,j-1}^n) + \frac{\Delta t}{2} f(u_{0,j}^n, v_{0,j}^n) \right), \\ b_i = \left(u_{i,j}^n + d_u r_y (u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n) + \frac{\Delta t}{2} f(u_{i,j}^n, v_{i,j}^n) \right), \quad (1 \leq i \leq N_x - 2), \\ b_{N_x-1} = \left(u_{N_x-1,j}^n + d_u r_y (u_{N_x-1,j+1}^n - 2u_{N_x-1,j}^n + u_{N_x-1,j-1}^n) + \frac{\Delta t}{2} f(u_{N_x-1,j}^n, v_{N_x-1,j}^n) \right), \end{cases} \quad (4.50)$$

$$\mathbf{b}_{N_y-1} = \begin{cases} b_0 = \left(u_{0,N_y-1}^n + d_u r_y (u_{0,N_y-1}^n - 2u_{0,N_y}^n + u_{0,1}^n) + \frac{\Delta t}{2} f(u_{0,N_y-1}^n, v_{0,N_y-1}^n) \right), \\ b_i = \left(u_{i,N_y}^n + d_u r_y (u_{i,N_y-1}^n - 2u_{i,N_y}^n + u_{i,1}^n) + \frac{\Delta t}{2} f(u_{i,N_y-1}^n, v_{i,N_y-1}^n) \right), \quad (1 \leq i \leq N_x - 2), \\ b_{N_x-1} = \left(u_{N_x-1,N_y}^n + d_u r_y (u_{N_x-1,0}^n - 2u_{N_x-1,N_y-1}^n + u_{N_x-1,N_y-2}^n) + \frac{\Delta t}{2} f(u_{N_x-1,N_y-1}^n, v_{N_x-1,N_y-1}^n) \right) \end{cases} \quad (4.51)$$

となる。ただし、 $0 \leq j \leq N_x - 1$ である。(4.48) を解くと

$$u_{i,j}^{n+\frac{1}{2}}, \quad (0 \leq i \leq N_x - 1, \quad 0 \leq j \leq N_y - 1)$$

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 $N_y \times N_y$ の正方行列である．この式は各 i に対して

$$B\mathbf{u}_i^{n+1} = \mathbf{b}_i, \quad (0 \leq i \leq N_x - 1) \quad (4.52)$$

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

$$\mathbf{b}_0 = \begin{cases} b_0 = \left(u_{0,0}^{n+\frac{1}{2}} + d_u r_x (u_{1,0}^{n+\frac{1}{2}} - 2u_{0,0}^{n+\frac{1}{2}} + u_{N_x-1,0}^{n+\frac{1}{2}}) + \frac{\Delta t}{2} f(u_{0,0}^{n+\frac{1}{2}}, v_{0,0}^{n+\frac{1}{2}}) \right), \\ b_j = \left(u_{0,j}^{n+\frac{1}{2}} + d_u r_x (u_{1,j}^{n+\frac{1}{2}} - 2u_{0,j}^{n+\frac{1}{2}} + u_{N_x-1,j}^{n+\frac{1}{2}}) + \frac{\Delta t}{2} f(u_{0,j}^{n+\frac{1}{2}}, v_{0,j}^{n+\frac{1}{2}}) \right), \quad (1 \leq j \leq N_y - 2), \\ b_{N_y-1} = \left(u_{0,N_y-1}^{n+\frac{1}{2}} + d_u r_x (u_{1,N_y-1}^{n+\frac{1}{2}} - 2u_{0,N_y-1}^{n+\frac{1}{2}} + u_{N_x-1,N_y-1}^{n+\frac{1}{2}}) + \frac{\Delta t}{2} f(u_{0,N_y-1}^{n+\frac{1}{2}}, v_{0,N_y-1}^{n+\frac{1}{2}}) \right) \end{cases} \quad (4.53)$$

$$\mathbf{b}_i = \begin{cases} b_0 = \left(u_{i,0}^{n+\frac{1}{2}} + d_u r_x (u_{i+1,0}^{n+\frac{1}{2}} - 2u_{i,0}^{n+\frac{1}{2}} + u_{i-1,0}^{n+\frac{1}{2}}) + \frac{\Delta t}{2} f(u_{i,0}^{n+\frac{1}{2}}, v_{i,0}^{n+\frac{1}{2}}) \right), \\ b_j = \left(u_{i,j}^{n+\frac{1}{2}} + d_u r_x (u_{i+1,j}^{n+\frac{1}{2}} - 2u_{i,j}^{n+\frac{1}{2}} + u_{i-1,j}^{n+\frac{1}{2}}) + \frac{\Delta t}{2} f(u_{i,j}^{n+\frac{1}{2}}, v_{i,j}^{n+\frac{1}{2}}) \right), \quad (1 \leq j \leq N_y - 2), \\ b_{N_y-1} = \left(u_{i,N_y-1}^{n+\frac{1}{2}} + d_u r_x (u_{i+1,N_y-1}^{n+\frac{1}{2}} - 2u_{i,N_y-1}^{n+\frac{1}{2}} + u_{i-1,N_y-1}^{n+\frac{1}{2}}) + \frac{\Delta t}{2} f(u_{i,N_y-1}^{n+\frac{1}{2}}, v_{i,N_y-1}^{n+\frac{1}{2}}) \right) \end{cases} \quad (4.54)$$

$$\mathbf{b}_{N_x-1} = \begin{cases} b_0 = \left(u_{N_x-1,0}^{n+\frac{1}{2}} + d_u r_x (u_{N_x-2,0}^{n+\frac{1}{2}} - 2u_{N_x-1,0}^{n+\frac{1}{2}} + u_{0,0}^{n+\frac{1}{2}}) + \frac{\Delta t}{2} f(u_{N_x-1,0}^{n+\frac{1}{2}}, v_{N_x-1,0}^{n+\frac{1}{2}}) \right), \\ b_j = \left(u_{N_x-1,j}^{n+\frac{1}{2}} + d_u r_x (u_{N_x-1,j}^{n+\frac{1}{2}} - 2u_{N_x-1,j}^{n+\frac{1}{2}} + u_{0,j}^{n+\frac{1}{2}}) + \frac{\Delta t}{2} f(u_{N_x-1,j}^{n+\frac{1}{2}}, v_{N_x-1,j}^{n+\frac{1}{2}}) \right), \quad (1 \leq j \leq N_y - 2), \\ b_{N_y-1} = \left(u_{N_x-1,N_y-1}^{n+\frac{1}{2}} + d_u r_x (u_{N_x-2,N_y-1}^{n+\frac{1}{2}} - 2u_{N_x-1,N_y-1}^{n+\frac{1}{2}} + u_{0,N_y-1}^{n+\frac{1}{2}}) + \frac{\Delta t}{2} f(u_{N_x-1,N_y-1}^{n+\frac{1}{2}}, v_{N_x-1,N_y-1}^{n+\frac{1}{2}}) \right) \end{cases} \quad (4.55)$$

となる．ただし， $0 \leq j \leq N_y - 1$ である．(4.52) を解くと

$$u_{i,j}^{n+1}, \quad (0 \leq i \leq N_x - 1, \quad 0 \leq j \leq N_y - 1)$$

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

拡張した FitzHugh-Nagumo 方程式 [30]

$$\begin{cases} \frac{\partial u}{\partial t} = d_u \Delta u + \frac{1}{\varepsilon} (u(1-u)(u-a) - v), \\ \frac{\partial v}{\partial t} = d_v \Delta v + u - \gamma v. \end{cases} \quad t > 0, \quad x \in (0, L) \times (0, L) \equiv \Omega \quad (4.56)$$

ただし， $0 < a < 1/2$.

$\gamma = 1.0, \varepsilon = 0.001, a = 0.125, d_u = 1.0, d_v = 0.1, L = 20.0$ に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 数値計算環境の構築と計算結果の可視化法

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, Matlab → 有料で高価なので買えない（使えると便利）。
- AVS → 有料で高価なので買えない！。流体計算の可視化には是非欲しいソフトです。

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]).
- GLSC3D(手作り, 北大電子研の秋山先生らが作った GLSC の 3D 対応, OpenGL ベースで書かれている ([23]). 詳しい解説 PDF マニュアルがある ([24]).

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 **GNUPLOT 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 数値計算と結果の可視化に必要なソフトウェアのインストール

ここでは、手持ちのノート型 PC や自宅の PC で数値計算するための環境構築について説明します。Windows を使っている人は、VMWare Workstation Player をインストールして、VMWare 上で Ubuntu

¹GNUPLOT 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

をインストールすることをお勧めします。VMware 上での Ubuntu のインストール方法はインターネットを調べてください。Mac だと Mac 上に数値計算環境を構築することも可能です。

5.2.1 VMWare

数値計算を行うための OS としては Linux が一番使い勝手がいいです。Windows や MacOS 上で Linux を動かすために VMware というソフトをインストールします。

5.2.1.1 Windows ユーザーの方

非商用利用や個人利用の場合は VMWare Workstation Player が無償で使えます。

<https://www.vmware.com/jp/products/workstation-player/workstation-player-evaluation.html>

からインストールしてください。

5.2.1.2 Mac ユーザーの方 (M1 チップ以降では使えません)

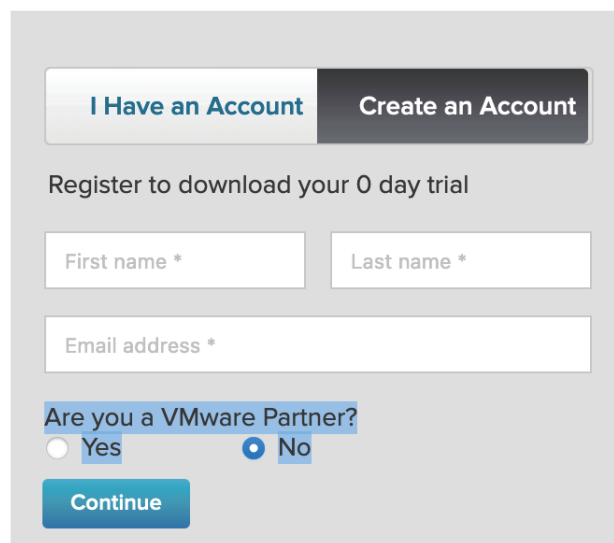
非商用利用や個人利用の場合は VMWare Fusion Player が無償で使えます。

<https://www.vmware.com/products/fusion/fusion-evaluation.html>

から

Register for a Personal User License

を選択してください。以下の画面でユーザー登録してください。その後の手続きを順に実行するとインストールできます。



The image shows a registration form for VMware. At the top, there are two buttons: "I Have an Account" and "Create an Account". Below these is the text "Register to download your 0 day trial". The form contains three input fields: "First name *", "Last name *", and "Email address *". Below the input fields is a question "Are you a VMware Partner?" with two radio buttons: "Yes" and "No". The "No" radio button is selected. At the bottom of the form is a "Continue" button.

図 5.1: ユーザー登録する。Create an Account

5.2.2 macOS 上で数値計算環境を作る

VMWare を使わない場合および M1 チップの Mac を使っている方は、MacOS 上で数値計算をするために開発環境のインストールが必要になります。比較的簡単に構築できるので試してみてください。ここでは、数値計算環境を Mac 上に構築するための 1 例を紹介します。

5.2.2.1 Xcode のインストール

Xcode をインストールについては、以下のサイトを見るとよいです。ただし、Homebrew をインストールすると Xcode の Command Line Tools も一緒にインストールすることができます。

```
http://www.cse.kyoto-su.ac.jp/~oomoto/lecture/program/tips/Xcode_install/index.html#Xcode_install
```

Xcode のパッケージは Mac App Store にあります。Apple ID を作ればダウンロードできます。注意すべき点: Unix のコマンドラインで使う make 等のコマンドが見つからない時は、Xcode の Preference → Download → Command Line Tools をインストールしてください。

5.2.2.2 Homebrew のインストール

こちらのパッケージ管理の方が楽だと思います。Mac Ports を使ったことがない人はこちらをインストールしてください。Homebrew のインストールは

```
https://brew.sh/index_ja
```

を参考してインストールしてください。

5.2.2.3 Mac Ports のインストール

初心者には Homebrew の方が楽に設定できるのでこちらは削除しました。

5.2.2.4 gfortran のインストール

```
https://github.com/fxcoudert/gfortran-for-macOS/releases
```

から OS に合わせたパッケージをダウンロードしてインストールします。

5.2.2.5 gnuplot のインストール

最近では以下のようにすれば簡単にインストールできます。

```
> brew install gnuplot
```

5.2.2.6 ImageMagick のインストール

M1-Mac では X11 をインストールしていないと ImageMagick の display が動かないので注意してください。Mac 上で計算結果の静止画や動画を再生するために使っているので、特に必要ないかもです。

```
> brew install ImageMagick
```

5.2.2.7 ffmpeg のインストール

```
> brew install ffmpeg
```

5.2.2.8 OpenGL 関連のインストール

OpenGL 関連で必要なものをインストールします。Gnuplot で可視化するのであれば不要です。

```
/opt/X11/include/GL
```

の中に、freeglut.h osmesa.h がインストールされていない場合は以下の方法でインストールしてください。

```
> brew install mesa  
> brew install mesa-glu  
> brew install glw  
> brew install freeglut
```

5.2.3 Linux で必要なソフトのインストール

5.2.3.1 Gnuplot のインストール

Linux(AlmaLinux 8.x) の場合：
sudo が実行できないときは root (Super User) になってください。以下の方法でインストールできます。

```
> sudo dnf install gnuplot
```

5.2.3.2 ImageMagick のインストール

Linux(AlmaLinux 8.x) の場合：
sudo が実行できないときは root (Super User) になってください。

```
> sudo dnf install epel-release
> sudo dnf config-manager --set-enabled powertools
> sudo dnf search ImageMagick
とすると例えば
ImageMagick-devel.x86_64 : Library links and header files for ImageMagick app
                           : development
とでてくるので
# sudo dnf install ImageMagick ImageMagick-devel
とすればインストールできる。
```

5.2.3.3 ffmpeg のインストール

Linux(AlmaLinux 8.x) の場合 :

sudo が実行できないときは root (Super User) になってください。以下の方法が便利です。

```
> sudo dnf install epel-release
> sudo dnf config-manager --set-enabled powertools
> sudo dnf install -y https://download.fedoraproject.org/\
  pub/epel/epel-release-latest-8.noarch.rpm
> sudo localinstall -y --nogpgcheck https://download1.rpmfusion.org/\
  free/el/rpmfusion-free-release-8.noarch.rpm
> sudo dnf install -y ffmpeg
```

5.2.3.4 OpenGL 関連のインストール

sudo が実行できないときは root (Super User) になってください。OpenGL 関連で必要なものをインストールします。

```
/use/include/GL
```

の中に、freeglut.h osmesa.h がインストールされていない場合は以下の方法でインストールしてください。

```
> sudo yum install mesa-libOSMesa-devel.x86_64
> sudo yum install freeglut-devel.x86_64su
```

5.3 必要になるかもしれないソフトウェアのインストール

5.3.1 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。Linux 上でのインストールは次のように行う。

³計算しながら結果を表示するためなら GNU PLOT で十分である。

```
$ 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。
```

5.3.2 vlc のインストール

動画再生で困った場合は vlc をインストールすれば、ほぼ解決するので一応解説しておく。
Mac OS X の場合：Mac 用にバイナリーファイルが用意されているので、以下のサイトからインストールしてください。

```
http://www.videolan.org/vlc/
```

Linux(AlmaLinux 8.5) の場合：次の手順に従ってインストールしてください。

```
$ sudo dnf install epel-release rpmfusion-free-release
$ sudo dnf install --nogpgcheck https://dl.fedoraproject.org/\
  pub/epel/epel-release-latest-8.noarch.rpm
$ sudo dnf install --nogpgcheck https://mirrors.rpmfusion.org/\
  free/el/rpmfusion-free-release-8.noarch.rpm
$ sudo dnf install vlc
```

5.4 GNUPLOT による数値計算結果の可視化 [15]

数値シミュレーション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 の詳しい使い方は書籍として [15] がある他、新潟工科大学の竹野研究室によって最新版のマニュアルが和訳されている (参照：<http://takeno.iee.niit.ac.jp/~foo/gp-jman/>)。

簡単な例を2つ挙げる。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：

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

数値データの形式が $(t, x, u(t, x))$ ならば、


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

5.4.1 一般的な注意

グラフを作成する際には、それらは他人に必要な情報を図示するためのものであることを意識する必要がある。一般的な注意事項として、

1. 各軸、及びグラフが何を表示しているかは明らかか？
2. グラフの表示範囲は、その特徴を説明するために適切なものであるか？
3. 表示される情報量は必要十分か？

といったものが挙げられる。多くの場合、これらに対しては、

1. 各軸、及びグラフのタイトルを設定する。
2. グラフの変動幅が認識できる程度の領域を表示する。
3. 同時に表示する系列を制限する。

といった方法で対処できる。

また、グラフを使用する場合、印刷やプロジェクターによる投影を行うと、往々にしてグラフは劣化し、最悪の場合は必要な情報が読み取れないものとなる。一般的な対応策として、

1. 線を太くする。

```
plot sin(x) with lines linewidth 3
```

などとすればよい。どの程度の太さが必要かは出力サイズにも依存するが、概ね3程度あれば十分な場合が多い。

2. 視認が容易な色で描画する。

```
plot sin(x) with lines linecolor rgb "blue"
```

とすると、青色で表示される。背景色と線色の組み合わせ次第では視認性が極端に低下する場合があることに注意すること。例として、白背景に緑線が挙げられる。これは Ver 4 の Gnuplot で3つ以上のデータを同時にプロットすると現れる組み合わせで、かなり評判が悪い。Ver 5 では標準で異なる線色が使用されるようになっている。

5.4.2 画像ファイルに出力する

通常、Gnuplot を立ち上げると出力先は x11(Linux), qt(Linux), aqua(macOS) などに設定されている。これらは可視化結果を直接画面上に表示するもので、結果を素早く確認できる。可視化結果を画像ファイルとして出力するためには、これらの出力先を切り替える必要がある。

以下に例と各形式に対するコメントを列挙する。

5.4.2.1 PNG 形式

```
set term pngcairo # pngcairo が利用できない場合は set term png とする.  
set output "fig01.png"  
plot 'sol_ODE.dat' using 1:2 with lines
```

ビットマップ形式で良ければ、PNG 形式が無難。png よりも pngcairo の方が高機能で出力もきれいなのでこちらが推奨。

5.4.2.2 EPS 形式

```
set term postscript eps color  
set output "fig01.eps"  
plot 'sol_ODE.dat' using 1:2 with lines
```

ベクター形式の出力。拡大縮小が可能で、やろうと思えば後でフォントを差し替えることも可能。

5.4.2.3 GIF 形式

```
set term gif  
set output "fig01.gif"  
plot 'sol_ODE.dat' using 1:2 with lines
```

256 色までしか表示できないが、JPEG のような劣化は無い。特にこだわりがあるのでなければ PNG でよい。

5.4.2.4 JPEG 形式

```
set term jpeg  
set output "fig01.jpg"  
plot 'sol_ODE.dat' using 1:2 with lines
```

JPEG 形式はファイルサイズを小さく抑えられるというメリットがあるが、境界がぼやけてしまう。グラフのような境界がはっきりとした画像では、そのような劣化は許容しがたいものであり、よほどの理由でもあるのでなければ、グラフの保存に用いるべきではない。

5.4.3 Gnuplot を C 言語から呼び出す [16, 17]

具体的にはパイプ機能を用いて行う。次に [16] に記載されていたサンプルを改良したものを示す。

```

sample_gnuplot01.c

#include <stdio.h>

int main(int argc, char **argv)
{
    /* gnuplotを開く */
    FILE *gpid = popen("gnuplot","w");

    /* gnuplotでグラフを書く */
    for(int i=1; i < 100; i++){
        fprintf(gpid, "set xrange [0:4]\n");
        const double 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 */
    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：

```

gcc -O2 -Wall -Wextra sample_gnuplot01.c -o sample_gnuplot01

./sample_gnuplot01

```

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] を参考のこと。改造の例として、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サンプルプログラムを示す：

```

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

#define N 10

int main()
{
    double u[N+1], new_u[N+1];
    const double dt = 0.0005;
    const double T = 10;
    const double dx = 1. / N;
    const double r = dt / (dx * dx);

    /* gnuplot を開く */
    FILE *gpid = popen("gnuplot -persist", "w");

```

```

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

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

for(int j = 0; j*dt < T; j++){
    /* 熱方程式を陽解法で解く */
    for(int 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 */
fprintf(gpid, "pause 10\n");

/* gnuplot を閉じる */
pclose(gpid);

return 0;
}

```

5.5 Gnuplot を用いた 2 次元データの可視化

2次元矩形領域上のデータを Gnuplot で可視化する方法を解説する。数値計算データが次の形式で与えられているとする：

```
test_data_2d.dat
```

```
<x_1> <y_1> <u(x_1, y_1)>
<x_2> <y_1> <u(x_2, y_1)>
:
<x_N> <y_1> <u(x_N, y_1)>

<x_1> <y_2> <u(x_1, y_2)>
<x_2> <y_2> <u(x_2, y_2)>
:
<x_N> <y_2> <u(x_N, y_2)>

:

<x_1> <y_M> <u(x_1, y_M)>
:
<x_N> <y_M> <u(x_N, y_M)>
```

y の値が変化する際の空行は必須であることに注意せよ。この空行が無いと、`with lines` を指定した際の表示が奇妙なものになる。

この時、gnuplot 内で以下のように実行すると、カラーマップによる表示が得られる：

```
set view map
set size ratio -1
set palette defined(1"#0000ff", 2"#0080ff", 3"#00ffff", 4"#00ff80", 5"#00ff00", \
6"#80ff00", 7"#ffff00", 8"#ff8000", 9"#ff0000")
splot "data" using 1:2:3 with pm3d
```

上のコマンド列は、概ね次のような意味となる：

- 1 行目：視点を x - y 平面を垂直に見下ろす位置に変更
- 2 行目： x 軸と y 軸の比が 1:1 になるように表示
- 3 行目：カラーパレットを最大値を赤、最小値を青で表示するように設定
- 4 行目：`with pm3d` でカラーパレットを用いて表示するように指定

この例では、カラーバーは与えられたデータの最大値、最小値が赤、青となるように自動的に調整される。このため、自己相似的な数値計算結果を可視化すると、見た目には変化が現れない。この挙動が望ましくない場合には、

```
set cbrange [-4:4]
```

として、カラーバーの範囲を固定することができる。

以下の perl スクリプトは 2 次元の数値計算のデータを時間分割して、jpeg イメージを作る gnuplot のスクリプトファイル `plot_image2d` を作るためのものです。使い方は以下のようになります。

```
test_data_2d.dat

% ./data2image2d01 < data.file
% gnuplot plot_image2d
```

data2image2d01

```
#!/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';
# 画像フォーマットの選択
$format = 'jpeg';
open(HA, ">./plot_image2d");
print HA "set term $format\n";
print HA "set view map\n";
print HA "set cbrange [-8.4:8.4]\n";
print HA "set size ratio -1\n";
print HA "set palette defined(1'#0000ff', 2'#0080ff', 3'#00ffff', 4'#00ff88', \
    5'#00ff00', 6'#80ff00', 7'#ffff00',8'#ff8000', 9'#ff0000')\n";
$i = 0; $j = 0; $c = 0;
print HA "set output './$image_dir/fig0000.$format' \n";
print HA "splot './$data_dir/$c' using 1:2:3 with pm3d\n";
open(GA, ">$data_dir/$c");
while(<>){
    $a = $&.$' if /\d/;
    @b = split(/\s+/, $a);
    unless($b[0] == $i){
        $i = $b[0];
        $c += 1;
        close(GA);
        open(GA, ">$data_dir/$c");
        if($c <=9){
            print HA "set output './$image_dir/fig000$c.$format'\n";
        }elsif($c >=10 && $c <= 99){
            print HA "set output './$image_dir/fig00$c.$format'\n";
        }else{
            print HA "set output './$image_dir/fig0$c.$format'\n";
        }
        print HA "splot 'cut/$c' using 1:2:3 with pm3d\n";
    }
    unless($b[1] == $j){
        $j = $b[1];
        print GA "\n";
    }
    print GA "$b[1] $b[2] $b[3] $b[4]\n";
}
close(GA);
print HA "pause -1 'Hit Return Key!' \n";
close(HA);
```

5.5.1 動画作成 その 1

計算機 (Linux, MacOS) 上で前節のようにして作られた時系列 JPEG 画像ファイルをまとめて MPEG4 の動画ファイルにする方法を解説する. ここでは ffmpeg を使って動画を作成します. Test というディレクトリにファイル名 000001.jpg から 000099.jpg が入っているとします. output.mp4 という mpeg4 形式の動画をつくると仮定します. このとき, もっとも簡単な方法は以下の通りです.

Linux, MacOS

```
> ffmpeg -i ./Test/0000%2d.jpg -pix_fmt yuv420p output.mp4
```

オプション (-pix_fmt yuv420p) を付けないと Mac 上で再生できない Mpeg4 動画になります. 動画の再生は次のようにコマンドを入力すればよい.

MacOS

```
> open output.mp4
```

Mac の場合は open コマンドを使うことで, Mpeg4 の動画を再生することが可能なアプリケーションが起動します. ほとんどの場合は Quick Time が起動すると思います.

Linux

ImageMagick がインストールされている場合

```
> animate output.mp4
```

上記の動画では解像度が低くもっと解像度の高い動画を作りたいと考える方は, 秋山さんが作ってくれた次のパラメータファイル (makemov_big, makemov_light) を使って動画作成をするとよい. 使い方は次の通りである.

```
> ./makemov_big Test
```

Test は連番画像が入っているディレクトリであり, Test.mov という画像ができる. 動画は MOV ファイル (QuickTime) 形式となっており, Window と Mac OS 上では問題なく再生できる. また, Linux (Cent OS 6.4) 上では標準的な再生ソフト Dragon Player を使うか vlc を使えば再生できる.

5.5.1.1 パラメータファイル (makemov_big)

```
#!/bin/bash
# Copyright (c) Masakazu Akiyama, 2010/2/12 ALL RIGHTS RESERVED
# 再配布 OK、改良 OK、変更点をメールください。

if [ $# -ne 1 ]
then
echo "Please type 'makemov folder'"
exit
fi

#if [ ! -e /opt/local/bin/ffmpeg ]
#then
#   echo "Please install ffmpeg from macports!!(http://www.macports.org/)"
#   exit
#fi
#if [ ! -e identify ]
#then
```

```

#echo "Please install ImageMagick from macports!!(http://www.macports.org/)"
#exit
#fi

folder=$1
folder=`basename $folder`

#CPU_CORES=2
CPU_CORES=$(/usr/bin/getconf _NPROCESSORS_ONLN)

#judge extension & numfilenum
cd ${folder}
firstfile=`ls -1 | head -1`
cd ..

hoge=$firstfile
foo=${#hoge}
bar=`echo ${hoge} | sed -e 's/\.[0-9a-zA-Z]*$//' | wc -c`
ext=`echo ${hoge} | cut -b ${bar}-${foo}`
numfilenum=`expr $bar - 1`

#GetfileSize
lastfile=`ls -1 "${folder}"/ | tail -1`
width=`identify -format "%w" "${folder}/${lastfile}"`
hight=`identify -format "%h" "${folder}/${lastfile}"`
if [ `expr $width % 2` -eq 1 ]
then
    width=`expr $width + 1`
fi
if [ `expr $hight % 2` -eq 1 ]
then
    hight=`expr $hight + 1`
fi
ffmpeg -y -threads ${CPU_CORES} -i "${folder}"/"%0${numfilenum}d${ext}"
-vcodec mjpeg -qscale 0 -s "${width}"x"${hight}" "${folder}".mov

```

5.5.1.2 パラメータファイル (makemov.light)

```

#!/bin/bash
# Copyright (c) Masakazu Akiyama, 2010/2/12 ALL RIGHTS RESERVED
# 再配布 OK、改良 OK、変更点をメールください。

if [ $# -ne 1 ]
then
echo "Please type 'makemov folder'"
exit
fi

#if [ ! -e /opt/local/bin/ffmpeg ]
#then
#    echo "Please install ffmpeg from macports!!(http://www.macports.org/)"
#    exit
#fi
#if [ ! -e identify ]

```



```

#then
#echo "Please install ImageMagick from macports!!(http://www.macports.org/)"
#exit
#fi

folder=$1
folder=`basename $folder`

#CPU_CORES=2
CPU_CORES=$(/usr/bin/getconf _NPROCESSORS_ONLN)

#judge extension & numfilenum
cd ${folder}
firstfile=`ls -1 | head -1`
cd ..

hoge=$firstfile
foo=${#hoge}
bar=`echo ${hoge} | sed -e 's/\.[0-9a-zA-Z]*$//' | wc -c`
ext=`echo ${hoge} | cut -b ${bar}-${foo}`
numfilenum=`expr $bar - 1`

#GetfileSize
lastfile=`ls -1 "${folder}"/ | tail -1`
width=`identify -format "%w" "${folder}"/"${lastfile}"`
hight=`identify -format "%h" "${folder}"/"${lastfile}"`
if [ `expr $width % 2` -eq 1 ]
then
    width=`expr $width + 1`
fi
if [ `expr $hight % 2` -eq 1 ]
then
    hight=`expr $hight + 1`
fi

ffmpeg -y -threads ${CPU_CORES} -i "${folder}"/"%0${numfilenum}d${ext}"
-vcodec libx264 -f mp4 -s "${width}"x"${hight}" -level 30 -crf 30
-coder 0 -r 30000/1001 -flags +loop -partitions +parti4x4 -me_method dia
-subq 1 -me_range 1 -g 150 -qcomp 0.7 -keyint_min 25 -sc_threshold 0
-i_qfactor 0.71 -b_strategy 0 -qmin 9 -qmax 41 -rc_eq 'blurCplx^(1-qComp)'
-qdiff 4 -i_qfactor 0.714286 -bf 0 -bidir_refine 1 -refs 1 "${folder}".mov

```

5.5.2 動画作成 (MPEG1) 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⁴

Cygwin 上では

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

Fedora8 上では

⁴mpeg2vidcodec は rpm 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_encode 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。

```
% 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な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.mp4
```

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した。

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 OpenGL によるオフスクリーンレンダリング

オフスクリーンレンダリングはバックグラウ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⁵。しかし、OpenGL は 3D グラフィクスのための汎用ライブラリであり、数値計算結果を可視化する目的では設計されていないため、OpenGL は非常に詳細な可視化が可能であるものの、描画のためのほぼ全てをユーザーがプログラムする必要があります。この節では、OpenGL のオフスクリーンレンダリングを用いた可視化プログラムを紹介します。

大事なおまじないは次の行である。

オフスクリーンレンダリング

```
...
#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);
}
```

オフスクリーンでは画像は描写されませんので、画像をファイルにして保存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することを前提とします。

⁵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。

— 画像のファイル出力 —

```
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);
    }
    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 (5.7.2.1) 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 5.5.1 で解説した方法で MPEG 動画に変換することができます。

5.7 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。

5.7.1 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。

————— Makefile_GLSC_sample01 —————

```

#make
LIB = -lglscd -lX11 -lm
DIR = -I/usr/X11/include -I/usr/local/include \
^^I-L/usr/X11/lib -L/usr/local/lib
OBJECT = GLSC_sample01.c
CC = gcc -g
all : main01
main01 : $(OBJECT)
^^I$(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。

run_glsc

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

5.7.2 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と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_ADL.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   /* Δ t*/
#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;
}

```

5.7.2.1 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,
    ~I      -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,
~I 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; */
}

```

5.7.2.2 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);
}

```

5.7.2.3 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) :

```

# ← #で始まる行はコメント

# CC ← C コンパイラを指定

```

```

# default では CC=cc
CC      = gcc

# CFLAGS ← C コンパイラに渡されるオプションを指定
CFLAGS = -O3 -Wall -Wextra

# LDLIBS ← リンクするライブラリを指定
LDLIBS = -lOSMesa -lGLU -lGL -lm

# ALL ← 非標準. ここでは最終生成物を指定
ALL     = heat1d

# all ← よく使われるデフォルトターゲット
# ℓ{ALL} ← 変数を展開
all    : ℓ{ALL}

# clean ← 生成物を消去するためのターゲット
clean  :
^^Irm -f *.o ℓ{ALL}

# heat1d を生成するための依存関係を記述
# heat1d.c は何も書かなくても探しに行く
heat1d : Draw_OSMesa2.o mk_color_full.o

```

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

```

Makefile の書き方は以下を参照のこと：情報は古いですが文法は変わっていないので充分役に立ちます。

https://www.ecoop.net/coop/translated/GNUMake3.77/make_toc.jp.html

5.7.2.4 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/X11/include -L/usr/X11/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

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

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

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

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

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

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

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

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

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

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

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

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

と書ける. 境界条件 $u_0^k = a$ $u_N^k = b$ より

1. $j = 1$ のとき

$$(1+2\theta r)u_1^{k+1} - \theta r u_2^{k+1} = ar + (1-2(1-\theta)r)u_1^k + (1-\theta)r u_2^k. \quad (\text{A.6})$$

2. $j = N-1$ のとき

$$-r\theta u_{N-2}^{k+1} + (1+2\theta r)u_{N-1}^{k+1} = (1-\theta)r u_{N-2}^k + (1-2(1-\theta)r)u_{N-1}^k + rb. \quad (\text{A.7})$$

(A.5)-(A.7) より

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

となり, これは

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

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

とすると, 差分方程式の解は連立一次方程式

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

を解くことによって求められる. ところで

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

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

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.

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

ここで

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

とすると

$$A = I - \theta r M,$$

$$B = I + (1-\theta)r M$$

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

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

と書ける。従って

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

今、 $\rho(B) = \max_{1 \leq i \leq N-1} |\lambda_i|$ (λ_i は B の固有値とする) とおくと

$$\rho(B) < 1 \iff \lim_{k \rightarrow \infty} B^k = 0,$$

$$\rho(B) > 1 \iff \lim_{k \rightarrow \infty} B^k = \infty.$$

差分方程式が安定であるためには (発散しないためには), $\lim_{k \rightarrow \infty} B^k = 0$ 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 \mathbf{x} とすると

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

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。

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

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

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

を得る。ここで,

第二種チェビシェフ多項式について

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

の解は

$$U_j(x) = \sin(j\theta), \quad x = \cos \theta$$

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

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

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

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

(A.9) かつ (A.10) をみたすような θ を決める.

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

故に

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

固有空間 \mathbf{x}^m は

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

(A.11) より, B の固有値は $1 - 4r\sin^2\frac{m\pi}{2N}$, $1 \leq m \leq N-1$. 従って, $-1 < 1 - 4r\sin^2\frac{m\pi}{2N} < 1$ のとき, $\lim_{k \rightarrow \infty} B^k = 0$ 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 $A\mathbf{u}^{k+1} = \mathbf{u}^k$ となる.

ここで M の固有値 λ_m と固有空間 \mathbf{v}_m に対して

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

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

より, A は逆行列 A^{-1} を持つ. 従って

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

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

$$\left| \frac{1}{1 + 4r \sin^2 \frac{m\pi}{2N}} \right| \leq 1$$

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–ニコルソン法の安定性

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

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

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

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

より、 A は逆行列を持つ。従って

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

安定性は $\rho(A^{-1}B) \leq 1$ で決まる。ただし、数値計算上では $\rho(A^{-1}B) < 1$ としなければならない。

M の固有値 λ_m と固有空間 \mathbf{v}_m に対して、

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

従って

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

となるので

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

$A^{-1}B$ の固有値 $\lambda_{A^{-1}B}$ は

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

従って、安定であるためには

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

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

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

について安定性条件を求めよ。

A.3 差分スキームの収束性

差分スキームの安定性 (Stability) についてはこれまで議論してきた。ここでは、差分スキームの適合性 (Consistency)、収束性 (Convergence) について解説する。

定義 1 微分方程式（熱方程式の初期値・境界値問題が）が *Well-posed* である。

$\xLeftrightarrow{\text{def}}$

解が一意的に存在し、パラメータを連続に変化させたとき、解も連続に変化する。

定義 2 差分スキームが適合である。

$\xLeftrightarrow{\text{def}}$

$\Delta t, \Delta x$ を $\downarrow 0$ としたとき、差分方程式が元の微分方程式に収束する。

定義 3 差分スキームが収束する。

$\xLeftrightarrow{\text{def}}$

$\Delta t, \Delta x$ を $\downarrow 0$ としたとき、差分方程式の解が元の微分方程式の解に収束する。

熱方程式の差分スキームの適切性は、差分化の手続きを見れば明らかにわかるが、収束性は全くもって自明ではない。そこで、次の Lax の同等定理が重要となってくる。

定理 1 線形微分方程式の初期値・境界値問題が *Well-posed* であり、差分スキームが適合かつ安定である。

$\xLeftrightarrow{\text{必要十分}}$

差分スキームは収束する。

注意すべき点は、この定理は線形方程式の場合に成立しているということである。非線形方程式の場合は成り立つとは限らないので注意が必要である。非線形問題の場合は、差分スキームが安定であるからといって、正しい数値計算結果（元の微分方程式の解の近似になっている）を得ているという保証がないことを忘れてはならない。数値計算結果がもっともらしいということはどういうことなのだろうかという疑問を持って数値計算をして欲しいものです。

付録B 発展編の完成部分(連立1次方程式の反復解法)

B.1 Jacobi法, Gauss-Seidel法, SOR法

この節では一般の $n \times n$ 正方行列を係数とする連立1次方程式に対する反復計算法を解説します。連立1次方程式

$$A\mathbf{x} = \mathbf{b} \quad (\text{B.1})$$

を考えます。ただし、 $A = (a_{ij}) \in \mathbb{R}^{n \times n}$, $\mathbf{x} = (x_j)$, $\mathbf{b} = (b_i) \in \mathbb{R}^n$ とし、 A は正則と仮定します。ここで適当な初期値

$$\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)})$$

から出発して、反復ベクトル列 $\{\mathbf{x}^{(k)}\}$ を作り、真の解に収束させるを考えます。行列 A に対して適当な正則行列 M を選んで

$$A = M - N$$

と分解します。このとき (B.1) は

$$M\mathbf{x} = N\mathbf{x} + \mathbf{b}$$

と書けます。このとき反復法

$$\mathbf{x}^{(k+1)} = M^{-1}N\mathbf{x}^{(k)} + M^{-1}\mathbf{b} \quad (\text{B.2})$$

を考えます。行列 $M^{-1}N$ の固有値を重複度を込めて $\lambda_i (i \leq i \leq n)$ とすると

$$\max_{1 \leq i \leq n} |\lambda_i| < 1$$

ならば、(B.2) は収束することが期待されますが、その議論は後の節で行います。ここで、行列 A を次のように3つの行列の和に分解します。

$$A = L + D + U$$

ただし

$$D = \begin{pmatrix} a_{11} & 0 & \dots & \dots & \dots & 0 \\ 0 & a_{22} & 0 & \dots & \dots & 0 \\ 0 & 0 & \ddots & \ddots & & \vdots \\ \vdots & \vdots & \ddots & a_{ii} & \ddots & \vdots \\ \vdots & \vdots & & \ddots & \ddots & 0 \\ 0 & 0 & \dots & \dots & 0 & a_{nn} \end{pmatrix}, \quad (\text{B.3})$$

$$L = \begin{pmatrix} 0 & 0 & \dots & \dots & \dots & 0 \\ a_{21} & 0 & 0 & \dots & \dots & 0 \\ a_{31} & a_{32} & \ddots & \ddots & & \vdots \\ \vdots & \vdots & \ddots & 0 & \ddots & \vdots \\ \vdots & \vdots & & \ddots & \ddots & 0 \\ a_{n1} & a_{n2} & \dots & \dots & a_{nn-1} & 0 \end{pmatrix}, U = \begin{pmatrix} 0 & a_{12} & \dots & \dots & a_{1n-1} & a_{1n} \\ 0 & 0 & a_{23} & \dots & \dots & a_{2n} \\ \vdots & 0 & \ddots & \ddots & & \vdots \\ \vdots & \vdots & \ddots & 0 & \ddots & \vdots \\ \vdots & \vdots & & \ddots & \ddots & a_{n-1n} \\ 0 & 0 & \dots & \dots & 0 & 0 \end{pmatrix} \quad (\text{B.4})$$

です。ここで $a_{ii} \neq 0$ ($1 \leq i \leq n$) を仮定します。このとき、 $M = D, N = -L - U$ とするとヤコビ (Jacob) 法と呼ばれる反復法となります。また、 $M = L + D, N = -U$ とするとガウス-ザイデル (Gauss-Seidel) 法と呼ばれる反復法になり

$$M = \frac{1}{\omega}(\omega L + D), N = \frac{1}{\omega}((1 - \omega)D - \omega U)$$

とすると SOR (successive over-relaxation) 法 (逐次過大緩和法) となります。ここで ω は緩和因子とか緩和係数と呼ばれます。

B.1.1 Jacobi 法

Jacobi 法は、 $\mathbf{x}^{(k)} = (x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)})$ を既知とすると

$$\begin{cases} a_{11}x_1^{(k+1)} + a_{12}x_2^{(k)} + \dots + a_{1n}x_n^{(k)} = b_1 \\ a_{21}x_1^{(k)} + a_{22}x_2^{(k+1)} + \dots + a_{2n}x_n^{(k)} = b_2 \\ \vdots \\ a_{n1}x_1^{(k)} + a_{n2}x_2^{(k)} + \dots + a_{nn}x_n^{(k+1)} = b_n \end{cases} \quad (\text{B.5})$$

として、第 1 式から順番に $x_1^{(k+1)}, x_2^{(k+1)}, \dots, x_n^{(k+1)}$ を求めることができます。この計算によって反復列 $\{\mathbf{x}^{(k+1)}\}$ を求め、反復列の収束から近似解を求める方法です。ここで (B.5) を (B.3), (B.4) を用いて表すと

$$D\mathbf{x}^{(k+1)} = -(L + U)\mathbf{x}^{(k)} + \mathbf{b} \quad (\text{B.6})$$

となります。もし D^{-1} が存在するならば

$$\mathbf{x}^{(k+1)} = -D^{-1}(L + U)\mathbf{x}^{(k)} + D^{-1}\mathbf{b} \quad (\text{B.7})$$

と表記できます。

B.1.2 Gauss-Seidel 法

Gauss-Seidel 法は、 $\mathbf{x}^{(k)} = (x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)})$ を既知とすると

$$\begin{cases} a_{11}x_1^{(k+1)} + a_{12}x_2^{(k)} + a_{13}x_3^{(k)} + \dots + a_{1n}x_n^{(k)} = b_1 \\ a_{21}x_1^{(k+1)} + a_{22}x_2^{(k+1)} + a_{23}x_3^{(k)} + \dots + a_{2n}x_n^{(k)} = b_2 \\ a_{31}x_1^{(k+1)} + a_{32}x_2^{(k+1)} + a_{33}x_3^{(k+1)} + \dots + a_{3n}x_n^{(k)} = b_3 \\ \vdots \\ a_{n1}x_1^{(k+1)} + a_{n2}x_2^{(k+1)} + a_{n3}x_3^{(k+1)} + \dots + a_{nn}x_n^{(k+1)} = b_n \end{cases} \quad (\text{B.8})$$

とし、第 1 式から $x_1^{(k+1)}$ を求め第 2 式に代入し、第 2 式から $x_2^{(k+1)}$ を求めます。そして、 $x_1^{(k+1)}, x_2^{(k+1)}$ を第 3 式に代入し、第 3 式から $x_3^{(k+1)}$ を求めます。順次繰り返す、最後に、 $x_1^{(k+1)}, x_2^{(k+1)}, \dots, x_{n-1}^{(k+1)}$ を第 n 式に代入し、 $x_n^{(k+1)}$ を求め、反復列 $\{\mathbf{x}^{(k+1)}\}$ を求め、反復列の収束から近似解を求める方法です。(B.8) を (B.3), (B.4) を用いて表すと

$$(L + D)\mathbf{x}^{(k+1)} + U\mathbf{x}^{(k)} = \mathbf{b} \quad (\text{B.9})$$

となります。 $(L + D)^{-1}$ が存在するならば

$$\mathbf{x}^{(k+1)} = -(L + D)^{-1}(U)\mathbf{x}^{(k)} + (L + D)^{-1}\mathbf{b} \quad (\text{B.10})$$

と書くことができます。Gauss-Seidel 法は Jacobi 法の改良版とみなせ、収束する場合は、多くの場面で Jacobi 法よりも収束が早くなります。

B.1.3 SOR 法

SOR 法は, 既知の $\mathbf{x}^{(k)}$ を用いて

$$D\mathbf{y}^{(k+1)} + L\mathbf{x}^{(k+1)} + U\mathbf{x}^{(k)} = \mathbf{b} \quad (\text{B.11})$$

として, $\mathbf{y}^{(k+1)}$ を作り, 次に, ω をパラメータとして

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \omega(\mathbf{y}^{(k+1)} - \mathbf{x}^{(k)}) \quad (\text{B.12})$$

によって反復列 $\{\mathbf{x}^{(k+1)}\}$ を求め, 反復列の収束から近似解を求める方法です. ω は緩和係数と呼ばれ, 行列 A が収束するためには $0 < \omega < 2$ が必要であることが知られています. もし D^{-1} が存在するならば, (B.11) から

$$\mathbf{y}^{(k+1)} = -D^{-1}L\mathbf{x}^{(k+1)} - D^{-1}U\mathbf{x}^{(k)} + D^{-1}\mathbf{b}$$

となり, これを (B.12) に代入すると

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \omega(-D^{-1}L\mathbf{x}^{(k+1)} - D^{-1}U\mathbf{x}^{(k)} + D^{-1}\mathbf{b} - \mathbf{x}^{(k)})$$

となります. これを整理すると

$$(E + \omega D^{-1}L)\mathbf{x}^{(k+1)} = ((1 - \omega)E - D^{-1}U)\mathbf{x}^{(k)} + \omega D^{-1}\mathbf{b}$$

となります. ただし, $E \in \mathbb{R}^{n \times n}$ は単位行列とします. 両辺に左から D をかけると

$$(D + \omega L)\mathbf{x}^{(k+1)} = ((1 - \omega)D - U)\mathbf{x}^{(k)} + \omega\mathbf{b}$$

となるので, SOR 法では

$$\mathbf{x}^{(k+1)} = (D + \omega L)^{-1}((1 - \omega)D - U)\mathbf{x}^{(k)} + \omega(D + \omega L)^{-1}\mathbf{b} \quad (\text{B.13})$$

において反復列 $\{\mathbf{x}^{(k)}\}$ を求めることができます.

B.1.4 反復法の収束定理

この節では, ヤコビ法, ガウス-ザイデル法, SOR 法の収束定理を示すために, 係数行列 A に関する狭義優対角性の定義を与えます.

定義 4 n 次正方形行列 A が狭義優対角行列であるとは, $A = (a_{ij})_{1 \leq i, j \leq n}$ に対して

$$\sum_{j \neq i}^n |a_{ij}| < |a_{ii}|, \quad i = 1, 2, \dots, n \quad (\text{B.14})$$

を満たすことである.

N 次連立 1 次方程式の係数行列 A が狭義優対角性を持つとき, 次の解の存在定理があります.

定理 2 $A \in \mathbb{R}^{n \times n}$, $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$ とする. このとき, 連立 1 次方程式

$$A\mathbf{x} = \mathbf{b} \quad (\text{B.15})$$

は, A が狭義優対角行列であれば一意解を持つ.

証明 任意の $\mathbf{x} \in \mathbb{R}^n$ に対して

$$\mathbf{y} = A\mathbf{x}$$

とおく. $\|\mathbf{x}\|_\infty = \max_{1 \leq i \leq n} |x_i|$ とする. このとき $|x_i| = \|\mathbf{x}\|_\infty$ となる i を選ぶ. このとき

$$a_{i,i}x_i + \sum_{j=1, j \neq i}^n a_{i,j}x_j = y_i$$

となります.

$$\begin{aligned} \|a_{i,i}\mathbf{x}\|_\infty &= |a_{i,i}|\|\mathbf{x}\|_\infty = |a_{i,i}||x_i| = |y_i - \sum_{j=1, j \neq i}^n a_{i,j}x_j| \\ &\leq |y_i| + \left| \sum_{j=1, j \neq i}^n a_{i,j}x_j \right| \\ &\leq |y_i| + \sum_{j=1, j \neq i}^n |a_{i,j}||x_j| \\ &\leq \|\mathbf{y}\|_\infty + \|\mathbf{x}\|_\infty \sum_{j=1, j \neq i}^n |a_{i,j}| \end{aligned}$$

従って

$$\left(|a_{i,i}| - \sum_{j=1, j \neq i}^n |a_{i,j}| \right) \|\mathbf{x}\|_\infty \leq \|\mathbf{y}\|_\infty$$

を得ます. 狭義優対角性 (B.14) から

$$\left(|a_{i,i}| - \sum_{j=1, j \neq i}^n |a_{i,j}| \right) > 0$$

より $\|\mathbf{y}\|_\infty > 0$ から $\mathbf{y} \neq \mathbf{0}$ となるので, 行列 A は正則行列なる (課題). 証明終

これによって, 狭義優対角行列を係数に持つ連立 1 次方程式は一意解を持つことがわかったので, あとは反復法が一意解に収束することを示せば良いことがわかります. また, 定理 2 から熱方程式の陰解法から導出される連立一次方程式の解が一意に存在することもわかります. ヤコビ法等の反復法の収束定理として次のことが知られています. 次節以降でこの収束定理を証明します.

定理 3 $A \in \mathbb{R}^{n \times n}$, $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$ とする. このとき, 連立 1 次方程式 $A\mathbf{x} = \mathbf{b}$ の係数行列 A が狭義優対角行列であるならば, 任意の初期値 $\mathbf{x}^{(0)} \in \mathbb{R}^n$ に対して, ヤコビ法, ガウス–ザイデル法は $A\mathbf{x} = \mathbf{b}$ の一意解に収束する. また, SOR 法は $0 < \omega < 1$ のとき $A\mathbf{x} = \mathbf{b}$ の一意解に収束する.

B.2 共役勾配法 (Conjugate Gradient method)

行列 A が正値対称行列ならば共役勾配法という反復計算が使える. 空間 2 次元拡散方程式の陰解法や Poisson 方程式の離散化に現れる行列は正値対称行列となっており, 空間 2 次元の数値計算には共役勾配法が有効である.

B.2.1 CG 法について

CG 法とは、共役勾配法 (Conjugate Gradient method) のことで、連立一次方程式 $A\mathbf{x} = \mathbf{b}$ に対する数値解法の 1 つであり、 A は対称正定値行列とする。CG 法の原理は、以下の定理に基づいている。

定理 4 A は n 次対称正定値行列とする。このとき、連立一次方程式 $A\mathbf{x} = \mathbf{b}$ の解 $\tilde{\mathbf{x}}$ は、関数

$$f(\mathbf{x}) = \frac{1}{2}(\mathbf{x}, A\mathbf{x}) - (\mathbf{x}, \mathbf{b})$$

を最小にする。

CG 法は、任意の初期ベクトル \mathbf{x} から始めて、 $f(\mathbf{x})$ の最小点を逐次探索する方法である。今、第 k 番目の近似値 \mathbf{x}_k と探索方向 \mathbf{p}_k が決まったとすると、

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

として、 $f(\mathbf{x}_{k+1})$ が最小になるように α_k を決めればよい。

$$\begin{aligned} f(\mathbf{x}_{k+1}) &= f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \\ &= \frac{1}{2}(\mathbf{x}_k + \alpha_k \mathbf{p}_k, A(\mathbf{x}_k + \alpha_k \mathbf{p}_k)) - (\mathbf{x}_k + \alpha_k \mathbf{p}_k, \mathbf{b}) \\ &\quad \vdots \\ &= f(\mathbf{x}_k) - \alpha_k(\mathbf{p}_k, \mathbf{b} - A\mathbf{x}_k) + \frac{1}{2}\alpha_k^2(\mathbf{p}_k, A\mathbf{p}_k) \end{aligned}$$

従って

$$f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) = \frac{1}{2}\alpha_k^2(\mathbf{p}_k, A\mathbf{p}_k) - \alpha_k(\mathbf{p}_k, \mathbf{r}_k) + f(\mathbf{x}_k) \quad (\text{ただし, } \mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k)$$

となる。 $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$ を α_k の関数と考えて、 $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$ が最小になるように α_k を決める。すなわち

$$\begin{aligned} f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) &= \frac{1}{2}(\mathbf{p}_k, A\mathbf{p}_k) \left\{ \alpha_k^2 - 2\alpha_k \frac{(\mathbf{p}_k, \mathbf{r}_k)}{(\mathbf{p}_k, A\mathbf{p}_k)} \right\} + f(\mathbf{x}_k) \\ &= \frac{1}{2}(\mathbf{p}_k, A\mathbf{p}_k) \left\{ \alpha_k - \frac{(\mathbf{p}_k, \mathbf{r}_k)}{(\mathbf{p}_k, A\mathbf{p}_k)} \right\}^2 + f(\mathbf{x}_k) - \frac{1}{2}(\mathbf{p}_k, A\mathbf{p}_k) \frac{(\mathbf{p}_k, \mathbf{r}_k)^2}{(\mathbf{p}_k, A\mathbf{p}_k)^2} \end{aligned}$$

となるので $\alpha_k = (\mathbf{p}_k, \mathbf{r}_k) / (\mathbf{p}_k, A\mathbf{p}_k)$ のとき、 $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$ は最小になる。

ただし、途中で用いた

$$\mathbf{r}_k = \mathbf{b}_k - A\mathbf{x}_k$$

は第 k 近似 \mathbf{x}_k に対する残差 \mathbf{r}_k である。 $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{p}_k$ であることから

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k$$

と、残差も漸化式で表すことができ、これにより計算できる。次に探索方向を漸化式

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$$

によって決める。ただし、 $\mathbf{p}_0 = \mathbf{r}_0$ とする。探索方向 \mathbf{p}_{k+1} としては、 \mathbf{x}_{k+1} から解 $\tilde{\mathbf{x}} = A^{-1}\mathbf{b}$ を望む方向にしたい。しかし解がわからないので、それは無理である。そこで次善の策として解 $\tilde{\mathbf{x}} = A^{-1}\mathbf{b}$ を \mathbf{x}_{k+1} から見るベクトル $A^{-1}\mathbf{b} - \mathbf{x}_{k+1}$ が $A\mathbf{p}_k$ と直交だという性質を利用し、それと同じようにして \mathbf{p}_{k+1} は $A\mathbf{p}_k$ と直交になるように選ぶことにする。そして、そのように β_k を選ぶとすると

$$(\mathbf{p}_{k+1}, A\mathbf{p}_k) = (\mathbf{r}_{k+1} + \beta_k \mathbf{p}_k, A\mathbf{p}_k) = (\mathbf{r}_{k+1}, A\mathbf{p}_k) + \beta_k(\mathbf{p}_k, A\mathbf{p}_k) = 0$$

となるので、これより

$$\beta_k = -\frac{(\mathbf{r}_{k+1}, A\mathbf{p}_k)}{(\mathbf{p}_k, A\mathbf{p}_k)}$$

となる。以上が CG 法の大筋である。また、CG 法では高々 n 回の反復で正解が求められることが示されている。CG 法の詳しい解説は [?, ?, 7, 8] を参考にしてください。

B.2.2 アルゴリズム

CG 法 (原型版)

適当な初期ベクトル \mathbf{x}_0 を設定する.

$$\left. \begin{aligned} \mathbf{r}_0 &= \mathbf{b} - A\mathbf{x}_0 \\ \mathbf{p}_0 &= \mathbf{r}_0 \end{aligned} \right\} \ast$$

 $k = 0, 1, 2, \dots$ の順に, 以下の手順を繰り返す.

$$\left. \begin{aligned} \alpha_k &= (\mathbf{p}_k, \mathbf{r}_k) / (\mathbf{p}_k, A\mathbf{p}_k) \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k \\ \mathbf{r}_{k+1} &= \mathbf{r}_k - \alpha_k A\mathbf{p}_k \end{aligned} \right\} \ast$$

ここで, もし $\|\mathbf{r}_k\| < \varepsilon \|\mathbf{b}\|$ ならば計算終了. そうでなければ, 以下の手順を続ける.

$$\left. \begin{aligned} \beta_k &= -(\mathbf{r}_{k+1}, A\mathbf{p}_k) / (\mathbf{p}_k, A\mathbf{p}_k) \\ \mathbf{p}_{k+1} &= \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k \quad \ast \\ k &= k + 1 \end{aligned} \right.$$

CG 法 (実用版)

適当な初期ベクトル \mathbf{x}_0 を設定する.

$$\left. \begin{aligned} \mathbf{r}_0 &= \mathbf{b} - A\mathbf{x}_0 \\ \mathbf{p}_0 &= \mathbf{r}_0 \\ \rho &= (\mathbf{r}_0, \mathbf{r}_0) \end{aligned} \right\} \ast$$

 $k = 0, 1, 2, \dots$ の順に, 以下の手順を繰り返す.

$$\left. \begin{aligned} \mathbf{q}_k &= A\mathbf{p}_k \\ \theta &= (\mathbf{p}_k, \mathbf{q}_k) \\ \alpha_k &= \rho / \theta \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k \\ \mathbf{r}_{k+1} &= \mathbf{r}_k - \alpha_k \mathbf{q}_k \end{aligned} \right\} \ast$$

ここで, もし $\|\mathbf{r}_k\| < \varepsilon \|\mathbf{b}\|$ ならば計算終了. そうでなければ, 以下の手順を続ける.

$$\left. \begin{aligned} \mu &= (\mathbf{r}_{k+1}, \mathbf{r}_{k+1}) \\ \beta_k &= \mu / \rho \\ \rho &= \mu \quad (k \text{ を } 1 \text{ つすすめる}) \\ \mathbf{p}_{k+1} &= \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k \quad \ast \\ k &= k + 1 \end{aligned} \right.$$

並列化計算を行う際は, \ast の部分のループの前に並列化構文を挿入して, 並列化を行う.

上記のアルゴリズムより, $(\mathbf{r}_i, \mathbf{r}_j) = 0$ ($i \neq j$) であることから ($\because \mathbf{r}_i$ の直交性より), ある $m \leq n$ で $\mathbf{r}_m = \mathbf{0}$ となることがわかる. $n+1$ 個以上の n 次元の非ゼロベクトルが互いに直交することはないから, 高々 n 回の反復で正解が求められるのである.

B.2.3 原型版と実用版の違いについて

2つの版で大きく異なっているのは、 α_k と β_k の計算の部分である。

$$\text{原型版} \quad \cdots \quad \alpha_k = (\mathbf{p}_k, \mathbf{r}_k) / (\mathbf{p}_k, A\mathbf{p}_k), \quad \beta_k = -(\mathbf{r}_{k+1}, A\mathbf{p}_k) / (\mathbf{p}_k, A\mathbf{p}_k)$$

$$\text{実用版} \quad \cdots \quad \alpha_k = (\mathbf{r}_k, \mathbf{r}_k) / (\mathbf{p}_k, A\mathbf{p}_k), \quad \beta_k = (\mathbf{r}_{k+1}, \mathbf{r}_{k+1}) / (\mathbf{r}_k, \mathbf{r}_k)$$

α_k と β_k については

$$(\mathbf{r}_k, \mathbf{p}_k) = (\mathbf{r}_k, \mathbf{r}_k), \quad (\text{B.16})$$

$$(\mathbf{p}_k, A\mathbf{p}_k) = (\mathbf{r}_k, A\mathbf{p}_k) \quad (\text{B.17})$$

の関係を用いると、原型版から実用版に書き換えられることがわかる。この関係については、次のことからいえる。

(4.1) について、数学的帰納法を用いて示すとして、まず $k=0$ のとき、

$$\mathbf{p}_0 = \mathbf{r}_0 \quad \Rightarrow \quad (\mathbf{r}_0, \mathbf{p}_0) = (\mathbf{r}_0, \mathbf{r}_0)$$

となり、成立する。

次に $k=n$ の時、(4.1) の関係が成り立つとして、 $k=n+1$ の時を考える。今

$$(\mathbf{r}_{k+1}, \mathbf{p}_k) = (\mathbf{r}_k - \alpha_k A\mathbf{p}_k, \mathbf{p}_k) = (\mathbf{r}_k, \mathbf{p}_k) - \alpha_k (A\mathbf{p}_k, \mathbf{p}_k) = (\mathbf{r}_k, \mathbf{p}_k) - \frac{(\mathbf{r}_k, \mathbf{p}_k)}{(\mathbf{p}_k, A\mathbf{p}_k)} (A\mathbf{p}_k, \mathbf{p}_k) = 0$$

なので

$$(\mathbf{r}_{n+1}, \mathbf{p}_{n+1}) = (\mathbf{r}_{n+1}, \mathbf{r}_{n+1} - \beta_n \mathbf{p}_n) = (\mathbf{r}_{n+1}, \mathbf{r}_{n+1}) + \beta_n (\mathbf{r}_{n+1}, \mathbf{p}_n) = (\mathbf{r}_{n+1}, \mathbf{r}_{n+1})$$

となるので、(4.1) は成立する。(4.2) については $(\mathbf{p}_i, A\mathbf{p}_j) = 0$ ($i \neq j$) を用いれば、同じように求められる。

ところで、 β_k を計算する前に、 $\|\mathbf{r}_k\| < \varepsilon \|\mathbf{b}\|$ となっていたら計算を終了するという条件（これを、終了判定条件とよぶことにする）を確認する操作があるが、この条件を $(\mathbf{r}_k, \mathbf{r}_k) < \varepsilon (\mathbf{b}, \mathbf{b})$ と置き換えることも実は可能である。先に挙げたアルゴリズムの終了判定条件との違いは、 \mathbf{r}_k と \mathbf{b} がそれぞれ自身同士の内積をとっているかノルムとなっているかということである。

これらは大体同じことをしているが、大きな違いが2つある。1つめは反復回数と計算速度である。それぞれの判定条件で計算した結果、内積のものの方がノルムのものに比べて反復回数は約半分で計算速度は約2倍となる。しかし、2つめの違いなのだが、ノルムとなっているほうが内積のものよりも精度がよい。とりあえず解が出ればよい、という場合などは内積のほうの判定条件でもよいかもしいないが、こちらのほうが判定が荒いので、ノルムのほうの判定条件を用いたほうがよい。

実際に数値計算プログラムを実行した時、原型版よりも実用版のほうが計算速度も速く、さらに反復回数も少なくなることもある。これは、 α_k については、実用版のほうを用いると、分子が β_k の計算式の分母（また、 β_{k-1} の分子）と共通になるので、計算量が節約されるためである。 β_k については、実用版のほうが原型版に比べて計算時間が節約されるためである。しかし、安全性（直交性保持）を考えると、あまり実用版のほうは用いないほうがよい。

B.2.4 PCG 法

B.2.4.1 PCG 法と前処理について

$n \times n$ 行列に対して、CG 法は高々 n 回の反復で収束することは先に説明した。しかし、今回のような偏微分方程式の数値計算では、2次元領域を各辺 n （たとえば、 $n=100$ とする）等分すると、計算に用

いる行列の大きさは $n^2 \times n^2$ ($n^2 = 10000$) となってしまふ。つまり、最高で n^2 回の反復計算が行われるわけになるのだが、これでは時間発展を計算するのは難しくなる。そこで、反復回数が少なくなるように工夫をする。その方法として、次から説明する前処理を行う方法が知られている。CG 法には次のような性質があるので、これを考慮した前処理を行うことにする。

CG 法は、 $A\mathbf{x} = \mathbf{b}$ の行列 A の固有値分布によって収束の速さが決まる。固有値に重複があると収束が速く、重複でなくても密集固有値があると収束が速い。

ここで前処理行列として適当な正則行列 C を選んだとする。これにより解くべき方程式 $A\mathbf{x} = \mathbf{b}$ を

$$(C^{-1}AC^{-T})(C^T\mathbf{x}) = (C^{-1}\mathbf{b}) \quad \dots(*)$$

と変形し、この方程式に CG 法を適用する方法を、前処理付き共役勾配法 (Preconditioned Conjugate Gradient method; PCG 法) と呼ぶ。行列 C の選び方には色々あるが、基本的には「 A に近い」対称正定値行列を選ぶのが一般的である。この選び方に関する詳しいことについては後述する。ここで単純に “ $C^{-1}A\mathbf{x} = C^{-1}\mathbf{b}$ ” とすると、 $C^{-1}A$ が非対称行列となってしまう CG 法の適用範囲外になってしまう。そのため $C^{-1}AC^{-T}$ とすることによって行列の対称性を維持することができ、CG 法が適用できる。そのために (*) のような変形を考える。

ここで C をコレスキー分解¹して

$$C = U^tU$$

としておく。ここで、 U は上三角行列である。

$$f(\mathbf{x}) = \frac{1}{2}(\mathbf{x}, A\mathbf{x}) - (\mathbf{x}, \mathbf{b})$$

の最小化問題において

$$\tilde{\mathbf{x}} = U\mathbf{x}$$

と変換することにより

$$f(\mathbf{x}) = f(U^{-1}\tilde{\mathbf{x}}) = \frac{1}{2}(\tilde{\mathbf{x}}, U^{-T}AU^{-1}\tilde{\mathbf{x}}) - (U^{-1}\tilde{\mathbf{x}}, \mathbf{b}) = \frac{1}{2}(\tilde{\mathbf{x}}, U^{-T}AU^{-1}\tilde{\mathbf{x}}) - (\tilde{\mathbf{x}}, U^{-T}\mathbf{b})$$

$$\therefore \tilde{f}(\tilde{\mathbf{x}}) = \frac{1}{2}(\tilde{\mathbf{x}}, \tilde{A}\tilde{\mathbf{x}}) - (\tilde{\mathbf{x}}, \tilde{\mathbf{b}}) \quad (\text{ただし, } \tilde{A} = U^{-T}AU^{-1}, \tilde{\mathbf{b}} = U^{-T}\mathbf{b})$$

とすることができる。 $\tilde{f}(\tilde{\mathbf{x}})$ を最小にする $\tilde{\mathbf{x}}$ は

$$\tilde{A}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$$

の解であり、これが求まれば $\mathbf{x} = U^{-1}\tilde{\mathbf{x}}$ の関係から $f(\mathbf{x})$ を最小にする解 \mathbf{x} が得られる。このとき

$$\mathbf{x} = U^{-1}\tilde{\mathbf{x}} = U^{-1}(\tilde{A}^{-1}\tilde{\mathbf{b}}) = U^{-1}(U^{-T}AU^{-1})^{-1}(U^{-T}\mathbf{b}) = U^{-1}UA^{-1}U^T U^{-T}\mathbf{b} = A^{-1}\mathbf{b}$$

となり、 \mathbf{x} は $A\mathbf{x} = \mathbf{b}$ の解となる。もし、 \tilde{A} の固有値分布が A に比べて改善されたならば、 \tilde{f} に対して CG 法を適用するほうが速く収束することが期待できる。ここで注意することは、 \tilde{A} の固有値は $C^{-1}A$ の固有値に等しいことである。これは、 \tilde{A} を相似変換²すると

$$U^{-1}\tilde{A}U = U^{-1}(U^{-T}AU^{-1})U = (U^{-1}U^{-T})A = C^{-1}A$$

となることから明らかである。また、今回 $\tilde{\mathbf{x}} = U\mathbf{x}$ としたが

$$\tilde{A}^T = (U^{-T}AU^{-1})^T = (U^{-T}AU^{-1}) = \tilde{A} \quad (\because U^{-T}AU^{-1} \text{ は正定値対称})$$

¹正定値対称行列 A を、対角成分がすべて正であるような上三角行列 U とこの行列 U の共役転置 U^T との積 $A = U^T U$ に分解することを、コレスキー分解という。

²行列 A を正則行列 P によって、 $P^{-1}AP$ と変換することを相似変換という。この変換により、固有値は不変である。

$$\langle \tilde{\mathbf{x}}, \tilde{A}\tilde{\mathbf{x}} \rangle = \langle \tilde{\mathbf{x}}, U^{-1}AU^{-1}\tilde{\mathbf{x}} \rangle = \langle U^{-1}\tilde{\mathbf{x}}, AU^{-1}\tilde{\mathbf{x}} \rangle = \langle \mathbf{x}, A\mathbf{x} \rangle$$

となることから、対称性および正定値性は満たされている。このときの固有値はもとの A の固有値よりも、1 の近くに密集することになる。もし、極端に $C = A$ とおくと

$$U^{-T}AU^{-1} = U^{-T}(U^T U)U^{-1} = I$$

と単位行列になって、全固有値が1 となってしまうことから容易に想像できる。

B.2.4.2 前処理行列 C の選び方

前処理行列 C としての望ましい性質として、以下のようなものが挙げられる。

1. A に近いこと。
→ C が A に近ければ、 $C^{-1}A$ は単位行列に近くなるので、固有値は1 の近くに密集するため。
2. コレスキー分解 $U^T U$ が簡単に計算できること。
→ C のコレスキー分解に時間がかかるようでは実用にならないため。
3. U と A が同程度に疎であること。
→ U が疎でないとメモリ量、演算量が共に多くなってしまうため。

数値計算の計算量と以上の性質を踏まえて、 C の選び方として、行列 A の3重対角部分だけを取り出して、それを改めて C とすることがもっとも単純な方法として挙げられる。また他の方法としては、 A を不完全コレスキー分解した行列を C とする方法 (ICCG 法) や不完全 LU 分解した行列を C とする方法 (ILUCG 法) がある。

B.2.4.3 アルゴリズム

PCG 法 (原型版)

適当な初期ベクトル \mathbf{x}_0 を設定する。

$$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0 \quad \times$$

$$\mathbf{p}_0 = C^{-1}\mathbf{r}_0$$

$k = 0, 1, 2, \dots$ の順に、以下の手順を繰り返す。

$$\alpha_k = (\mathbf{p}_k, \mathbf{r}_k) / (\mathbf{p}_k, A\mathbf{p}_k)$$

$$\left. \begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k \\ \mathbf{r}_{k+1} &= \mathbf{r}_k - \alpha_k A\mathbf{p}_k \end{aligned} \right\} \times$$

ここで、もし $\|\mathbf{r}_k\| < \varepsilon \|\mathbf{b}\|$ ならば計算終了そうでなければ、以下の手順を続ける。

$$\beta_k = -(C^{-1}\mathbf{r}_{k+1}, A\mathbf{p}_k) / (\mathbf{p}_k, A\mathbf{p}_k)$$

$$\mathbf{p}_{k+1} = C^{-1}\mathbf{r}_{k+1} + \beta_k \mathbf{p}_k \quad \times$$

$$k = k + 1$$

PCG 法 (実用版)

適当な初期ベクトル \mathbf{x}_0 を設定する.

$$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0 \quad ※$$

$$\mathbf{q}_0 = C^{-1}\mathbf{r}_0$$

$$\mathbf{p}_0 = \mathbf{q}_0 \quad ※$$

$k = 0, 1, 2, \dots$ の順に, 以下の手順を繰り返す.

$$\alpha_k = (C^{-1}\mathbf{r}_k, \mathbf{r}_k) / (\mathbf{p}_k, A\mathbf{p}_k)$$

$$\left. \begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k \\ \mathbf{r}_{k+1} &= \mathbf{r}_k - \alpha_k A\mathbf{p}_k \end{aligned} \right\} ※$$

ここで, もし $\|\mathbf{r}_k\| < \varepsilon \|\mathbf{b}\|$ ならば計算終了そうでなければ, 以下の手順を続ける.

$$\mathbf{q}_{k+1} = C^{-1}\mathbf{r}_{k+1}$$

$$\beta_k = (\mathbf{q}_{k+1}, \mathbf{r}_{k+1}) / (\mathbf{q}_k, \mathbf{r}_k)$$

$$\mathbf{p}_{k+1} = \mathbf{q}_k + \beta_k \mathbf{p}_k \quad ※$$

$$k = k + 1$$

並列化計算を行う際は, ※の部分のループの前に並列化構文を挿入して, 並列化を行う.

上記の PCG 法のアルゴリズムは, $\tilde{A}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$ に対する CG 法を,

$$\tilde{\mathbf{x}} = U\mathbf{x}, \quad \tilde{\mathbf{p}} = U\mathbf{p}, \quad \tilde{\mathbf{r}} = U^{-T}\mathbf{r}$$

により, \sim のつかない変数に還元することで得られる.

また, 原型版と実用版の違いについては, CG 法の時と同様である. 今回のアルゴリズムも, CG 法の原型版と実用版からそれぞれ作成することができる.

B.2.5 ICCG 法

行列 A を, 上三角行列 U と U の転置行列の下三角行列 U^T と対角行列 D を用いて

$$A = U^T D U - R$$

と不完全コレスキー分解する. この時に前処理行列 C を

$$C = U^T D U$$

として, この C を用いて PCG 法を行う方法を不完全コレスキー分解つき共役勾配法 (Incomplete Cholesky CG method; ICCG 法) という.

B.2.5.1 不完全コレスキー (IC) 分解

上記でも書いたように, 不完全コレスキー分解とは

$$A = U^T D U - R$$

と行列 A を分解することである。もし疎行列 A をコレスキー分解したとする。するとこの時、 U 、 U^T は A に比べて密になってしまう。そこで、 A の非ゼロ要素の場所の集合を G_A 、すなわち

$$G_A = \{(i, j) \mid a_{ij} \neq 0\}$$

として

$$G \supseteq G_A$$

なる集合 G を予め決めておいて、 A をコレスキー分解する時に U 、 U^T の要素中で G に属する場所のものだけを計算し、他のものは 0 にする。これを不完全コレスキー分解とよぶ。

では、実際にどうするかというと

$$U = \begin{pmatrix} u_{1,1} & u_{1,2} & \cdots & \cdots & u_{1,N} \\ & u_{2,2} & & & u_{2,N} \\ & & \ddots & & \vdots \\ 0 & & & \ddots & \vdots \\ & & & & u_{N,N} \end{pmatrix}, \quad U^T = \begin{pmatrix} u_{1,1} & & & & \\ u_{1,2} & u_{2,2} & & & 0 \\ \vdots & & \ddots & & \\ \vdots & & & \ddots & \\ u_{1,N} & \cdots & \cdots & \cdots & u_{N,N} \end{pmatrix}$$

$$D = \begin{pmatrix} d_{1,1} & & & & \\ & \ddots & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & d_{N,N} \end{pmatrix}$$

として、 $U^T D U$ を計算して行列の形で書き下すと次のようになる。

$$\begin{pmatrix} u_{1,1}^2 d_{1,1} & u_{1,1} d_{1,1} u_{1,2} & \cdots & u_{1,1} d_{1,1} u_{1,N} \\ u_{1,2} d_{1,1} u_{1,1} & u_{1,2}^2 d_{1,1} + u_{2,2}^2 d_{2,2} & \cdots & u_{1,2} d_{1,1} u_{1,N} + u_{2,2} d_{2,2} u_{2,N} \\ u_{1,3} d_{1,1} u_{1,1} & u_{1,3} d_{1,1} u_{1,2} + u_{2,3} d_{2,2} u_{2,2} & \cdots & u_{1,3} d_{1,1} u_{1,N} + u_{2,3} d_{2,2} u_{2,N} + u_{3,3} d_{3,3} u_{3,N} \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ u_{1,N} d_{1,1} u_{1,1} & u_{1,N} d_{1,1} u_{1,2} + u_{2,N} d_{2,2} u_{2,2} & \cdots & u_{1,N}^2 d_{1,1} + \cdots + u_{N,N}^2 d_{N,N} \end{pmatrix}$$

ここで U は、 A の非ゼロ要素に対応した要素位置 G_A にだけ非ゼロ値がくるようにさせ、しかも

$$\begin{cases} a_{i,i} = \sum_k u_{k,i}^2 d_{k,i} & (k, i) \in G_A \\ a_{i,j} = \sum_k u_{k,i} d_{k,i} u_{k,j} & (k, i) \text{ および } (k, j) \in G_A \end{cases}$$

となるようにする。これより、 U と D の要素は次式により求めればよい。

$$\begin{cases} u_{i,i}^2 d_{i,i} = a_{i,i} - \sum_{k=1}^{i-1} u_{k,i}^2 d_{k,k} & (k, i) \in G_A \\ u_{i,i} d_{i,i} u_{i,j} = a_{i,j} - \sum_{k=1}^{i-1} u_{k,i} d_{k,k} u_{k,j} & (k, i) \text{ および } (k, j) \in G_A \end{cases}$$

ここで、 $d_{i,i} = u_{i,i}^{-1}$ とおけば、 $(i, i) \in G_A$ 、 $(i, j) \in G_A$ について

$$\begin{cases} u_{i,i} = a_{i,i} - \sum_{k=1}^{i-1} u_{k,i}^2 d_{k,k}, & u_{1,1} = a_{1,1} & (k, i) \in G_A \\ u_{i,j} = a_{i,j} - \sum_{k=1}^{i-1} u_{k,i} d_{k,k} u_{k,j} & & (k, i) \text{ および } (k, j) \in G_A \end{cases}$$

となる。

B.2.5.2 おまけ 1 : 不完全コレスキー分解と修正コレスキー分解との比較

上記で求めた $u_{i,i}$, $u_{i,j}$ と, 修正コレスキー分解 $\tilde{U}^T \tilde{D} \tilde{U}$ と比べてみよう. 修正コレスキー分解では

$$\begin{cases} \tilde{d}_{i,i} = a_{i,i} - \sum_{k=1}^{i-1} \tilde{u}_{k,j}^2 \tilde{d}_{k,k}, & \tilde{u}_{i,i} = 1 \\ \tilde{u}_{i,j} = a_{i,j} - \left(\sum_{k=1}^{i-1} \tilde{u}_{k,i} \tilde{d}_{k,k} \tilde{u}_{k,j} \right) / \tilde{d}_{i,i} \end{cases}$$

となる.

つまり, 不完全コレスキー分解では, $u_{i,i} = 1$ ではなく, 本来の修正コレスキー分解の $\tilde{d}_{i,i}$ にとり, $d_{i,i}$ も $u_{i,i}$ の逆数をとっている. $u_{i,i}$ や $d_{i,i}$ の決め方により別な不完全コレスキー分解も考えられる. 不完全コレスキー分解は 5 点差分または 7 点差分によって得られる特殊な行列に適用するとよい.

B.2.5.3 おまけ 2 : 修正コレスキー分解について

上記のおまけ 1 で出てきた修正コレスキー分解について, 簡単に説明する.

コレスキー分解 $A = U^T U$ で, A の要素を $a_{i,j}$, U の要素を $u_{i,j}$ とすると, $u_{i,j}$ は次のように求められる.

$$\begin{cases} u_{i,j} = (a_{i,j} - \sum_{k=1}^{i-1} u_{k,i} u_{k,j}) / u_{i,i} & i < j \leq N \\ u_{i,j} = \sqrt{a_{i,j} - \sum_{k=1}^{i-1} u_{k,i}^2} & i = j \end{cases}$$

上記のように, コレスキー分解では $i = j$ の時に平方根の計算を行わなくてはならない. これは, 四則演算に比べて計算コストが高い. そこで, この平方根の計算をなくすように改良されたのが, 今から説明していく修正コレスキー分解である.

まず, 上三角行列 U を次のように分解する.

$$U = \begin{pmatrix} u_{1,1} & & & & & & \\ & u_{2,2} & & & & & \\ & & \ddots & & & & \\ & & & \ddots & & & \\ & & & & \ddots & & \\ & & & & & \ddots & \\ & & & & & & u_{N,N} \end{pmatrix} \begin{pmatrix} 1 & \frac{u_{1,2}}{u_{1,1}} & \frac{u_{1,3}}{u_{1,1}} & \cdots & \cdots & \cdots & \frac{u_{1,N}}{u_{1,1}} \\ 0 & 1 & \frac{u_{2,3}}{u_{2,2}} & \cdots & \cdots & \cdots & \frac{u_{2,N}}{u_{2,2}} \\ \vdots & 0 & 1 & \ddots & & & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & 0 & & \ddots & \ddots & \frac{u_{N-1,N}}{u_{N-1,N-1}} \\ 0 & 0 & \cdots & \cdots & \cdots & 0 & 1 \end{pmatrix}$$

次に, これを $U = \tilde{D} \tilde{U}$ とおいて, $D = \tilde{D}^T \tilde{D}$ とすると,

$$A = U^T U = (\tilde{D} \tilde{U})^T (\tilde{D} \tilde{U}) = \tilde{U}^T \tilde{D}^T \tilde{D} \tilde{U} = \tilde{U}^T \tilde{D} \tilde{U}$$

となる。この分解を、修正コレスキー分解という。ここで、 D の要素を $d_{i,j}$ 、 \tilde{U} の要素を $\tilde{u}_{i,j}$ とすると、

$$\begin{aligned}
 A &= \tilde{U}^T D \tilde{U} \\
 &= \begin{pmatrix} \tilde{u}_{1,1} & \cdots & \cdots & \cdots & \tilde{u}_{1,N} \\ & \tilde{u}_{2,2} & & & \tilde{u}_{2,N} \\ & & \ddots & & \vdots \\ & 0 & & \ddots & \vdots \\ & & & & \tilde{u}_{N,N} \end{pmatrix} \begin{pmatrix} d_{1,1} & & & & \\ & \ddots & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & d_{N,N} \end{pmatrix} \begin{pmatrix} \tilde{u}_{1,1} & & & & \\ \tilde{u}_{1,2} & \tilde{u}_{2,2} & & & 0 \\ \vdots & & \ddots & & \\ \vdots & & & \ddots & \\ \tilde{u}_{1,N} & \cdots & \cdots & \cdots & \tilde{u}_{N,N} \end{pmatrix} \\
 &= \begin{pmatrix} \tilde{u}_{1,1}^2 d_{1,1} & \tilde{u}_{1,1} d_{1,1} \tilde{u}_{1,2} & \cdots & \tilde{u}_{1,1} d_{1,1} \tilde{u}_{1,N} \\ \tilde{u}_{1,2} d_{1,1} \tilde{u}_{1,1} & \tilde{u}_{1,2}^2 d_{1,1} + \tilde{u}_{2,2}^2 d_{2,2} & \cdots & \tilde{u}_{1,2} d_{1,1} \tilde{u}_{1,N} + \tilde{u}_{2,2} d_{2,2} \tilde{u}_{2,N} \\ \tilde{u}_{1,3} d_{1,1} \tilde{u}_{1,1} & \tilde{u}_{1,3} d_{1,1} \tilde{u}_{1,2} + \tilde{u}_{2,3} d_{2,2} \tilde{u}_{2,2} & \cdots & \tilde{u}_{1,3} d_{1,1} \tilde{u}_{1,N} + \tilde{u}_{2,3} d_{2,2} \tilde{u}_{2,N} + \tilde{u}_{3,3} d_{3,3} \tilde{u}_{3,N} \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ \tilde{u}_{1,N} d_{1,1} \tilde{u}_{1,1} & \tilde{u}_{1,N} d_{1,1} \tilde{u}_{1,2} + \tilde{u}_{2,N} d_{2,2} \tilde{u}_{2,2} & \cdots & \tilde{u}_{1,N}^2 d_{1,1} + \cdots + \tilde{u}_{N,N}^2 d_{N,N} \end{pmatrix}
 \end{aligned}$$

となるので、これより、

$$a_{i,j} = \sum_{k=1}^i \tilde{u}_{k,i} d_{k,k} \tilde{u}_{k,j} = \sum_{k=1}^{i-1} \tilde{u}_{k,i} d_{k,k} \tilde{u}_{k,j} + \tilde{u}_{i,i} d_{i,i} \tilde{u}_{i,j}$$

となる。 $\tilde{u}_{i,i} = 1$ に注意すれば、

$$\tilde{u}_{i,j} = \frac{1}{d_{i,i}} \left(a_{i,j} - \sum_{k=1}^{i-1} \tilde{u}_{k,i} d_{k,k} \tilde{u}_{k,j} \right) \quad i = 1, 2, \dots, j-1$$

となる。また、 $a_{i,i} = \sum_{k=1}^{i-1} \tilde{u}_{k,i} d_{k,k} \tilde{u}_{k,i}$ なので、

$$d_{i,i} = a_{i,i} - \sum_{k=1}^{i-1} \tilde{u}_{k,i}^2 d_{k,k} \quad i = 2, 3, \dots, N$$

と計算できる。

B.2.5.4 ICCG(1,1) 法

前章のおまけ1で書いた5点差分³によって得られる行列（特殊な5重対角行列となる）に適用するとよいのが、今から説明するIC(1,1)である。

行列 A の非ゼロ要素の場所の集合は、

$$G_A = \{(i, j) \mid j = i, i \pm 1, i \pm m\}$$

である。 A の対角要素を a_i 、副対角要素を b_i 、 c_i とする。ただし、 i は行番号を表す。

不完全コレスキー分解における非ゼロ要素の場所の集合 G を G_A に等しくしたものをIC(1,1)とよぶ。

$$A = U^T D U - R$$

³有限要素法の二次元問題の考え方で、まず長方形領域を考える。節点番号を x 方向、 y 方向の順につけ、 x 方向の節点数を m とすると、非ゼロ要素の場所の集合は、 $G_A = \{(i, j) \mid j = i, i \pm 1, i \pm m\}$ となる。5点差分を用いると、行列 A は5重対角行列となる。

としたときの U の対角要素を \tilde{a}_i , 副対角要素を \tilde{b}_i , \tilde{c}_i とし, D の要素を \tilde{d}_i とする.
 行列 A について,

$$A = \begin{pmatrix} a_1 & b_1 & & & c_1 & & & & & & \\ b_1 & a_2 & b_2 & & c_2 & & & & & & \\ & & \ddots & \ddots & \ddots & & & & & \ddots & \\ & & & \ddots & \ddots & \ddots & & & & & c_m \\ c_1 & & & & & \ddots & \ddots & \ddots & & & \\ & c_2 & & & & \ddots & \ddots & \ddots & & & \\ & & \ddots & & & \ddots & \ddots & \ddots & & & b_{N-1} \\ & & & c_m & & & & \ddots & & & \\ & & & & & & & b_{N-1} & & & a_N \end{pmatrix}$$

となるので, これを不完全コレスキー分解すると,

$$U^T D U - R = \begin{pmatrix} \tilde{a}_1 & & & & & & & & & & \\ \tilde{b}_1 & \tilde{a}_2 & & & & & & & & & \\ & \tilde{b}_2 & \tilde{a}_3 & & & & & & & & 0 \\ & & \ddots & \ddots & & & & & & & \\ \tilde{c}_1 & & \ddots & \ddots & \ddots & & & & & & \\ & \ddots & & \ddots & \ddots & \ddots & & & & & \\ & & & \tilde{c}_m & & & & & & & \tilde{b}_{N-1} \\ & & & & & & & \tilde{a}_N & & & \end{pmatrix} \begin{pmatrix} \tilde{d}_1 & & & & \\ & \ddots & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & \tilde{d}_N \end{pmatrix} \begin{pmatrix} \tilde{a}_1 & \tilde{b}_1 & & & \tilde{c}_1 & & & & & & \\ & \tilde{a}_2 & \tilde{b}_2 & & & \ddots & & & & & \\ & & \tilde{a}_3 & \tilde{b}_3 & & & & & & & \tilde{c}_m \\ & & & \ddots & \ddots & & & & & & \\ & & & & \ddots & \ddots & & & & & \\ & & & & & \ddots & \ddots & & & & \\ & & & & & & \ddots & & & & \\ & & & & & & & \ddots & & & \\ & & & & & & & & \ddots & & \\ & & & & & & & & & \tilde{b}_{N-1} & \\ & & & & & & & & & & \tilde{a}_N \end{pmatrix} - R$$

この行列計算を行い, さらに, $\tilde{a}_i = \tilde{d}_i^{-1}$ を用いると, 次のことが導かれる.

$$\begin{cases} a_i = \tilde{a}_i^2 \tilde{d}_i + \tilde{b}_{i-1}^2 \tilde{d}_{i-1} + \tilde{c}_{i-m}^2 \tilde{d}_{i-m} = \tilde{a}_i + \tilde{b}_{i-1}^2 \tilde{d}_{i-1} + \tilde{c}_{i-m}^2 \tilde{d}_{i-m} \\ b_i = \tilde{a}_i \tilde{d}_i \tilde{b}_i = \tilde{b}_i \\ c_i = \tilde{a}_i \tilde{d}_i \tilde{c}_i = \tilde{c}_i \end{cases}$$

$$\therefore \begin{cases} \tilde{a}_i = a_i - \tilde{b}_{i-1}^2 \tilde{d}_{i-1} - \tilde{c}_{i-m}^2 \tilde{d}_{i-m} \\ \tilde{b}_i = b_i \\ \tilde{c}_i = c_i \\ \tilde{d}_i = \frac{1}{\tilde{a}_i} \end{cases}$$

ただし, ここで添字が範囲外のものはゼロにするものとする. U の非対角要素 \tilde{b}_i , \tilde{c}_i は A の同位置にある要素とまったく同じものとなるので, 計算する必要はない. この $U^T D U$ によって前処理をした CG 法が, ICCG(1,1) 法である.

B.2.5.5 ICCG(1,2) 法

上記で説明した IC(1,1) において, 不完全コレスキー分解における非ゼロ要素の場所の集合 G を,

$$G = \{(i, j) \mid j = i, i \pm 1, i \pm m, i \pm (m-1)\}$$

としたものを IC(1,2) と呼ぶ. これは, 有限要素法の二次元問題でいう, 7 点差分にあたる. また行列 U については, 副対角要素 \tilde{c}_i の 1 つ内側に \tilde{e}_i を新たに加える形となる.

不完全分解 $U^T D U$ の要素は, 計算すると,

$$\begin{cases} \tilde{a}_i = a_i - \tilde{b}_{i-1}^2 \tilde{d}_{i-1} - \tilde{c}_{i-m}^2 \tilde{d}_{i-m} - \tilde{c}_{i-(m-1)}^2 \tilde{d}_{i-(m-1)} \\ \tilde{b}_i = b_i - c_{i-(m-1)} \tilde{c}_{i-(m-1)} \tilde{d}_{i-(m-1)} \\ \tilde{c}_i = c_i \\ \tilde{d}_i = \frac{1}{\tilde{a}_i} \\ \tilde{e}_i = -c_{i-1} \tilde{b}_{i-1} \tilde{d}_{i-1} \end{cases}$$

となる. 副対角要素 \tilde{c}_i は A の同位置にある要素と同じものとなる. これによって前処理をした CG 法が ICCG(1,2) 法である. この方法は, ICCG(1,1) 法よりも収束が速いと言われている.

B.3 CR 法, PCR 法, ICCR 法

B.3.1 CR 法について

CR 法とは, 共役残差法 (Conjugate Residual method) のことである. CG 法と同様に方程式 $A\mathbf{x} = \mathbf{b}$ の解法であるが, その違いは行列 A が非対称でもよいことである.

CR 法は, 任意の初期値 \mathbf{x}_0 から始めて, 関数

$$f(\mathbf{x}) = (\mathbf{b} - A\mathbf{x}, \mathbf{b} - A\mathbf{x}) = (\mathbf{r}, \mathbf{r})$$

を最小にする \mathbf{x} を逐次探索する方法である. そして $f(\mathbf{x})$ を最小にする \mathbf{x} が $A\mathbf{x} = \mathbf{b}$ の解となる. 実際,

$$f(\mathbf{x}) = (\mathbf{r}, \mathbf{r}) \geq 0$$

で, $\mathbf{r} = \mathbf{0}$ の時に限って最小値 0 となる. したがって, この方法では CG 法とは異なり A の正定値性も対称性も必要ない.

ただし, この方法は CG 法のように, A が $N \times N$ 行列の時に, 高々 N 回の反復で正解が求められる保証はない.

B.3.1.1 アルゴリズム

CR 法 (原型版)

適当な初期ベクトル \mathbf{x}_0 を設定する.

$$\left. \begin{array}{l} \mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0 \\ \mathbf{p}_0 = \mathbf{r}_0 \end{array} \right\} \times$$

$k = 1, 2, \dots$ の順に, 以下の手順を繰り返す.

$$\left. \begin{array}{l} \alpha_k = (\mathbf{r}_k, A\mathbf{p}_k) / (A\mathbf{p}_k, A\mathbf{p}_k) \\ \mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \\ \mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k \end{array} \right\} \times$$

ここで, もし $\|\mathbf{r}_k\| < \varepsilon \|\mathbf{b}\|$ ならば計算終了. そうでなければ, 以下の手順を続ける.

$$\beta_k = -(A\mathbf{r}_{k+1}, A\mathbf{p}_k) / (A\mathbf{p}_k, A\mathbf{p}_k)$$

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k \quad \times$$

$$k = k + 1$$

CR 法 (実用版)

適当な初期ベクトル \mathbf{x}_0 を設定する.

$$\left. \begin{aligned} \mathbf{r}_0 &= \mathbf{b} - A\mathbf{x}_0 \\ \mathbf{p}_0 &= \mathbf{r}_0 \\ \mathbf{q}_0 &= A\mathbf{p}_0 \end{aligned} \right\} \ast$$

$k = 1, 2, \dots$ の順に, 以下の手順を繰り返す.

$$\left. \begin{aligned} \alpha_k &= (\mathbf{r}_k, \mathbf{q}_k) / (\mathbf{q}_k, \mathbf{q}_k) \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k \\ \mathbf{r}_{k+1} &= \mathbf{r}_k - \alpha_k \mathbf{q}_k \end{aligned} \right\} \ast$$

ここで, もし $\|\mathbf{r}_k\| < \varepsilon \|\mathbf{b}\|$ ならば, 計算終了. そうでなければ, 以下の手順を続ける.

$$\left. \begin{aligned} \beta_k &= -(A\mathbf{r}_{k+1}, \mathbf{q}_k) / (\mathbf{q}_k, \mathbf{q}_k) \\ \mathbf{p}_{k+1} &= \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k \\ \mathbf{q}_{k+1} &= A\mathbf{r}_{k+1} + \beta_k \mathbf{q}_k \end{aligned} \right\} \ast$$

$$k = k + 1$$

並列化計算を行う際は, \ast の部分のループの前に並列化構文を挿入して, 並列化を行う.

B.3.1.2 原型版と実用版の違いについて

原型版のアルゴリズムでは, 1 回の反復の中に, $A\mathbf{p}_k$ と $A\mathbf{r}_{k+1}$ の 2 ヲ所で A をかける計算が必要になるが, 新しいベクトル \mathbf{q}_k を導入して, \mathbf{p}_{k+1} を求める式に A をかけて,

$$A\mathbf{p}_{k+1} = A\mathbf{r}_{k+1} + \beta_k A\mathbf{p}_k$$

となることに注意すると, この計算により $A\mathbf{p}_{k+1}$ も $A\mathbf{r}_{k+1}$ も求められるので, A のかけ算を 1 回だけにすることが出来る.

したがって, このことから作成されたものが, 実用版のアルゴリズムである. 計算回数が節約されていることがあり, 実際に数値計算プログラムを実行すると, 原型版よりも実用版のほうが計算速度は速い.

B.3.1.3 PCR 法について

PCR 法とは, 前処理付き共役残差法 (Preconditioned Conjugate Residual method) のことで, 前処理行列として適当な正則行列 C を選び, これにより解くべき方程式 $A\mathbf{x} = \mathbf{b}$ を,

$$(C^{-1}AC^T)(C^T\mathbf{x}) = (C^{-1}\mathbf{b})$$

と変形して, この方程式に CR 法を適用する方法である. PCG 法との違いは, 適用する反復法が CG 法か CR 法かという点だけであるので, 詳細は割愛する.

B.3.1.4 アルゴリズム

PCR 法 (原型版)

適当な初期ベクトル \mathbf{x}_0 を設定する.

$$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0 \quad ※$$

$$\mathbf{p}_0 = C^{-1}\mathbf{r}_0$$

$k = 0, 1, 2, \dots$ の順に, 以下の手順を繰り返す.

$$\alpha_k = (\mathbf{r}_k, A\mathbf{p}_k) / (A\mathbf{p}_k, A\mathbf{p}_k)$$

$$\left. \begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k \\ \mathbf{r}_{k+1} &= \mathbf{r}_k - \alpha_k A\mathbf{p}_k \end{aligned} \right\} ※$$

ここで, もし $\|\mathbf{r}_k\| < \varepsilon \|\mathbf{b}\|$ ならば計算終了. そうでなければ, 以下の手順を続ける.

$$\beta_k = -(AC^{-1}\mathbf{r}_{k+1}, A\mathbf{p}_k) / (A\mathbf{p}_k, A\mathbf{p}_k)$$

$$\mathbf{p}_{k+1} = C^{-1}\mathbf{r}_{k+1} + \beta_k \mathbf{p}_k \quad ※$$

$$k = k + 1$$

PCR 法 (実用版)

適当な初期ベクトル \mathbf{x}_0 を設定する.

$$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0 \quad ※$$

$$\mathbf{p}_0 = C^{-1}\mathbf{r}_0$$

$$\mathbf{q}_0 = A\mathbf{p}_0$$

$k = 0, 1, 2, \dots$ の順に, 以下の手順を繰り返す.

$$\alpha_k = (\mathbf{r}_k, \mathbf{q}_k) / (\mathbf{q}_k, \mathbf{q}_k)$$

$$\left. \begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k \\ \mathbf{r}_{k+1} &= \mathbf{r}_k - \alpha_k \mathbf{q}_k \end{aligned} \right\} ※$$

ここで, もし $\|\mathbf{r}_k\| < \varepsilon \|\mathbf{b}\|$ ならば計算終了. そうでなければ, 以下の手順を続ける.

$$\beta_k = -(AC^{-1}\mathbf{r}_{k+1}, \mathbf{q}_k) / (\mathbf{q}_k, \mathbf{q}_k)$$

$$\left. \begin{aligned} \mathbf{p}_{k+1} &= C^{-1}\mathbf{r}_{k+1} + \beta_k \mathbf{p}_k \\ \mathbf{q}_{k+1} &= AC^{-1}\mathbf{r}_{k+1} + \beta_k \mathbf{q}_k \end{aligned} \right\} ※$$

$$k = k + 1$$

並列化計算を行う際は, ※の部分のループの前に並列化構文を挿入して並列化を行う.

B.3.2 ICCR 法について

PCR 法同様に ICCR 法についても, ICCG 法との違いは適用する反復法が CG 法か CR 法かという点だけであるので, こちらも詳細は割愛する.

付録C 発展編の完成部分 (有限体積法, その他)

C.1 軸対称問題の数値計算法 (有限体積近似)

次の軸対称問題において一様な差分化を行うと $r = 0$ の近くで著しく精度が悪くなる. そこで原点近くで離散化誤差が小さくなるように不均一な差分化を行う手法を解説する.

$$\frac{\partial u}{\partial t} = d \frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial u}{\partial r} \right) + f(u), \quad t > 0, 0 < r < R, \quad (\text{C.1})$$

$$\frac{\partial u}{\partial r}(0, r) = u_0(r), \quad 0 \leq x \leq R, \quad (\text{C.2})$$

$$\frac{\partial u}{\partial r}(t, 0) = \frac{\partial u}{\partial r}(t, R) = 0, \quad t > 0. \quad (\text{C.3})$$

$$\int_0^R u_t r dr = \int_0^R d \frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial u}{\partial r} \right) r dr + \int_0^R f(u) r dr, \quad (\text{C.4})$$

$$\begin{aligned} \int_0^R (u_t - f(u)) r dr &= \int_0^{\frac{1}{2}r_1} (u_t - f(u)) r dr + \sum_{j=1}^{N-1} \int_{\frac{1}{2}(r_{j-1}+r_j)}^{\frac{1}{2}(r_j+r_{j+1})} (u_t - f(u)) r dr \\ &\quad + \int_{\frac{1}{2}(r_{N-1}+r_N)}^{r_N} (u_t - f(u)) r dr. \end{aligned} \quad (\text{C.5})$$

$u(t, r)$ に対して $u_j = u(t, r_j)$ とおく.

$$\int_0^{\frac{1}{2}r_1} (u_t - f(u)) r dr \simeq \left(\frac{d}{dt} u_0 - f(u_0) \right) \int_0^{\frac{1}{2}r_1} r dr = \frac{1}{8} r_1^2 \left(\frac{d}{dt} u_0 - f(u_0) \right) \quad (\text{C.6})$$

$$\begin{aligned} \int_{\frac{1}{2}(r_{N-1}+r_N)}^{r_N} (u_t - f(u)) r dr &\simeq \left(\frac{d}{dt} u_N - f(u_N) \right) \int_{\frac{1}{2}(r_{N-1}+r_N)}^{r_N} r dr \\ &= \left(\frac{d}{dt} u_N - f(u_N) \right) \left(\frac{1}{2} r_N^2 - \frac{1}{8} (r_{N-1} + r_N)^2 \right). \end{aligned} \quad (\text{C.7})$$

$$\begin{aligned} \int_{\frac{1}{2}(r_{j-1}+r_j)}^{\frac{1}{2}(r_j+r_{j+1})} (u_t - f(u)) r dr &\simeq \left(\frac{d}{dt} u_j - f(u_j) \right) \int_{\frac{1}{2}(r_{j-1}+r_j)}^{\frac{1}{2}(r_j+r_{j+1})} r dr \\ &= \left(\frac{d}{dt} u_j - f(u_j) \right) \left(\frac{1}{8} (r_{j+1} + r_j)^2 - \frac{1}{8} (r_j + r_{j-1})^2 \right). \end{aligned} \quad (\text{C.8})$$

拡散項の離散化

$$\begin{aligned} \int_0^R d \frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial u}{\partial r} \right) r dr &= \int_0^{\frac{1}{2}r_1} d \frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial u}{\partial r} \right) r dr + \sum_{j=1}^{N-1} \int_{\frac{1}{2}(r_{j-1}+r_j)}^{\frac{1}{2}(r_j+r_{j+1})} d \frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial u}{\partial r} \right) r dr \\ &\quad + \int_{\frac{1}{2}(r_{N-1}+r_N)}^{r_N} d \frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial u}{\partial r} \right) r dr. \end{aligned} \quad (\text{C.9})$$

$$\begin{aligned}
\int_0^{\frac{1}{2}r_1} d\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial u}{\partial r} \right) r dr &= [dru_r]_0^{\frac{1}{2}r_1} \\
&= d\frac{1}{2}r_1 \frac{u_1 - u_0}{r_1 - r_0} = d\frac{1}{2}r_1 \frac{u_1 - u_0}{r_1} = \frac{d}{2}(u_1 - u_0). \tag{C.10}
\end{aligned}$$

$$\begin{aligned}
\int_{\frac{1}{2}(r_{N-1}+r_N)}^{r_N} d\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial u}{\partial r} \right) r dr &= [dru_r]_{\frac{1}{2}(r_{N-1}+r_N)}^{r_N} \\
&= -d\frac{1}{2}(r_{N-1} + r_N) \frac{u_N - u_{N-1}}{r_N - r_{N-1}}. \tag{C.11}
\end{aligned}$$

$$\begin{aligned}
&\int_{\frac{1}{2}(r_{j-1}+r_j)}^{\frac{1}{2}(r_j+r_{j+1})} d\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial u}{\partial r} \right) r dr = [dru_r]_{\frac{1}{2}(r_{j-1}+r_j)}^{\frac{1}{2}(r_j+r_{j+1})} \\
&= d\frac{1}{2}(r_{j+1} + r_j) \frac{u_{j+1} - u_j}{r_{j+1} - r_j} - d\frac{1}{2}(r_j + r_{j-1}) \frac{u_j - u_{j-1}}{r_j - r_{j-1}}. \tag{C.12}
\end{aligned}$$

従って, (C.6), (C.10) より

$$\frac{1}{8}r_1^2 \left(\frac{d}{dt}u_0 - f(u_0) \right) = \frac{d}{2}(u_1 - u_0),$$

$j = 0$ の離散方程式

$$\frac{d}{dt}u_0 = \frac{4}{r_1^2}d(u_1 - u_0) + f(u_0). \tag{C.13}$$

(C.7), (C.11) より

$$\left(\frac{d}{dt}u_N - f(u_N) \right) \left(\frac{1}{2}r_N^2 - \frac{1}{8}(r_{N-1} + r_N)^2 \right) = -d\frac{1}{2}(r_{N-1} + r_N) \frac{u_N - u_{N-1}}{r_N - r_{N-1}},$$

$j = N$ の離散方程式

$$\frac{d}{dt}u_N = -\frac{d}{2} \frac{r_{N-1} + r_N}{\frac{1}{2}r_N^2 - \frac{1}{8}(r_{N-1} + r_N)^2} \frac{u_N - u_{N-1}}{r_N - r_{N-1}} + f(u_N). \tag{C.14}$$

(C.8), (C.12) より

$$\begin{aligned}
&\left(\frac{d}{dt}u_j - f(u_j) \right) \left(\frac{1}{8}(r_{j+1} + r_j)^2 - \frac{1}{8}(r_j + r_{j-1})^2 \right) \\
&= \frac{d}{2} \left((r_{j+1} + r_j) \frac{u_{j+1} - u_j}{r_{j+1} - r_j} - (r_j + r_{j-1}) \frac{u_j - u_{j-1}}{r_j - r_{j-1}} \right),
\end{aligned}$$

$j = 1, 2, \dots, N-2, N-1$ の離散方程式

$$\begin{aligned}
\frac{d}{dt}u_j &= \frac{d}{2} \frac{1}{\frac{1}{8}(r_{j+1} + r_j)^2 - \frac{1}{8}(r_j + r_{j-1})^2} \\
&\quad \left((r_{j+1} + r_j) \frac{u_{j+1} - u_j}{r_{j+1} - r_j} - (r_j + r_{j-1}) \frac{u_j - u_{j-1}}{r_j - r_{j-1}} \right) + f(u_j). \tag{C.15}
\end{aligned}$$

C.2 2次元円板領域での数値計算法

$$\frac{\partial u}{\partial t} = D\Delta u, \quad t > 0, \quad x^2 + y^2 < R^2, \quad (\text{C.16})$$

$$u(0, x, y) = u_0(x, y), \quad x^2 + y^2 < R^2, \quad (\text{C.17})$$

$$\frac{\partial u}{\partial n} = 0, \quad x^2 + y^2 = R^2, \quad t > 0. \quad (\text{C.18})$$

ただし, n は外向き法線ベクトル,

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \quad (\text{C.19})$$

(C.16) に対して変数変換

$$x = r \cos \theta, \quad y = r \sin \theta \quad (\text{C.20})$$

を行うと (C.19) は次のように書き換えられる :

$$\Delta_r = \frac{\partial^2}{\partial r^2} + \frac{1}{r} \frac{\partial}{\partial r} + \frac{1}{r^2} \frac{\partial^2}{\partial \theta^2}. \quad (\text{C.21})$$

このとき, (C.16)–(C.18) は次のように書き換えられる :

$$\frac{\partial u}{\partial t} = D \left(\frac{\partial^2}{\partial r^2} + \frac{1}{r} \frac{\partial}{\partial r} + \frac{1}{r^2} \frac{\partial^2}{\partial \theta^2} \right) u, \quad t > 0, 0 < r < R, 0 < \theta < 2\pi, \quad (\text{C.22})$$

$$u(0, r, \theta) = u_0(r, \theta), \quad 0 < r < R, 0 < \theta < 2\pi, \quad (\text{C.23})$$

$$\frac{\partial u}{\partial r} = 0, \quad t > 0, r = 0, R, 0 < \theta < 2\pi, \quad (\text{C.24})$$

$$u(t, r, 0) = u(t, r, 2\pi), \quad \frac{\partial}{\partial \theta} u(t, r, 0) = \frac{\partial}{\partial \theta} u(t, r, 2\pi), \quad t > 0, 0 < r < R. \quad (\text{C.25})$$

今, N_r, N_θ を自然数とし

$$\Delta r = \frac{R}{N_r}, \quad \Delta \theta = \frac{2\pi}{N_\theta}$$

と置き

$$r_i = i \cdot \Delta r \quad (0 \leq i \leq N_r), \quad \theta_j = j \cdot \Delta \theta \quad (0 \leq j \leq N_\theta)$$

を定義する. 分点は O (原点) と (r_i, θ_j) ($1 \leq i \leq N_r; 0 \leq j \leq N_\theta$) になり, $u(t, r, \theta)$ を

$$u_{i,j}^m = u(m\Delta t, r_i, \theta_j) \quad (\text{C.26})$$

と離散化する. ただし, $\Delta t > 0$ は時間離散間隔である. ここで,

$$r_{i+1/2} = (r_i + r_{i+1})/2$$

とおき, $0 \leq a \leq 1$ に対して

$$u_{i,j+1,k}^{m+a} = a u_{i,j+1}^m + (1-a) u_{i,j+1}^{m+1}. \quad (\text{C.27})$$

を定義する¹.

境界条件 (C.24) の $r = R$ に対する離散化は $0 \leq j \leq N_\theta - 1, m \geq 0$ に対して

$$u_{N_r+1,j}^m = u_{N_r-1,j}^m \quad (\text{C.28})$$

¹ $a = 1$ なら陽解法, $a = 0$ なら完全陰解法にそれぞれ対応する.

となり, 境界条件 (C.25) に対する離散化は $1 \leq i \leq N_r$, $m \geq 0$ に対して

$$u_{i,0}^m = u_{i,N_\theta}^m, \quad u_{i,-1}^m = u_{i,N_\theta-1}^m \quad (\text{C.29})$$

となる. 原点での離散化は境界条件 (C.24) を考慮して, 次のように書くことができる:

$$\frac{\pi}{4} \Delta r^2 \cdot \frac{1}{\Delta t} (u_0^{m+1} - u_0^m) = \sum_{j=1}^{N_\theta} \frac{1}{2} \Delta r \Delta \theta \frac{1}{\Delta r} (u_{1,j}^{m+a} - u_0^{m+a}). \quad (\text{C.30})$$

また

$$\frac{\partial^2 u}{\partial r^2} + \frac{1}{r} \frac{\partial u}{\partial r} = \frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial u}{\partial r} \right). \quad (\text{C.31})$$

であるから, 前節の有限体積法による軸対称問題の離散化において等間隔で空間を分割すると

$$\begin{aligned} \frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial u}{\partial r} \right) &\sim \frac{1}{r_i} \left(r_{i+1/2} \frac{u_{i+1,j} - u_{i,j}}{\Delta r^2} - r_{i-1/2} \frac{u_{i,j} - u_{i-1,j}}{\Delta r^2} \right) \\ &= \left(\frac{2i+1}{i} \frac{u_{i+1,j} - u_{i,j}}{\Delta r^2} - \frac{2i-1}{i} \frac{u_{i,j} - u_{i-1,j}}{\Delta r^2} \right) \end{aligned} \quad (\text{C.32})$$

となることから, $1 \leq i \leq N_r$, $0 \leq j \leq N_\theta - 1$ に対しては次のように離散化できる:

$$\begin{aligned} \frac{1}{\Delta t} (u_{i,j}^{m+1} - u_{i,j}^m) &= D \left(\frac{1}{\Delta r^2} \left(\frac{2i+1}{i} (u_{i+1,j}^{m+a} - u_{i,j}^{m+a}) - \frac{2i-1}{i} (u_{i,j}^{m+a} - u_{i-1,j}^{m+a}) \right) \right. \\ &\quad \left. + \left(\frac{1}{r_i \Delta \theta} \right)^2 (u_{i,j+1}^{m+a} - 2u_{i,j}^{m+a} + u_{i,j-1}^{m+a}) \right). \end{aligned} \quad (\text{C.33})$$

C.3 円柱領域での数値計算法

$$\frac{\partial u}{\partial t} = D \left(\frac{\partial^2}{\partial r^2} + \frac{1}{r} \frac{\partial}{\partial r} + \frac{1}{r^2} \frac{\partial^2}{\partial \theta^2} + \frac{\partial^2}{\partial z^2} \right) u, \quad t > 0, (r, \theta, z) \in \partial \Omega, \quad (\text{C.34})$$

初期条件:

$$u(0, r, \theta, z) = u_0(r, \theta, z), \quad (r, \theta, z) \in \bar{\Omega}, \quad (\text{C.35})$$

境界条件:

$$\frac{\partial}{\partial r} u(t, r, \theta, z) = 0, \quad t > 0, \quad r = 0, R, \quad 0 \leq \theta < 2\pi, \quad 0 < z < L_z, \quad (\text{C.36})$$

$$\frac{\partial}{\partial z} u(t, r, \theta, z) = 0, \quad t > 0, \quad 0 < r < R, \quad 0 \leq \theta < 2\pi, \quad z = 0, L_z, \quad (\text{C.37})$$

$$u(t, r, 0, z) = u(t, r, 2\pi, z), \quad t > 0, \quad 0 < r < R, \quad 0 < z < L_z, \quad (\text{C.38})$$

$$\frac{\partial}{\partial \theta} u(t, r, 0, z) = \frac{\partial}{\partial \theta} u(t, r, 2\pi, z), \quad t > 0, \quad 0 < r < R, \quad 0 < z < L_z. \quad (\text{C.39})$$

この数値計算は円板領域での数値計算法を z 方向に拡張すればよいだけである.

C.4 移流拡散方程式の数値計算法

$$u_t = du_{xx} - h(x)u_x, \quad x \in I. \quad (\text{C.40})$$

d が非常に小さい場合, $h(x)u_x$ の差分化には注意が必要である. このような場合は風上差分といわれる差分化を行う.

C.4.1 1 次風上差分

移流項の差分化を $h(u)$ の符号によって場合分けする:

$$h(x)u_x \simeq \begin{cases} h(x_j) \frac{u_j - u_{j-1}}{\Delta x} + O(\Delta x), & h(x_j) \geq 0, \\ h(x_j) \frac{u_{j+1} - u_j}{\Delta x} + O(\Delta x), & h(x_j) < 0. \end{cases} \quad (\text{C.41})$$

(C.41) を書き換えるとつぎのようになる:

$$h(x)u_x \simeq \frac{h(x_j) + |h(x_j)|}{2} \left(\frac{u_j - u_{j-1}}{\Delta x} \right) + \frac{h(x_j) - |h(x_j)|}{2} \left(\frac{u_{j+1} - u_j}{\Delta x} \right). \quad (\text{C.42})$$

従って, 次の 1 次風上差分の公式が得られる.

1 次風上差分

$$h(x) \frac{\partial u}{\partial x} \simeq h(x_j) \frac{u_{j+1} - u_{j-1}}{2\Delta x} - |h(x_j)| \frac{u_{j+1} - 2u_j + u_{j-1}}{2\Delta x}. \quad (\text{C.43})$$

第 2 項は人工粘性項であり, 1 次風上差分では, 拡散項の離散化が人工粘性として与えられる.

C.4.2 3 次風上差分

移流項の打ち切り誤差を $O(\Delta x^3)$ とすると, 風上差分は次のように書ける:

$$h(x)u_x \simeq \begin{cases} h(x_j) \frac{u_{j-2} - 6u_{j-1} + 3u_j + 2u_{j+1}}{6\Delta x} + O(\Delta x^3), & h(x_j) > 0, \\ h(x_j) \frac{-u_{j+2} + 6u_{j+1} - 3u_j - 2u_{j-1}}{6\Delta x} + O(\Delta x^3), & h(x_j) < 0. \end{cases} \quad (\text{C.44})$$

(C.44) は次のように書き換えられる:

$$h(x)u_x \simeq \frac{h(x_j) + |h(x_j)|}{2} \left(\frac{u_{j-2} - 6u_{j-1} + 3u_j + 2u_{j+1}}{6\Delta x} \right) + \frac{h(x_j) - |h(x_j)|}{2} \left(\frac{-u_{j+2} + 6u_{j+1} - 3u_j - 2u_{j-1}}{6\Delta x} \right). \quad (\text{C.45})$$

従って, 3 次風上差分の公式を得る:

3 次風上差分

$$h(x) \frac{\partial u}{\partial x} \simeq h(x_j) \frac{-u_{j+2} + 8u_{j+1} - 8u_{j-1} + u_{j-2}}{12\Delta x} + |h(x_j)| \frac{u_{j+2} - 4u_{j+1} + 6u_j - 4u_{j-1} + u_{j-2}}{12\Delta x}. \quad (\text{C.46})$$

第 2 項は人工粘性項と呼ばれる.

C.5 非線形拡散方程式の離散化 (差分法)

次のような非線形拡散方程式の初期値境界値問題に対する差分法を説明する：

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x} \left(d(u) \frac{\partial u}{\partial x} \right), \quad t > 0, 0 < x < L, \quad (\text{C.47})$$

$$u(0, x) = u_0(x), \quad 0 \leq x \leq L, \quad (\text{C.48})$$

$$\frac{\partial u}{\partial x}(t, 0) = \frac{\partial u}{\partial x}(t, L) = 0. \quad (\text{C.49})$$

区間 $(0, L)$ を $(x_0, x_1, \dots, x_i, \dots, x_n)$ と等分割する. このとき $\Delta x = L/N$ となる. (C.47) の右辺に対して以下のように中心差分を行う.

$$\begin{aligned} \frac{\partial}{\partial x} \left(d(u) \frac{\partial u}{\partial x} \right) &\sim \frac{1}{\Delta x} \left(d(u(t, x + \Delta x/2)) \frac{\partial u(t, x + \Delta x/2)}{\partial x} - d(u(t, x - \Delta x/2)) \frac{\partial u(t, x - \Delta x/2)}{\partial x} \right) \\ &\sim \frac{1}{\Delta x} \left(d(u(t, x + \Delta x/2)) \frac{u(t, x + \Delta x) - u(t, x)}{\Delta x} - d(u(t, x - \Delta x/2)) \frac{u(t, x) - u(t, x - \Delta x)}{\Delta x} \right) \\ &= \frac{1}{\Delta x^2} \left(d \left(u(t, x + \frac{\Delta x}{2}) \right) u(t, x + \Delta x) - \left(d \left(u(t, x + \frac{\Delta x}{2}) \right) + d \left(u(t, x - \frac{\Delta x}{2}) \right) \right) u(t, x) \right. \\ &\quad \left. + d \left(u(t, x - \frac{\Delta x}{2}) \right) u(t, x - \Delta x) \right). \end{aligned} \quad (\text{C.50})$$

ここで

$$u \left(t, x + \frac{\Delta x}{2} \right) = \frac{u(t, x + \Delta x) + u(t, x)}{2}, \quad u \left(t, x - \frac{\Delta x}{2} \right) = \frac{u(t, x) + u(t, x - \Delta x)}{2}$$

とする. さらに, $u(k \times \Delta t, i \times \Delta x) = u_i^k(t)$ と記述すると, 右辺は次のように与えられる.

$$\begin{aligned} &\frac{\partial}{\partial x} \left(d(u) \frac{\partial u}{\partial x} \right) \\ &\sim \frac{1}{\Delta x^2} \left(d \left(\frac{u_{i+1}^k + u_i^k}{2} \right) u_{i+1}^k - \left(d \left(\frac{u_{i+1}^k + u_i^k}{2} \right) + d \left(\frac{u_i^k + u_{i-1}^k}{2} \right) \right) u_i^k + d \left(\frac{u_i^k + u_{i-1}^k}{2} \right) u_{i-1}^k \right). \end{aligned}$$

境界条件から

$$u_{-1}^k = u_1^k, \quad u_{n-1}^k = u_{n+1}^k, \quad (k = 1, 2, 3, \dots)$$

より, $i = 0$ のとき

$$\frac{\partial}{\partial x} \left(d(u) \frac{\partial u}{\partial x} \right) \sim \frac{1}{\Delta x^2} \left(2d \left(\frac{u_1^k + u_0^k}{2} \right) u_1^k - \left(2d \left(\frac{u_1^k + u_0^k}{2} \right) \right) u_0^k \right)$$

となり, $i = n$ のとき

$$\frac{\partial}{\partial x} \left(d(u) \frac{\partial u}{\partial x} \right) \sim \frac{1}{\Delta x^2} \left(2d \left(\frac{u_n^k + u_{n-1}^k}{2} \right) u_{n-1}^k - \left(2d \left(\frac{u_n^k + u_{n-1}^k}{2} \right) \right) u_n^k \right)$$

となる.

上記で示した非線形拡散方程式に対する差分方程式は有限体積法を用いると自然に導出できる. いま, $u(t, x)$ の空間離散化を考える. 区間 $(0, L)$ を $(x_0, x_1, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n)$ と分割し, 図 C.1 のよう

に $u(t, x)$ を区分定数関数 $u_i(t)$ で近似する²。このとき，(C.47) の両辺を区間 $(0, L)$ で積分すると

$$\int_0^L \frac{\partial u}{\partial t} dx = \int_0^L \frac{\partial}{\partial x} \left(d(u) \frac{\partial u}{\partial x} \right) dx$$

となる。

$$\sum_{i=0}^N \int_{\frac{x_{i-1}+x_i}{2}}^{\frac{x_i+x_{i+1}}{2}} \frac{d}{dt} u dx = \sum_{i=0}^N \int_{\frac{x_{i-1}+x_i}{2}}^{\frac{x_i+x_{i+1}}{2}} \frac{\partial}{\partial x} \left(d(u) \frac{\partial u}{\partial x} \right) dx$$

と書き換えることができる。ただし， $i=0$ のときは

$$\int_{x_0}^{\frac{x_0+x_1}{2}} \frac{\partial}{\partial t} u dx = \int_{x_0}^{\frac{x_0+x_1}{2}} \frac{\partial}{\partial x} \left(d(u) \frac{\partial u}{\partial x} \right) dx,$$

$i=n$ のときは

$$\int_{\frac{x_{n-1}+x_n}{2}}^{x_n} \frac{\partial}{\partial t} u dx = \int_{\frac{x_{n-1}+x_n}{2}}^{x_n} \frac{\partial}{\partial x} \left(d(u) \frac{\partial u}{\partial x} \right) dx$$

とする。ここで，区分定数関数 $u_i(t)$ を用いると $1 \leq i \leq n-1$ において，左辺は

$$\frac{d}{dt} u_i(t) \int_{\frac{x_{i-1}+x_i}{2}}^{\frac{x_i+x_{i+1}}{2}} dx = \frac{d}{dt} u_i(t) \times \frac{x_{i+1} - x_{i-1}}{2}$$

となり，右辺は

$$\int_{\frac{x_{i-1}+x_i}{2}}^{\frac{x_i+x_{i+1}}{2}} \frac{\partial}{\partial x} \left(d(u) \frac{\partial u}{\partial x} \right) dx = \left[d(u) \frac{\partial u}{\partial x} \right]_{\frac{x_{i-1}+x_i}{2}}^{\frac{x_i+x_{i+1}}{2}} \quad (\text{C.51})$$

$$= d \left(u \left(t, \frac{x_i + x_{i+1}}{2} \right) \right) \frac{\partial}{\partial x} u \left(t, \frac{x_i + x_{i+1}}{2} \right) - d \left(u \left(t, \frac{x_{i-1} + x_i}{2} \right) \right) \frac{\partial}{\partial x} u \left(t, \frac{x_{i-1} + x_i}{2} \right). \quad (\text{C.52})$$

ここで

$$\begin{aligned} \frac{\partial}{\partial x} u \left(t, \frac{x_i + x_{i+1}}{2} \right) &\sim \frac{u_{i+1} - u_i}{x_{i+1} - x_i}, \\ \frac{\partial}{\partial x} u \left(t, \frac{x_{i-1} + x_i}{2} \right) &\sim \frac{u_i - u_{i-1}}{x_i - x_{i-1}} \end{aligned}$$

と近似する。また，図 C.1 より

$$u \left(t, \frac{x_i + x_{i+1}}{2} \right) = \frac{u_i(t) + u_{i+1}(t)}{2}, \quad u \left(t, \frac{x_{i-1} + x_i}{2} \right) = \frac{u_{i-1}(t) + u_i(t)}{2}$$

となるので，(C.52) は

$$= d \left(\frac{u_i(t) + u_{i+1}(t)}{2} \right) \frac{u_{i+1} - u_i}{x_{i+1} - x_i} - d \left(\frac{u_{i-1}(t) + u_i(t)}{2} \right) \frac{u_i - u_{i-1}}{x_i - x_{i-1}} \quad (\text{C.53})$$

となる。(C.51), (C.53) から

$$\frac{d}{dt} u_i(t) = d \left(\frac{u_i(t) + u_{i+1}(t)}{2} \right) \frac{2(u_{i+1} - u_i)}{(x_{i+1} - x_i)(x_{i+1} - x_{i-1})} - d \left(\frac{u_{i-1}(t) + u_i(t)}{2} \right) \frac{2(u_i - u_{i-1})}{(x_i - x_{i-1})(x_{i+1} - x_{i-1})}$$

を得る。ここで，等間隔での差分化 $\Delta x = x_i - x_{i-1} (0 \leq i \leq N)$ を考えると

$$\frac{d}{dt} u_i(t) = d \left(\frac{u_i(t) + u_{i+1}(t)}{2} \right) \frac{u_{i+1} - u_i}{\Delta x^2} - d \left(\frac{u_{i-1}(t) + u_i(t)}{2} \right) \frac{u_i - u_{i-1}}{\Delta x^2} \quad (\text{C.54})$$

となり，非線形拡散方程式に対する差分方程式を得る。 $i=0$ と $i=n$ の場合は，境界条件を考慮することによって

$$\int_{x_0}^{\frac{x_0+x_1}{2}} \frac{\partial}{\partial t} u dx = \frac{d}{dt} u_0(t) \times \frac{x_1 - x_0}{2},$$

²ここでの分割は等間隔の分割である必要性はない。

$$\begin{aligned} \int_{x_0}^{\frac{x_0+x_1}{2}} \frac{\partial}{\partial x} \left(d(u) \frac{\partial u}{\partial x} \right) dx &= d \left(u(t, \frac{x_0+x_1}{2}) \right) \frac{\partial}{\partial x} u(t, \frac{x_0+x_1}{2}) \\ &= d \left(\frac{u_0(t) + u_1(t)}{2} \right) \frac{u_1 - u_0}{x_1 - x_0}. \end{aligned} \quad (\text{C.55})$$

$$\int_{\frac{x_{n-1}+x_n}{2}}^{x_n} \frac{\partial}{\partial t} u dx = \frac{d}{dt} u_n(t) \times \frac{x_n - x_{n-1}}{2},$$

$$\begin{aligned} \int_{\frac{x_{n-1}+x_n}{2}}^{x_n} \frac{\partial}{\partial x} \left(d(u) \frac{\partial u}{\partial x} \right) dx &= -d \left(u(t, \frac{x_{n-1}+x_n}{2}) \right) \frac{\partial}{\partial x} u(t, \frac{x_{n-1}+x_n}{2}) \\ &= -d \left(\frac{u_{n-1}(t) + u_n(t)}{2} \right) \frac{u_n - u_{n-1}}{x_n - x_{n-1}}. \end{aligned} \quad (\text{C.56})$$

ここで, 等間隔の差分化を考えると, $i=0$ のとき

$$\frac{d}{dt} u_0(t) = d \left(\frac{u_0(t) + u_1(t)}{2} \right) \frac{2(u_1 - u_0)}{\Delta x^2}. \quad (\text{C.57})$$

$i=n$ のとき

$$\frac{d}{dt} u_n(t) = -d \left(\frac{u_{n-1}(t) + u_n(t)}{2} \right) \frac{2(u_{n-1} - u_n)}{\Delta x^2} \quad (\text{C.58})$$

となり, 差分法による近似と同じ差分方程式を得る.

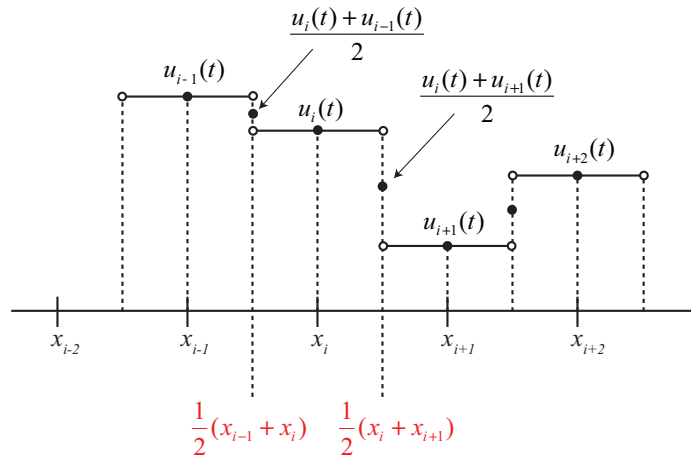


図 C.1: 有限体積法を用いるための区分定数近似

空間 2 次元問題や空間 3 次元問題でも同様の方法で導出することができる。

付録D 並列化編の完成部分

D.1 ADI法

ADI法のOpenMPによる並列化について解説する。ここのは空間2次元反応拡散系に対するADI法について並列化を行う。ADI法について、アルゴリズムやプログラム、その他の詳しいことについては「私にとっての”反応拡散方程式数値計算入門”」のADI法の項を参考にして下さい。ADIのアルゴリズムは以下の通りです。

ADI法のアルゴリズム

行列 A, B を設定する。

行列 A, B について、 LU 分解を行う。

$k = 1, 2, \dots, M$ (指定した計算回数) の順に、以下の手順を繰り返す。

※ここに並列化指示文を挿入する。

$j = 1, 2, \dots, N_y$ の順に、以下の手順を繰り返す。

ここに並列化指示文を挿入しては行けません。

(各 j に対してスレッドの起動時間が必要になるので、計算が速くなりません)。

$i = 1, 2, \dots, N_x$ の順に、以下の手順を繰り返す。

j について、 \mathbf{b}_j を場合分けする。

$A\mathbf{u}_j^{k+\frac{1}{2}} = \mathbf{b}_j$ を解く。

求めた $\mathbf{u}_j^{k+\frac{1}{2}}$ を新しい \mathbf{b}_j として置きなおす。

ここに※並列化指示文を挿入する。

$i = 1, 2, \dots, N_x$ の順に、以下の手順を繰り返す。

ここに並列化指示文を挿入しては行けない (計算が速くなりません)。

(各 j に対してスレッドの起動時間が必要になるので、計算が速くなりません)。

$j = 1, 2, \dots, N_y$ の順に、以下の手順を繰り返す。

i について、 \mathbf{b}_i を場合分けする。

$B\mathbf{u}_i^{k+1} = \mathbf{b}_i$ を解く。

求めた \mathbf{u}_i^{k+1} を新しい \mathbf{b}_i として置きなおす。

ループの範囲であるが、上記では N_x もしくは N_y までとしているが、これは境界条件によって異なるので注意すること。

OpenMPにより並列化プログラムに書き換える場合は、※の部分の指示に従って書き換える。(書き換えない場合は、※の部分は無視すればよい)

※の部分に入れる並列化指示文については

```
#pragma omp parallel for private(*) shared(**)
```

と書けばよい。*や**の部分には、それぞれのプライベート変数や共有変数を入れる。詳しくは OpenMP の資料を調べてください [25, 26, 27].

D.1.1 並列化された ADI 法を用いたプログラムの例

ここでは 2 変数反応拡散系

$$\begin{cases} \frac{\partial u}{\partial t} = d_u \Delta u + \frac{1}{\varepsilon} (u(1-u)(u-a) - v), \\ \frac{\partial v}{\partial t} = d_v \Delta v + u - \gamma v. \end{cases} \quad (\text{D.1})$$

に対する並列計算プログラムの例を提示する。コンパイル方法は次の通りです。

Mac 上で GCC を使ったコンパイル方法

```
> /opt/homebrew/bin/gcc-13 -fopenmp -O3 sample-omp01.c -o a.out01-omp -lm
```

Mac 標準の GCC では OpenMP がサポートされていないので、homebrew を使って GCC をインストールする必要がある。

Linux 上で ICC を使ったコンパイル方法

```
> icc -qopenmp -O3 sample-omp01.c -o a.out01-omp -lm
```

計算機上で起動するスレッド数を設定する方法は次の通りです。

数値計算時のスレッド数の指定

```
> export OMP_NUM_THREADS=4 (スレッド数)
```

この設定を忘れると 1 スレッドで計算されることになるので注意してください。

次のように、コンパイル、スレッド数の設定、実行を 1 つのスクリプトファイル (omp_run.sh) にするのが便利です。例えば、スクリプトファイルの中に次のように書いておけばよいと思います。

omp_run.sh

```
#sh
rm ./a.out-omp01
/opt/homebrew/bin/gcc-13 -fopenmp -O3 sample-omp01.c -o a.out01-omp -lm
export OMP_NUM_THREADS=4 (スレッド数)
time ./a.out-omp01
```

以下では私の書いた OpenMP を使った ADI 法のサンプルプログラムを掲載します。OpenMP を使い始めて 2 日間で書いたサンプルプログラムですので、改良の余地は十分あると思いますので、お気づきの方はお知らせください。

```
/* ===== */
/*          Include header files          */
/* ===== */
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <omp.h>
```

```

#include <sys/types.h>
#include <sys/stat.h>

/* ===== */
/*           Parameter set                               */
/* ===== */
#define MAXSTEP    10000    /*           */
#define SOUTPUT    100     /* SOUTPUT  */
#define dt         0.001   /*           */
#define Nx         400     /*           */
#define Ny         400     /*           */
#define Lx         20.0    /*           */
#define Ly         20.0    /*           */
#define du         1.0     /*           */
#define dv         0.1     /*           */
#define eps        0.002   /*           */
#define aa         0.125   /*           */
#define gamma      4.0     /*           */
#define Mx         Nx-1   /*           */
#define My         Ny-1   /*           */
#define dmod       2      /*           */
#define Restart    0      /*           */
/*===== */
/*           Initial data                               */
/* ===== */
void initial_condition( double dx, double dy, double u[][Ny+1], double v[][Ny+1] )
{
    int i, j;
    FILE *fp;
    double mx, my;

#pragma omp for private(j)
    for( i = 0; i <= Nx; i++ ){
        for( j = 0; j <= Ny; j++ ){
            u[i][j] = 0.0;
            v[i][j] = 0.0;
        }
    }

#pragma omp for private(i)
    for( j = 0; j <= Ny/2; j++ ){
        for( i = Nx/2 - 10; i <= Nx/2; i++ ){
            u[i][j] = 1.0;
        }
        for( i = Nx/2 - 20; i <= Nx/2 - 10; i++ ){
            v[i][j] = 0.5;
        }
    }
}

```

```

}
}

/* ===== */
/*      非線形項      */
/* ===== */
void reaction_term(double u[][Ny+1], double v[][Ny+1], double f[][My+1], double g[][My+1])
{
    int i, j;

#pragma omp parallel for private(j)
    for( i = 0; i <= Mx; i++){
        for( j = 0; j <= My; j++){
            f[i][j] = (u[i][j]*(1.0-u[i][j])*(u[i][j] - aa) - v[i][j] ) / (eps);
            g[i][j] = u[i][j] - gamma * v[i][j];
        }
    }
}

/* ===== */
/*      File Open      */
/* ===== */
FILE *fopen_w( char filename[] )
{
    FILE *fp;
    if( (fp=fopen(filename,"w")) == NULL ){
        printf( "cannot open output file\n" );
        exit(1);
    }
    return fp;
}

FILE *fopen_r( char filename[] )
{
    FILE *fp;
    if( (fp=fopen(filename,"r")) == NULL ){
        printf( "cannot open output file\n" );
        exit(1);
    }
    return fp;
}

/* ===== */
/*      初期値の読み込み      */
/* ===== */
void read_initial_dat(double u[][Ny+1], double v[][Ny+1], FILE *fp)
{
    int i, j;

```



```

    for(j = 0; j <= Ny; j+=1){
        for(i = 0; i <= Nx; i+=1){
            fscanf(fp, "%lf %lf", &u[i][j], &v[i][j] );
        }
    }
}
/* ===== */
/*          Date Output                                */
/* ===== */
void output(double dx, double dy, double time, double u[][Ny+1], double v[][Ny+1],
FILE *fp_data)
{
    int i, j;
    for( i = 0; i <= Nx; i+=dmod ){
        for( j = 0; j <= Ny; j+=dmod ){
            fprintf(fp_data, "%20.15E %20.15E %20.15E %20.15E %20.15E\n",
time, i*dx, j*dy, u[i][j], v[i][j]);
        }
    }
    fflush(stdout);
}
/* ===== */
/*   解の更新                                         */
/* ===== */
void update_sol_period( double u[][Ny+1], double new_u[][My+1] )
{
    int i, j;
#pragma omp parallel for private(j)
    for( i = 0; i <= Nx-1; i++ ){
        for( j = 0; j <= Ny-1; j++ ){
            u[i][j] = new_u[i][j];
        }
    }
    i = Nx;
#pragma omp parallel for
    for(j = 0; j <= Ny; j++){
        u[i][j] = new_u[0][j];
    }
    j = Ny;
#pragma omp parallel for
    for(i = 0; i <= Nx; i++){
        u[i][j] = new_u[i][0];
    }
}
/* ===== */
/*          LU solve for OpenMP                        */
/* ===== */

```

```

/* ===== */
void lu_solve_period_OMP( int i, const int M, double L[][M+1], double U[][M],
double b[][My+1], double x[][My+1] )
{
    int j;
    double sum;

    b[i][0] /= L[0][0];
    for( j = 1; j <= M-1; j++ ){
        b[i][j] = (b[i][j]-L[1][j]*b[i][j-1])/L[0][j];
    }
    sum = 0.0;
    for( j = 0; j <= M-1; j++ ){
        sum += L[2][j]*b[i][j];
    }
    x[i][M] = (b[i][M]-sum)/L[2][M];
    x[i][M-1] = b[i][M-1]-U[1][M-1]*x[i][M];
    for( j = M-2; j >= 0; j-- ){
        x[i][j] = b[i][j]-U[0][j]*x[i][j+1]-U[1][j]*x[i][M];
    }
}
/* ===== */
/*          ADI method 2 for OpenMP          */
/* ===== */
void adi2_period( double d, double r, double u[][Ny+1], double f[][My+1],
double L[][My+1], double U[][My], double new_u[][My+1] )
{
    int i, j, k;
    static double RS[Mx+1][My+1], s[Mx+1][My+1];
    i = 0;
#pragma omp parallel for
    for( j=0; j <= My; j++ ){
        RS[i][j] = u[i][j]+r*d*(u[i-1+Nx][j]-2.0*u[i][j]+u[i+1][j])+ 0.5*dt*f[i][j];
    }

#pragma omp parallel for
    for( j = 0; j <= My; j++ ){
        s[i][j] = u[i][j];
    }
    lu_solve_period_OMP( i, My, L , U, RS, s);

#pragma omp parallel for
    for( j = 0; j <= My; j++ ){
        new_u[i][j] = s[i][j];
    }
}

```

```

#pragma omp parallel for private(j, k)
  for( i = 1; i <= Mx; i++ ){
    for( j = 0; j <= My; j++ ){
      RS[i][j] = u[i][j]+r*d*(u[i-1][j]-2.0*u[i][j]+u[i+1][j])+ 0.5*dt*f[i][j];
    }
    for( j = 0; j <= My; j++ ){
      s[i][j] = u[i][j];
    }
    lu_solve_period_OMP( i, My, L , U, RS, s);
    for( k = 0; k <= My; k++ ){
      new_u[i][k] = s[i][k];
    }
  }
}
/* ===== */
/*          ADI method 1 for OpenMP          */
/* ===== */
void ad1_period( double d, double r, double u[][Ny+1], double f[][My+1],
double L[][Mx+1], double U[][Mx], double new_u[][My+1] )
{
  int    i, j, k;
  static double RS[My+1][Mx+1], s[My+1][Mx+1];

  j = 0;
#pragma omp parallel for
  for( i=0; i <= Mx; i++ ){
    RS[j][i] = u[i][j]+r*d*(u[i][j-1+My]-2.0*u[i][j]+u[i][j+1])+ 0.5*dt*f[i][j];
  }

#pragma omp parallel for
  for( i = 0; i <= Mx; i++ ){
    s[j][i] = u[i][j];
  }
  lu_solve_period_OMP(j, Mx, L , U, RS, s);

#pragma omp parallel for
  for( i = 0; i <= Mx; i++ ){
    new_u[i][j] = s[j][i];
  }

#pragma omp parallel for private(i, k)
  for( j = 1; j <= My; j++ ){
    for( i = 0; i <= Mx; i++ ){
      RS[j][i] = u[i][j]+r*d*(u[i][j-1]-2.0*u[i][j]+u[i][j+1])+ 0.5*dt*f[i][j];
    }
    for( i = 0; i <= Mx; i++ ){

```

```

        s[j][i] = u[i][j];
    }
    lu_solve_period_OMP(j, Mx, L, U, RS, s);
    for( k = 0; k <= Mx; k++ ){
        new_u[k][j] = s[j][k];
    }
}
}
/* ===== */
/*          周期境界条件下での行列のLU分解          */
/* ===== */
void lu_decomp_period( const int M, double a[], double b[], double c[],
double L[][M+1], double U[][M] )
{
    int    i;
    double sum;
    L[0][0] = a[0];
    U[0][0] = c[0] / L[0][0];
    U[1][0] = b[0] / L[0][0];
    for( i = 1; i<= M-2; i++ ){
        L[1][i] = b[i];
        L[0][i] = a[i]-U[0][i-1]*L[1][i];
        U[0][i] = c[i]/L[0][i];
        U[1][i] = -L[1][i]*U[1][i-1]/L[0][i];
    }
    i = M-1;
    L[1][i] = b[i];
    L[0][i] = a[i]-U[0][i-1]*L[1][i];
    U[1][i] = (c[i]-L[1][i]*U[1][i-1])/L[0][i];
    L[2][0] = c[M];
    for( i = 1; i<= M-2; i++ ){
        L[2][i] = -U[0][i-1]*L[2][i-1];
    }
    i = M-1;
    L[2][i] = b[M]-U[0][i-1]*L[2][i-1];
    sum = 0.0;
    for( i = 0; i<= M-1; i++ ){
        sum += L[2][i]*U[1][i];
    }
    L[2][M] = a[M]-sum;
}
/* ===== */
/*          Input matrix          */
/* ===== */
void matrix_period( int M, double rd, double a[], double b[], double c[] )
{

```

```

    int i;
#pragma omp parallel for
    for( i = 0; i <= M; i++ ){
        b[i] = - rd;
        a[i] = 1.0+2.0*rd;
        c[i] = - rd;
    }
}
/* ===== */
/*          main program          */
/* ===== */
int main( int argc, char *argv[] )
{
    FILE *fp_data, *fp_final, *fp_read;
    char filename1[100], filename2[100], filename3[100];
    int    step, i, j;
    double ax[Mx+1], bx[Mx+1], cx[Mx+1];
    double ay[Mx+1], by[Mx+1], cy[Mx+1];
    static double u[Nx+1][Ny+1], new_u[Mx+1][My+1], f[Mx+1][My+1];
    static double v[Nx+1][Ny+1], new_v[Mx+1][My+1], g[Mx+1][My+1];
    double Lu1[3][Mx+1], Uu1[2][Mx], Lu2[3][My+1], Uu2[2][My];
    double Lv1[3][Mx+1], Uv1[2][Mx], Lv2[3][My+1], Uv2[2][My];
    double dx = Lx/(double) Nx;
    double dy = Ly/(double) Ny;
    double ri1 = 0.5*dt/(dx*dx);
    double re1 = 0.5*dt/(dy*dy);
    double ri2 = re1;
    double re2 = ri1;

    sprintf( filename1, "results.dat");
    sprintf( filename2, "final.dat");
    fp_data = fopen_w(filename1);
    fp_final = fopen_w(filename2);
    step = 0;

    if(Restart == 0){
        initial_condition(dx, dy, u, v);
    }
    if(Restart == 1){
        sprintf( filename3, "initial.dat");
        fp_read = fopen_r(filename3);
        read_initial_dat(u, v, fp_read);
        fclose(fp_read );
    }
    output(dx, dy, step*dt , u, v, fp_data);
    fflush(fp_data);
}

```

```

/* for variable U */
matrix_period( Mx, du*ri1, ax, bx, cx ); /* */
lu_decomp_period( Mx, ax, bx, cx, Lu1, Uu1 ); /* LU */
matrix_period( My, du*ri2, ay, by, cy ); /* */
lu_decomp_period( My, ay, by, cy, Lu2, Uu2 ); /* LU */

/* for variable V */
matrix_period( Mx, dv*ri1, ax, bx, cx );
lu_decomp_period( Mx, ax, bx, cx, Lv1, Uv1 ); /* LU */
matrix_period( My, dv*ri2, ay, by, cy ); /* */
lu_decomp_period( My, ay, by, cy, Lv2, Uv2 ); /* LU */
for( step = 1; step <= MAXSTEP; step++ ){
    reaction_term( u, v, f, g);
#pragma omp sections
{
    #pragma omp section
    adi1_period( du, re1, u, f, Lu1, Uu1, new_u ); /* ADI 1 */
    #pragma omp section
    adi1_period( dv, re1, v, g, Lv1, Uv1, new_v ); /* ADI 1 */
}
#pragma omp sections
{
    #pragma omp section
    update_sol_period( u, new_u ); /* uの更新 */
    #pragma omp section
    update_sol_period( v, new_v ); /* vの更新 */
}
    reaction_term(u, v, f, g);
#pragma omp sections
{
    #pragma omp section
    adi2_period( du, re2, u, f, Lu2, Uu2, new_u ); /* ADI 2 */
    #pragma omp section
    adi2_period( dv, re2, v, g, Lv2, Uv2, new_v ); /* ADI 2 */
}
#pragma omp sections
{
    #pragma omp section
    update_sol_period( u, new_u ); /* uの更新 */
    #pragma omp section
    update_sol_period( v, new_v ); /* vの更新 */
}
    if( step%SOUTPUT == 0 ){ /* 計算途中の結果出力 */
        output(dx, dy, step*dt , u, v, fp_data);
    }
}

```

```

}
for( j = 0; j <= Ny; j++ ){ /* 最終データ出力 */
    for( i = 0; i <= Nx; i++){
        fprintf( fp_final, "%20.15E %20.15E\n", u[i][j], v[i][j]);
    }
}
}
fflush(fp_final);
fflush(fp_data);
fclose(fp_data );
fclose(fp_final);
return 0;
}

```

D.1.1.1 並列化とその効率について

並列化するループについて、二重ループの内側ループを並列化することは容易であるが、スレッドの起動等で内側ループの並列化は効率が悪い。そこで外側ループを並列化することにした。注意すべきことは、内側のループ内にある、 j について右辺 b の場合分けの箇所、外側のループに関しての繰り返し間の依存関係があるために、並列化した場合解がまったく違うものになってしまうことである。以下は、並列化したプログラムを M3 チップの MacOS 上で gcc によりコンパイルした後に実行し、実行開始から終了までにかかった時間を測定したものである。

1 スレッド	2 スレッド	4 スレッド	8 スレッド
72.670s	44.194	29.783s	33.042s

1 スレッドでの計算時にかかった時間を 1 とすると、それぞれにかかった時間は以下のように表せる。

1 スレッド	2 スレッド	4 スレッド	8 スレッド
1.00	0.608	0.410	0.455

このことから、並列化したプログラムの並列化効率以下の通りである。

1 スレッド	2 スレッド	4 スレッド	8 スレッド
1.00	1.64	2.44	2.2

以下は Z8G4 上の LinuxOS 上で gcc によりコンパイルした後に実行し、実行開始から終了までにかかった時間を測定したものである。

1 スレッド	2 スレッド	4 スレッド	8 スレッド	16 スレッド
100.605s	68.080s	38.616s	23.669s	16.883s

1CPU での計算時にかかった時間を 1 とすると、それぞれにかかった時間は以下のように表せる。

1 スレッド	2 スレッド	4 スレッド	8 スレッド	16 スレッド
1.00	0.676	0.384	0.235	0.168

このことから、並列化したプログラムの並列化効率以下の通りである。

1 スレッド	2 スレッド	4 スレッド	8 スレッド	16 スレッド
1.00	1.478	2.61	4.25	5.96

今回は数値計算規模の比較的小さい計算であり、その場合は4スレッド程度が比較的有效であると考えられる。4スレッドでの数値計算を計算機の最大スレッド数を超えない程度で計算することをオススメする。計算規模が大きくなる場合は再度検討する必要があるので、数値計算を開始する前にどの程度の並列化を行うか検討してください。

今後は、iccを使った場合やZ8G5上でどの程度速くなるのか検討する。

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] “<http://www-mmc.es.hokudai.ac.jp/masakazu/>”
- [24] “<http://nalab.mind.meiji.ac.jp/mk/labo/text/glsc3d.pdf> ”
- [25] “OpenMP による並列プログラミングと数値計算法”, 牛島 省, 丸善株式会社, 2006
- [26] “並列プログラミング入門: サンプルプログラムで学ぶ OpenMP と OpenACC”, 片桐 孝洋, 東京大学出版会, 2015

- [27] “<https://www.cc.kyushu-u.ac.jp/scp/doc/users/lecture/2019/openmp-2019.pdf>”
- [28] “非平衡系の科学 I”, 北原 和夫, 吉川 研一, 講談社.
- [29] “非平衡系の科学 III”, 三池 秀敏, 森 義仁, 山口 智彦, 講談社.
- [30] “非線形科学”, 吉川 研一, 学会出版センター.

おわりに

2012年4月1日から北海道大学電子科学研究所に異動しました。本解説書は、北海道大学 電子科学研究所 附属社会創造数学研究センター 人間数理研究分野で配布している反応拡散数値計算の入門書です。本書は入門編，発展編，並列計算編の3編から成っていますが，現在のところ完成しているのは入門編だけであり，発展編，並列計算編の完成には到っていません¹。基本的に非公開の入門書でしたが，金沢大学に所属していたときに，いろいろな事情²があって一般に公開することになりました。非公開文書を公開したからといって真面目に修正をしているわけではないので，多くの誤植や間違いがあると思います。一部修正しましたが，まだまだ間違いがあると思いますので，ご指摘して頂ければ修正したいと思います。

数値計算結果の可視化部分はかなり古くなってきたので，少し更新しました。全面改定には至っていませんが，Linux，Macを使って数値計算するためには十分であると思います。Windows上での数値計算ですが，最近は個人使用だとVMWearが無償で使えるので，VMWearをインストールしてWindows上でLinuxを使うことをお勧めします。Windows10以降ではWindows上でLinuxを動かすことができます。詳しくは<https://qiita.com/ayatokura/items/e89c8360cc96f25c434a>等をみるとよいと思います。私は最近Windowsを使っていないので，使用感はわかりません。

2018年5月，ffmpegのインストール部分を書き換えました。

2021年3月，大学院生だった岡本守さんの協力を得て，第5章可視化のパートを書き換えました。

2024年3月，第5章可視化パートで，MacOSで再生できる動画としてMpeg1の古い形式がサポートされなくなったため，Mpeg4形式で動画を作成するように書き換えました。

2024年3月，最近スレッド数が非常に多いCUPが出てきて，ある程度並列化して計算した方が効率的にパラメータ探索ができるようになってきたと感じたので，OpenMPを用いたADI法の計算について追記しました。並列化編の一部となっています。

2024年3月，連立1次方程式の反復解法を加えました。陰解法に対するOpenMPを使った並列化には必要になると考えています。

2024年4月

長山 雅晴

¹完成しない可能性大です

²さきがけ数学塾で使用することもその一つです。