熱方程式に対する差分法 I

— 区間における熱方程式 —

桂田 祐史

1998 年〜 2024 年 9 月 14 日

目 次

序

この文書では、開区間における熱伝導方程式の初期値境界値問題の差分法による解法を扱う。

他の文書の紹介

1. 「発展系の数値解析」
   https://m-katsurada.sakura.ne.jp/labo/text/heat-fdm-0.pdf
   放送大学の講義科目「応用数学」 (藤田宏担当, 私も差分法のところをお手伝いしました) のテキストの一部です。1 次元区間における熱方程式の初期値境界値問題に対する差分法の入門的解説です。
   もともとは藤田先生と共同で担当していた卒研の内容から、適当な部分を抜き出したものです。最初に参考にしたのは、菊地文雄・山本昌宏「微分方程式と計算機演習」山海堂、という本です。

2. 「『発展系の数値解析』に加えること」
   https://m-katsurada.sakura.ne.jp/labo/text/heat-fdm-0-add.pdf
   1 はそのまま卒研のテキストとして使うには不便なところがあるので、少し補ったものです。LU 分解や安定性を解析するための行列法などを説明してあります。

3. 「熱方程式に対する差分法 I — 区間における熱方程式 —」
   https://m-katsurada.sakura.ne.jp/labo/text/heat-fdm-1.pdf
   この文書です。2 次元の区間、つまり長方形領域における熱方程式を扱っています。

4. 「熱方程式に対する差分法 II — 円盤領域、円柱領域、球における熱方程式 —」
   https://m-katsurada.sakura.ne.jp/labo/text/heat-fdm-2.pdf
   ここはまだ基本的に工事中です。

熱方程式ではないですが、

1. 「Poisson 方程式に対する差分法」
   https://m-katsurada.sakura.ne.jp/labo/text/poisson.pdf
   2 次元長方形領域における Poisson 方程式の差分解の厳密解への収束証明 (これは内容的には、F. John の本を読み解いたものです) と、プログラムなどを載せています。

2. 「波動方程式に対する差分法」
   https://m-katsurada.sakura.ne.jp/labo/text/wave.pdf
   1 次元波動方程式の初期値境界値問題に対する差分法の収束証明などを書いてあります。

なお、プログラムの多くは、

「公開プログラムのページ」
https://m-katsurada.sakura.ne.jp/program/

に置いてあります。

私の研究室周辺での差分法による熱方程式とのお付き合い

卒業研究レポートについては、https://m-katsurada.sakura.ne.jp/labo/report/ を見よ。

1994 年度の桂田ゼミの卒業研究で、初めて空間 2 次元の熱方程式の初期値境界値問題を解くプログラムが出来た (by 志村結城、山口尚人)。彼らのプログラムは行列の対称化は行わずに、LU 分解を行って解くものであった。バンド幅は考慮に入れないなど、素朴なプログラムである。はじめて目にした実行結果はなかなか印象的であった。最初に LU 分解をするため、第 1 段の結果が出るまでかなり時間がかかる。しかしその後は軽快にステップが進む。彼らは Dirichlet 境界値問題のみならず、Neumann 境界値問題も扱っている。

また 1994 年度修士論文の研究では、空間 2 次元の楕円型境界値問題 (定常移流拡散方程式) が解かれている (by 上野 康浩)。

1995 年度の桂田ゼミの卒業研究のテーマは有限要素法だったが、有限要素法による解と比較をするため、熱方程式を差分法で解くプログラムが作られた (有限要素法版はなかなかの力作 — by 川崎 純也)。

1995 年度の修士論文の研究では、領域分割法の研究のため、やはり空間 2 次元のポアソン方程式の境界値問題が差分法で解かれている (by 長坂吉晃、小張朝子)。

以後は割と気軽にチャレンジできるようになった (やはりノウハウの蓄積は大きい)。

1996 年度の桂田ゼミの卒業研究では、ADI 法をキーワードに色々な数値実験を行なった (爆発問題 — by 吉澤勇一, 西澤智恵子, 平均曲率流 — by 平謙一)。また円盤領域、円環領域などにおける熱伝導方程式に取り組んだ学生 (松本英久) がいた。

1997 年度以降も説明しておきたい結果があるが、それについては、上記 WWW ページを参照してもらいたい。

これから

- MATLAB 等のソフトウェアでどこまで効率的なプログラムがかけるのか。
- von Neumann の安定性解析をきちんと取り扱いたい (有名だが、いいかげんな書き方をされていることが多いと感じている…多分ちゃんと書けると思う…時間がない…)。
- Java によるシミュレーション・プログラムをきちんとした形で提示したい。
- 円柱領域または球における熱方程式にチャレンジしてみたい。
  (円盤で遭遇した問題がここでも出て来るようだが、安定性の問題、解決できるのか？？)
- 2 次元以上の領域でどういうアルゴリズムがお奨めなのか？ (1 次元ならば θ 法で OK だが、2 次元だと θ 法は損みたい、ADI 法か？ 熱方程式ならばそれで良いが移流項が入ったりすると？)

第1章　同次 Dirichlet 境界条件以外の境界条件 (1次元)

(誤植が発見されるのはいつものことであるけれど、とてもたくさん見つけてくれた Y.S. 君に感謝する。)

1.1　はじめに

「発展系の数値解析」では、同次 Dirichlet 境界条件を課した 1 次元熱方程式の初期値境界値問題

$$\begin{aligned} u_t(x,t) &= u_{xx}(x,t) \quad (t>0,\, 0<x<1) \\ u(0,t) &= u(1,t) = 0 \quad (t>0) \\ u(x,0) &= f(x) \quad (0 \le x \le 1) \end{aligned}$$

に対する差分方程式は提示したが、Neumann 境界条件の場合などについては、詳しいことは述べなかった。

ここでは、

(1) 同次 Neumann 境界条件

(2) 非同次 Dirichlet 境界条件

(3) 非同次 Neumann 境界条件

(4) 第 3 種境界条件 (Robin 境界条件)

の 4 つの場合の差分方程式について述べる ((1) はカットというか、(3) に吸収させるのが良いかも)。

卒研で取り組ませてみると:　これらの差分方程式を求めて数値実験させるのは、手頃な良い演習と思うのだが、事前に注意しておかないと、思いの外手こずる人が多い。大抵の場合は、きちんとした差分方程式を書き下す前にプログラムの書き直しに取り掛かり、行き詰まってしまっている。数値計算のプログラミングでは、コンピューターの前に座るのに先だって、紙の上で差分方程式を完成させるのが鉄則である (この種のことを記述するには、数学語が一番であるから)。 — とは言っても、実際にハマルまでは、このことはなかなか分からないものらしい (いつものように)。

2024〜2025 年改善目標　久しぶりに差分法プログラムを使う必要が生じてこの文書を見返して見たら、色々と不行き届きな点が見つかった。

(1) プログラムのうち改訂したものがあるが、それがこの文書には入っていないし、探しにくい。

(2) プログラムをコンパイルするのに手間がかかる場合がある。

(3) C よりは MATLAB, Python という人が増えた。そういう言語を使えばより使いやすくなる。(C+GLSC という環境は用意するのが大変と感じる人が多いだろうが、MATLAB や Python, Julia 等ならばデフォールトの環境で済む。) そういうプログラムはすでに世の中に絶対あるはずだと思うが、意外と探しにくい (学生がへんてこりんな Python プログラムの載ったレポートを提出して、「探さなかったのかな？」と訝しく思ったけれど、自分で探そうとしたら、見つけられなかった。)。

(4) 新しい文献が出ているが、そういう情報が書いていない。

(5) 常識的なことで書いていないことがある。「○○を見よ」位は書いておこう。

(6) 自分が分からないことはそのままにしてあるけれど、こういうことが分からないと書いておく方が良いかもしれない。

(7) 何をしようと考えているのか、自分のスタンス (立ち位置) のようなことを書いていないが、それは書いておくべきかもしれない。

(8) 差分解の公式を書いてみる (4 章「差分法の安定性と収束性」の 4.1.1 に書いた理由)。

できる限り、自分にとって必要なところから手を入れてみるつもり。どこまで出来るかは分からない。

1.2 数値実験の手引き

1.2.1 参考となるプログラム

熱方程式の初期値境界値問題を差分法で解くプログラムをいくつか「公開プログラムのページ」[1] で公開している[2]。基本的なものは次の二つである。

heat1d-i-glsc.c 同次 Dirichlet 境界条件のもとでの熱方程式の初期値境界値問題を θ 法で解くプログラム。(「発展系の数値解析」[3] で説明した、差分方程式を解いている。菊地・山本 [1] に掲載されていたプログラムが元になっている。)

heat1n-i-glsc.c 同次 Neumann 境界条件のもとでの熱方程式の初期値境界値問題を θ 法で解くプログラム。境界条件の離散化には、以下の 1.4 で説明する「仮想格子点の方法」を用いている。

[1] https://m-katsurada.sakura.ne.jp/program/

[2] 可視化のために (X11 環境下で利用できる) グラフィックス・ライブラリィ **GLSC** (ftp://ftp.st.ryukoku.ac.jp/pub/ryukoku/software/math/) を利用しているが、明治大学数学科の計算機室 (6701) のコンピューター (Linux, Cygwin, Solaris) や、学生貸し出し用のパソコンの多く (Cygwin, Knoppix Math) にインストール済みである。

[3] https://m-katsurada.sakura.ne.jp/labo/text/heat-fdm-0.pdf

1.2.2 実験の方針

最初はこの二つのプログラムを使って、『発展系の数値解析』にある例を再現する方針で実験するのがよいであろう。(行列の LU 分解と、それを用いて連立 1 次方程式を解く方法については、「『発展系の数値解析』に加えること」[4] を見よ。`trilu()`, `trisol()` の理解を後回しにして、最初はブラックボックスとして扱うことも出来るであろう。)

それが一段落したら、この章の説明を参考に以下のようなプログラムを作成して、実験するとよい。

1. 非同次 Dirichlet 境界条件のプログラム。1.3 を参考に、同次条件の場合の `heat1d-i-glsc.c` を元にして作る。
2. 非同次 Neumann 境界条件のプログラム (境界条件を「素朴に」近似する方法)。1.4.2 を参考に、同次 Dirichlet 条件の場合の `heat1d-i-glsc.c` を元にして作る。
3. 非同次 Neumann 境界条件のプログラム (境界条件を仮想格子点を導入して近似する方法)。1.4.3 を参考に、同次 Neumann 条件の場合の `heat1n-i-glsc.c` を元にして作る。

1.2.3 計算のチェック

計算がうまく行っていることの一つのチェック法として、解の漸近挙動を見ることがある。

(a) 同次 Dirichlet 境界条件 $(u(0,t) = A,\ u(1,t) = B)$ の場合は、

$$\lim_{t\to\infty} u(x,t) = A + (B-A)x \quad (x \in [0,1])$$

が成り立つか？

(b) 非同次 Neumann 境界条件 $(u_x(0,t) = A,\ u_x(1,t) = B)$ で $A = B$ の場合、

$$\lim_{t\to\infty} u(x,t) = Ax + C \quad (x \in [0,1])$$

が成り立つか？ ここで C は定数であるが、具体的な値を求めることは読者の演習とする。

(c) 非同次 Neumann 境界条件で $A \neq B$ の場合、

$$u(x,t) \sim (B-A)t \quad (t \to \infty)$$

が成り立っているか？

(d) (余裕がある場合) 厳密解が具体的に分かるような、初期条件が単純な場合に計算して、誤差を実際に測り、N の増加とともにどのように減衰するか調べる。非同次 Neumann 境界条件の場合に、境界条件の近似法によって結果がどう変るか詳しく調べる。

[4] `https://m-katsurada.sakura.ne.jp/labo/text/heat-fdm-0-add.pdf`

1.3 非同次 Dirichlet 境界条件

1.3.1 問題の設定

$$u_t(x,t) = u_{xx}(x,t) \quad (t>0,\ 0<x<1), \tag{1.1}$$

$$u(0,t) = A, \quad u(1,t) = B \quad (t>0), \tag{1.2}$$

$$u(x,0) = f(x) \quad (0 \le x \le 1) \tag{1.3}$$

を考える。ここで A, B は $A = B = 0$ とは限らない定数である。(1.2) を**非同次 Dirichlet 境界条件**と呼ぶ。

「微分方程式 2」で、この問題の厳密解 u の公式と、その漸近挙動について説明してある。

1.3.2 差分方程式

熱方程式 (1.1) から

$$(1+2\theta\lambda)U_i^{n+1} - \theta\lambda(U_{i+1}^{n+1} + U_{i-1}^{n+1}) = [1-2(1-\theta)\lambda]U_i^n + (1-\theta)\lambda(U_{i+1}^n + U_{i-1}^n) \tag{1.4}$$
$$(n = 0, 1, \cdots;\ i = 1, 2, \cdots, N-1)$$

を導き、初期条件 (1.3) から

$$U_i^0 = f(x_i) \quad (i = 0, 1, \cdots, N)$$

を導くのはこれまでと同じで、問題は境界条件 (1.2) をどう扱うかである。

これは特に難しくはなく、

$$U_0^{n+1} = A, \quad U_N^{n+1} = B$$

とするのが自然な扱いである。

(1.4) で $i = 1$ とした式

$$(1+2\theta\lambda)U_1^{n+1} - \theta\lambda(U_2^{n+1} + U_0^{n+1}) = [1-2(1-\theta)\lambda]U_1^n + (1-\theta)\lambda(U_2^n + U_0^n)$$

に $U_0^{n+1} = A$ を代入して整理すると

$$(1+2\theta\lambda)U_1^{n+1} - \theta\lambda U_2^{n+1} = [1-2(1-\theta)\lambda]U_1^n + (1-\theta)\lambda(U_2^n + U_0^n) + A\theta\lambda$$

を得る。

同様に (1.4) で $i = N-1$ とした式

$$(1+2\theta\lambda)U_{N-1}^{n+1} - \theta\lambda(U_N^{n+1} + U_{N-2}^{n+1}) = [1-2(1-\theta)\lambda]U_{N-1}^n + (1-\theta)\lambda(U_N^n + U_{N-2}^n)$$

に $U_N^{n+1} = B$ を代入して整理すると

$$(1+2\theta\lambda)U_{N-1}^{n+1} - \theta\lambda U_{N-2}^{n+1} = [1-2(1-\theta)\lambda]U_{N-1}^n + (1-\theta)\lambda(U_N^n + U_{N-2}^n) + B\theta\lambda$$

を得る。

こうして、$N-1$ 個の未知数 $\{U_i^{n+1}\}_{i=1,2,\cdots,N-1}$ に関する $N-1$ 個の連立 1 次方程式を得る。ベクトルと行列を用いて表わすと

$$\mathcal{A}\boldsymbol{U}^{n+1} = \boldsymbol{b}^n. \tag{1.5}$$

ただし

$$\mathcal{A} = \begin{pmatrix} 1+2\theta\lambda & -\theta\lambda & & & & & & \\ -\theta\lambda & 1+2\theta\lambda & -\theta\lambda & & & & & \\ & \ddots & \ddots & \ddots & & & & \\ & & -\theta\lambda & 1+2\theta\lambda & -\theta\lambda & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & -\theta\lambda & 1+2\theta\lambda & -\theta\lambda & \\ & & & & & -\theta\lambda & 1+2\theta\lambda & \end{pmatrix},$$

$$\boldsymbol{U}^{n+1} = \begin{pmatrix} U_1^{n+1} \\ U_2^{n+1} \\ \vdots \\ U_{i-1}^{n+1} \\ U_i^{n+1} \\ U_{i+1}^{n+1} \\ \vdots \\ U_{N-2}^{n+1} \\ U_{N-1}^{n+1} \end{pmatrix},$$

$$\boldsymbol{b}^n = \begin{pmatrix} [1-2(1-\theta)\lambda]\,U_1^n & + & (1-\theta)\lambda(U_2^n + U_0^n) \\ [1-2(1-\theta)\lambda]\,U_2^n & + & (1-\theta)\lambda(U_3^n + U_1^n) \\ \vdots & \vdots & \vdots \\ [1-2(1-\theta)\lambda]\,U_i^n & + & (1-\theta)\lambda(U_{i+1}^n + U_{i-1}^n) \\ \vdots & \vdots & \vdots \\ [1-2(1-\theta)\lambda]\,U_{N-2}^n & + & (1-\theta)\lambda(U_{N-1}^n + U_{N-3}^n) \\ [1-2(1-\theta)\lambda]\,U_{N-1}^n & + & (1-\theta)\lambda(U_N^n + U_{N-2}^n) \end{pmatrix} + \begin{pmatrix} A\theta\lambda \\ 0 \\ \vdots \\ 0 \\ B\theta\lambda \end{pmatrix}.$$

問 $\lim_{t\to\infty} u(x,t) = A + (B-A)x$ となることを示せ。

問 (1.5) は、適当な仮定のもとで

$$[(1+2\theta\lambda)I - \theta\lambda J]\,\boldsymbol{U}^{n+1} = \{[1-2(1-\theta)\lambda]\,I + (1-\theta)\lambda J\}\,\boldsymbol{U}^n + \theta\lambda \begin{pmatrix} A \\ 0 \\ \vdots \\ 0 \\ B \end{pmatrix}$$

と書くことが出来ることを示せ (この形が便利なこともある)。ただし I は $N-1$ 次の単位行列,

$$J = \begin{pmatrix} 0 & 1 & & & & \\ 1 & 0 & 1 & & & \\ & 1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 0 & 1 \\ & & & & 1 & 0 \end{pmatrix} \in M(N-1;\mathbf{R}).$$

1.4 非同次 Neumann 境界条件

1.4.1 問題の設定

$$u_t(x,t) = u_{xx}(x,t) \quad (t>0,\ 0<x<1) \tag{1.1}$$

$$u_x(0,t) = A, \quad u_x(1,t) = B \quad (t>0) \tag{1.2}$$

$$u(x,0) = f(x) \quad (0 \le x \le 1). \tag{1.3}$$

を考える。ここで A, B は (0 とは限らない) 定数である。

(1.2) を Neumann 境界条件という。特に $A = B = 0$ の場合、(1.2) を同次 Neumann 境界条件と呼び、厳密解や $t \to \infty$ のときの解の漸近挙動については、「微分方程式 2 (応用解析 II)」 (桂田 [2]) や「発展系の数値解析」で解説済みである。

熱方程式 (1.1) を近似する差分方程式として

$$(1+2\theta\lambda)U_i^{n+1} - \theta\lambda(U_{i+1}^{n+1} + U_{i-1}^{n+1}) = [1-2(1-\theta)\lambda]U_i^n + (1-\theta)\lambda(U_{i+1}^n + U_{i-1}^n) \tag{1.4}$$
$$(n = 0, 1, \cdots;\ i = 1, 2, \cdots, N-1)$$

を導き、初期条件 (1.3) から

$$U_i^0 = f(x_i) \quad (i = 0, 1, \cdots, N)$$

を導くのは、これまでと同じである。

境界条件 (1.2) をどう扱うかが問題である。以下、二つの方法を紹介する。一つは素朴で簡単な方法、もう一つは仮想格子点という概念を用いた少し凝った (それだけ高精度な近似解を得られる) 方法である。

1.4.2 素朴な方法

$u_x(0,t)$ を前進差分近似し、$u_x(1,t)$ を後退差分近似する、すなわち

$$u_x(0,t_{n+1}) = u_x(x_0,t_{n+1}) \fallingdotseq \frac{u_1^{n+1} - u_0^{n+1}}{h}, \quad u_x(1,t_{n+1}) = u_x(x_N,t_{n+1}) \fallingdotseq \frac{u_N^{n+1} - u_{N-1}^{n+1}}{h}$$

のように近似すると、境界条件の方程式 (1.2) に対応して

$$\frac{U_1^{n+1} - U_0^{n+1}}{h} = A, \quad \frac{U_N^{n+1} - U_{N-1}^{n+1}}{h} = B, \tag{1.5}$$

すなわち

$$U_0^{n+1} = U_1^{n+1} - Ah, \quad U_N^{n+1} = U_{N-1}^{n+1} + Bh \tag{1.6}$$

という差分方程式を採用することが考えられる。以下、この素朴な方法について説明する。

ある n について、既に $\{U_i^n\}_{i=0,1,\cdots,N}$ が分かっているとしたとき、(1.4), (1.6) は $N+1$ 個の未知数 $\{U_i^{n+1}\}_{i=0,1,\cdots,N}$ に対する $N+1$ 個の連立 1 次方程式となっている。

まず (1.6) を (1.4) に代入して U_0^{n+1}, U_N^{n+1} を消去しよう。

(1.4) で $i=1$ とした式

$$(1+2\theta\lambda)U_1^{n+1} - \theta\lambda(U_2^{n+1} + U_0^{n+1}) = [1-2(1-\theta)\lambda]U_1^n + (1-\theta)\lambda(U_2^n + U_0^n)$$

に $U_0^{n+1} = U_1^{n+1} - Ah$ を代入して、

$$(1+2\theta\lambda)U_1^{n+1} - \theta\lambda(U_2^{n+1} + U_1^{n+1} - Ah) = [1-2(1-\theta)\lambda]U_1^n + (1-\theta)\lambda(U_2^n + U_0^n),$$

すなわち

$$(1+\theta\lambda)U_1^{n+1} - \theta\lambda U_2^{n+1} = [1-2(1-\theta)\lambda]U_1^n + (1-\theta)\lambda(U_2^n + U_0^n) - A\theta\lambda h.$$

同様に (1.4) で $i=N-1$ とした式

$$(1+2\theta\lambda)U_{N-1}^{n+1} - \theta\lambda(U_N^{n+1} + U_{N-2}^{n+1}) = [1-2(1-\theta)\lambda]U_{N-1}^n + (1-\theta)\lambda(U_N^n + U_{N-2}^n)$$

に $U_N^{n+1} = U_{N-1}^{n+1} + Bh$ を代入して、

$$(1+2\theta\lambda)U_{N-1}^{n+1} - \theta\lambda(U_{N-1}^{n+1} + Bh + U_{N-2}^{n+1}) = [1-2(1-\theta)\lambda]U_{N-1}^n + (1-\theta)\lambda(U_N^n + U_{N-2}^n),$$

すなわち

$$(1+\theta\lambda)U_{N-1}^{n+1} - \theta\lambda U_{N-2}^{n+1} = [1-2(1-\theta)\lambda]U_{N-1}^n + (1-\theta)\lambda(U_N^n + U_{N-2}^n) + B\theta\lambda h.$$

こうして、$N-1$ 個の未知数 $\{U_i^{n+1}\}_{i=1,2,\cdots,N-1}$ に関する $N-1$ 個の連立 1 次方程式を得る。ベクトルと行列を用いて表わすと

$$\mathcal{A}\boldsymbol{U}^{n+1} = \boldsymbol{b}^n.$$

ただし

$$\mathcal{A} = \begin{pmatrix} 1+\theta\lambda & -\theta\lambda & & & & & \\ -\theta\lambda & 1+2\theta\lambda & -\theta\lambda & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & -\theta\lambda & 1+2\theta\lambda & -\theta\lambda & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & -\theta\lambda & 1+2\theta\lambda & -\theta\lambda \\ & & & & & -\theta\lambda & 1+\theta\lambda \end{pmatrix}, \tag{1.7}$$

$$\boldsymbol{U}^{n+1} = \begin{pmatrix} U_1^{n+1} \\ U_2^{n+1} \\ \vdots \\ U_{i-1}^{n+1} \\ U_i^{n+1} \\ U_{i+1}^{n+1} \\ \vdots \\ U_{N-2}^{n+1} \\ U_{N-1}^{n+1} \end{pmatrix}, \tag{1.8}$$

$$\boldsymbol{b}^n = \begin{pmatrix} [1-2(1-\theta)\lambda]\,U_1^n & + & (1-\theta)\lambda(U_2^n + U_0^n) \\ [1-2(1-\theta)\lambda]\,U_2^n & + & (1-\theta)\lambda(U_3^n + U_1^n) \\ \vdots & \vdots & \vdots \\ [1-2(1-\theta)\lambda]\,U_i^n & + & (1-\theta)\lambda(U_{i+1}^n + U_{i-1}^n) \\ \vdots & \vdots & \vdots \\ [1-2(1-\theta)\lambda]\,U_{N-2}^n & + & (1-\theta)\lambda(U_{N-1}^n + U_{N-3}^n) \\ [1-2(1-\theta)\lambda]\,U_{N-1}^n & + & (1-\theta)\lambda(U_N^n + U_{N-2}^n) \end{pmatrix} + \begin{pmatrix} -A\theta\lambda h \\ 0 \\ \vdots \\ 0 \\ B\theta\lambda h \end{pmatrix}. \tag{1.9}$$

($\mathcal{A}$ の対角成分がすべて同じではないことに注意しよう。同次 Dirichlet 境界条件の場合とは異なる行列である。)

この連立 1 次方程式が一意可解であることは、同次 Dirichlet 境界条件の場合と同様に証明できる。要するに係数行列が行について狭義優対角であることを示せばよいが、それは簡単である。

なお、念のために述べておくと、U_0^{n+1}, U_N^{n+1} については、上の連立 1 次方程式を解いて U_1^{n+1}, U_{N-1}^{n+1} を求めてから、

$$U_0^{n+1} = U_1^{n+1} - Ah, \quad U_N^{n+1} = U_{N-1}^{n+1} + Bh \tag{再掲 1.6}$$

で計算すればよい。

1.4.3 仮想格子点を用いる方法

前進差分近似

$$f'(x) \fallingdotseq \frac{f(x+h) - f(x)}{h} \qquad (\text{誤差は } O(h))$$

や後退差分近似

$$f'(x) \fallingdotseq \frac{f(x) - f(x-h)}{h} \qquad (\text{誤差は } O(h))$$

よりも 1 階中心差分近似

$$f'(x) \fallingdotseq \frac{f(x+h) - f(x-h)}{2h} \qquad (\text{誤差は } O(h^2))$$

の方が近似のオーダーが高いのであった。これを利用するために、

$$x_{-1} = -h, \quad x_{N+1} = (N+1)h = 1 + h$$

という x 座標を持つ「**仮想格子点**」$(x_{-1}, t_{n+1}), (x_{N+1}, t_{n+1})$ を導入して (もちろん、その点における u の値 $u_{-1}^{n+1}, u_{N+1}^{n+1}$ を考える — 本当は、仮想格子点は u の定義域の外なのだが — だから「仮想」と言うわけ)、$u_x(0,t), u_x(1,t)$ を共に 1 階中心差分近似する、すなわち

$$u_x(0,t_{n+1}) = u_x(x_0,t_{n+1}) \fallingdotseq \frac{u_1^{n+1} - u_{-1}^{n+1}}{2h}, \quad u_x(1,t_{n+1}) = u_x(x_N,t_{n+1}) \fallingdotseq \frac{u_{N+1}^{n+1} - u_{N-1}^{n+1}}{2h}$$

のように近似すると、境界条件 (1.2) から U_i^n に関する方程式として

$$\frac{U_1^{n+1} - U_{-1}^{n+1}}{2h} = A, \quad \frac{U_{N+1}^{n+1} - U_{N-1}^{n+1}}{2h} = B, \tag{1.10}$$

すなわち

$$U_{-1}^{n+1} = U_1^{n+1} - 2Ah, \quad U_{N+1}^{n+1} = U_{N-1}^{n+1} + 2Bh \tag{1.11}$$

を採用することが考えられる。この節ではこの近似法について考察する。

ある n について、既に $\{U_i^n\}_{i=0,1,\cdots,N}$ が分かっているとしたとき、(1.4), (1.11) は $N+3$ 個の未知数 $\{U_i^{n+1}\}_{i=-1,0,1,\cdots,N,N+1}$ に対する $N+1$ 個の連立 1 次方程式となっている。このままでは、方程式が 2 個不足するが、それを補うため、(1.4) が $i = 0, N$ についても成立していると仮定する。すると、未知数の個数と方程式の個数がともに $N+3$ になってつりあう。

まず (1.11) を (1.4) に代入して $U_{-1}^{n+1}, U_{N+1}^{n+1}$ を消去しよう。

(1.4) に $i = 0$ を代入した式

$$(1+2\theta\lambda)U_0^{n+1} - \theta\lambda(U_1^{n+1} + U_{-1}^{n+1}) = [1-2(1-\theta)\lambda]U_0^n + (1-\theta)\lambda(U_1^n + U_{-1}^n)$$

に $U_{-1}^{n+1} = U_1^{n+1} - 2Ah$ (と $U_{-1}^n = U_1^n - 2Ah$) を代入して、

$$(1+2\theta\lambda)U_0^{n+1} - \theta\lambda(U_1^{n+1} + U_1^{n+1} - 2Ah) = [1-2(1-\theta)\lambda]U_0^n + (1-\theta)\lambda(U_1^n + U_1^n - 2Ah),$$

すなわち

$$(1+2\theta\lambda)U_0^{n+1} - 2\theta\lambda U_1^{n+1} = [1-2(1-\theta)\lambda]U_0^n + 2(1-\theta)\lambda U_1^n - 2A\lambda h.$$

同様に (1.4) に $i = N$ を代入した式

$$(1+2\theta\lambda)U_N^{n+1} - \theta\lambda(U_{N+1}^{n+1} + U_{N-1}^{n+1}) = [1-2(1-\theta)\lambda]U_N^n + (1-\theta)\lambda(U_{N+1}^n + U_{N-1}^n)$$

に $U_{N+1}^{n+1} = U_{N-1}^{n+1} + 2Bh$ (と $U_{N+1}^n = U_{N-1}^n + 2Bh$) を代入して、

$$(1+2\theta\lambda)U_N^{n+1} - \theta\lambda(U_{N-1}^{n+1} + 2Bh + U_{N-1}^{n+1}) = [1-2(1-\theta)\lambda]U_N^n + (1-\theta)\lambda(U_{N-1}^n + 2Bh + U_{N-1}^n),$$

すなわち

$$(1+2\theta\lambda)U_N^{n+1} - 2\theta\lambda U_{N-1}^{n+1} = [1-2(1-\theta)\lambda]U_N^n + 2(1-\theta)\lambda U_{N-1}^n + 2B\lambda h.$$

こうして、$N+1$ 個の未知数 $\{U_i^{n+1}\}_{i=0,1,2,\cdots,N}$ に関する $N+1$ 個の連立 1 次方程式を得る。ベクトルと行列を用いて表わすと

$$\mathcal{A}\boldsymbol{U}^{n+1} = \boldsymbol{b}^n.$$

ただし

$$\mathcal{A} = \begin{pmatrix} 1+2\theta\lambda & -2\theta\lambda & & & & & & \\ -\theta\lambda & 1+2\theta\lambda & -\theta\lambda & & & & & \\ & \ddots & \ddots & \ddots & & & & \\ & & -\theta\lambda & 1+2\theta\lambda & -\theta\lambda & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & -\theta\lambda & 1+2\theta\lambda & -\theta\lambda & \\ & & & & & -2\theta\lambda & 1+2\theta\lambda & \end{pmatrix}, \tag{1.12}$$

$$\boldsymbol{U}^{n+1} = \begin{pmatrix} U_0^{n+1} \\ U_1^{n+1} \\ U_2^{n+1} \\ \vdots \\ U_{i-1}^{n+1} \\ U_i^{n+1} \\ U_{i+1}^{n+1} \\ \vdots \\ U_{N-2}^{n+1} \\ U_{N-1}^{n+1} \\ U_N^{n+1} \end{pmatrix}, \tag{1.13}$$

$$\boldsymbol{b}^n = \begin{pmatrix} [1-2(1-\theta)\lambda]\,U_0^n & + & 2(1-\theta)\lambda U_1^n \\ [1-2(1-\theta)\lambda]\,U_1^n & + & (1-\theta)\lambda(U_2^n + U_0^n) \\ [1-2(1-\theta)\lambda]\,U_2^n & + & (1-\theta)\lambda(U_3^n + U_1^n) \\ \vdots & \vdots & \vdots \\ [1-2(1-\theta)\lambda]\,U_i^n & + & (1-\theta)\lambda(U_{i+1}^n + U_{i-1}^n) \\ \vdots & \vdots & \vdots \\ [1-2(1-\theta)\lambda]\,U_{N-2}^n & + & (1-\theta)\lambda(U_{N-1}^n + U_{N-3}^n) \\ [1-2(1-\theta)\lambda]\,U_{N-1}^n & + & (1-\theta)\lambda(U_N^n + U_{N-2}^n) \\ [1-2(1-\theta)\lambda]\,U_N^n & + & 2(1-\theta)\lambda U_{N-1}^n \end{pmatrix} + \begin{pmatrix} -2A\lambda h \\ 0 \\ \vdots \\ 0 \\ 2B\lambda h \end{pmatrix}. \tag{1.14}$$

注意 1.4.1 (対称な係数行列が欲しい場合) 実際に数値計算する場合、連立 1 次方程式の係数行列は対称 (実対称 or Hermite) であることが望ましい。上に現われる $\mathcal{A}$ は対称ではないが、第 0, N 行を 1/2 倍すると対称にすることができる (連立 1 次方程式については、b の第 0, N 成分も 1/2 倍にすればよい)。 — このように何も考えないで差分近似すると非対称な差分方程式が得られるが、少し工夫すると対称にできることは良くある。 ■

1.4.4 二つの方法の比較

何回か卒研で取り組んでもらったのだが、最後までやってもらえなくて…$\lambda = \tau/h^2$ を安定性の条件を満たすように小さく固定して、N を大きくした場合、ある固定した (x,t) における精度を調

べると、$O(N^{-1}), O(N^{-2})$ となる。ところで、実際に実験してみると、二つの方法の差は非同次境界条件の場合に特に著しくなる (逆に言うと、同次境界条件の場合にはあまり差が出ない — これは結構はっきりとした差が出て、面白い結果だと思うのだが、まとめてもらえてなくて残念だなぁ)。

1.4.5 総熱量のチェック

あるとき、上で説明した境界条件の近似について疑いを持ったため、総熱量

$$J(t) := \int_0^1 u(x,t)\,dx$$

に相当するものを計算してチェックしてみよう、と考えた。

$$J'(t) = \int_0^1 \frac{\partial u}{\partial t}(x,t)\,dx = \int_0^1 \frac{\partial^2 u}{\partial x^2}(x,t)\,dx = [u_x(x,t)]_{x=0}^{x=1} = u_x(1,t) - u_x(0,t) = \beta - \alpha$$

であるから

$$J(t) = J(0) + (\beta - \alpha)t$$

となるはずである。

$J(t_n)$ に相当する量として、台形則による近似

$$E^n := h\left(\frac{1}{2}U_0^n + \sum_{i=1}^{N-1} U_i^n + \frac{1}{2}U_N^n\right)$$

を採用してみよう。

θ 法の差分方程式

$$\text{(1.15a)} \qquad (1+2\theta\lambda)U_0^{n+1} - 2\theta\lambda U_1^{n+1} = [1-2(1-\theta)\lambda]U_0^n + 2(1-\theta)\lambda U_1^n - 2\alpha\lambda h,$$

$$\text{(1.15b)} \qquad (1+2\theta\lambda)U_N^{n+1} - 2\theta\lambda U_{N-1}^{n+1} = [1-2(1-\theta)\lambda]U_N^n + 2(1-\theta)\lambda U_{N-1}^n + 2\beta\lambda h,$$

$$\text{(1.15c)} \qquad (1+2\theta\lambda)U_i^{n+1} - \theta\lambda\left(U_{i+1}^{n+1} + U_{i-1}^{n+1}\right) = [1-2(1-\theta)\lambda]U_i^n + (1-\theta)\lambda(U_{i+1}^n + U_{i-1}^n)$$
$$(1 \le i \le N-1)$$

について、第 1 式 $\times\frac{1}{2}$, 第 2 式 $\times\frac{1}{2}$, それから $1 \le i \le N-1$ を満たす i についての第 3 の式を加えると

$$\begin{aligned}
&(1-2\theta\lambda)\left(\frac{1}{2}U_0^{n+1} + \sum_{i=1}^{N-1} U_i^{n+1} + \frac{1}{2}U_N^{n+1}\right) - \theta\lambda\left(U_1^{n+1} + U_{N-1}^{n+1} + \sum_{i=2}^{N} U_i^{n+1} + \sum_{i=0}^{N-2} U_i^{n+1}\right) \\
&= [1-2(1-\theta)\lambda]\left(\frac{1}{2}U_0^n + \sum_{i=1}^{N-1} U_i^n + \frac{1}{2}U_N^n\right) + (1-\theta)\lambda\left(U_1^n + U_{N-1}^n + \sum_{i=2}^{N} U_i^n + \sum_{i=0}^{N-2} U_i^n\right) \\
&\quad + (\beta-\alpha)\lambda h
\end{aligned}$$

1.4.6 Neumann 境界条件の場合の解析解の導出

以下、Neumann 境界条件の場合の解析解の求め方について述べる。$A = B$ ならば、定常解が存在するので、それを熱方程式と境界条件を満す特解として用いて、同次 Neumann 境界値問題に帰着できる (これは簡単である)。

そこで、$A \neq B$ の場合を説明しよう。最初に考えるのは $A = B$ の場合と同じであり、特解を探してみる。ところが、

$$\begin{aligned} 0 &= v''(x), \\ v'(0) &= A, \quad v'(1) = B \end{aligned}$$

を満たす v は存在しない。

そこで、とりあえず (しかたなく) 境界条件

$$v'(0) = A, \quad v'(1) = B$$

のみを満たす v を適当に一つ取る。それから $w(x,t) = u(x,t) - v(x)$ とおくと、w は

$$\begin{aligned} &w_t(x,t) = w_{xx}(x,t) + v''(x) \quad (x \in (0,1),\, t \in (0,\infty)) \\ &w_x(0,t) = w_x(1,t) = 0 \quad (t \in (0,\infty)) \\ &w(x,0) = f(x) - v(x) \quad (x \in [0,1]) \end{aligned}$$

を満たす。

v が定常解ではないので、微分方程式が熱方程式ではなくなってしまったが (もっとも、v'' は簡単なものにできる)、境界条件は同次境界条件になった。

さて、ここからやり方は色々考えられる。

1. このような非同次熱方程式の初期値境界値問題に対する (Green 関数というものを用いる) 解の公式がある (「微分方程式 2 (旧 応用解析 II)」のノートで説明しておいた — それは本質的には、次のやり方と同等である)。

2. 割と一般的に使える方法として、固有関数展開法[5]というのがある。それは、

$$w(x,t) = \frac{a_0(t)}{2} + \sum_{n=1}^{\infty} a_n(t) \cos n\pi x$$

というように、固有関数 $\cos n\pi x$ で解 w が展開出来ると仮定して、これを方程式に代入して $a_n(t)$ についての方程式を導き、それを解く、というものである。

3. 今の問題の場合、v として簡単な 2 次関数が取れるので、v'' は定数になって簡単である。そこで、w についての問題の特解は困難なく発見できる。もっとも、定常解は存在しないだろうから、今度は見方を買えて、$V = V(t)$ と、t だけに依存する特解を探してみる。すると… (以下、各自やってみること)

[5]Galerkin 法という人もいる。古典的な Galerkin 法は、固有関数を基底関数に用いて、弱形式により未定係数を決定するというものであり、このうちの「固有関数を基底関数に」、「未定係数を決定」に注目して Galerkin 法と言っているのであろう。しかし有限要素法の話をしているとき、「有限要素法は近似関数に区分多項式を用いた Galerkin 法である」と言うことが多いので、個人的にはあまりしっくり来ない。

たまたまという感じは強いが、とりあえずこの問題を解くだけならば第 3 の方法が簡単である。結果だけ書いておくと、

$$u(x,t) = (B-A)t + \frac{B-A}{2}x^2 + Ax + \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n e^{-n^2\pi^2 t}\cos n\pi x,$$

$$a_n = 2\int_0^1 \left[f(x) - \left(\frac{B-A}{2}x^2 + Ax\right)\right]\cos n\pi x\,dx \quad (n = 0, 1, \cdots).$$

途中経過

v として

$$v(x) = \frac{B-A}{2}x^2 + Ax$$

を取ると、$v''(x) \equiv B - A$ となるので、

$$w_t = w_{xx} + B - A.$$

そこで $V(t) := (B-A)t$ と選び、$W(x,t) := w(x,t) - V(t)$ とおくと、

$$\begin{aligned} W_t(x,t) &= W_{xx}(x,t) \quad ((x,t) \in (0,1)\times(0,\infty)), \\ W_x(0,t) &= W_x(1,t) = 0 \quad (t \in (0,\infty)), \\ W(x,0) &= f(x) - v(x) \quad (x \in [0,1]). \end{aligned}$$

これは同次 Neumann 問題であるから、W は次のように求まる。

$$W(x,t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n e^{-n^2\pi^2 t}\cos n\pi x, \quad a_n = 2\int_0^1 (f(x) - v(x))\cos n\pi x\,dx.$$

ゆえに

$$\begin{aligned} u(x,t) &= w(x,t) + v(x) = W(x,t) + V(t) + v(x) \\ &= \frac{B-A}{2}x^2 + Ax + (B-A)t + \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n e^{-n^2\pi^2 t}\cos n\pi x. \blacksquare \end{aligned}$$

なお、固有関数展開法ついては、(あちこちに載っているが) 例えばスタンリー・ファーロウ [3] に入門的な記述がある。最近は「見直された」のか、説明が載っている本が増えたので、自力で少し探索してみることを勧める。

問　$A = B$ のとき、$\lim_{t\to\infty} u(x,t)$ を求めよ。

1.5　色々な境界条件を扱えるプログラムの作成

(久しぶりにこの章に書き足す。まだ書き足りない。厳密解の導出とか。)

1.5.1 Robin の条件

例えば、境界において外界への熱の「放散」が起っているとき、数学的にはいわゆる**第三種境界条件 (Robin の境界条件, Newton の境界条件)**

$$\frac{\partial u}{\partial n} + hu = T \tag{1.16}$$

となる。ここで h は既知の物理的定数 (正の定数)、T は物理的定数と周囲の温度で定まる既知量 (関数となりうる) である。

Newton の冷却の法則

物体の表面上の点 x_0 から、温度 U の外界へ熱の放散が起っているとき、その放散量は温度差 $u(x_0,t)-U$ に比例する。特に

$$\frac{\partial u}{\partial n}(x_0,t) = \sigma(u(x_0,t)-U) \quad (\sigma \text{ は正の定数})$$

が成り立つ。

ここでは γ, T_0, T_1 を既知定数として、

$$\begin{aligned} -\frac{\partial u}{\partial x}(0,t) + \gamma u(0,t) &= T_0, \\ \frac{\partial u}{\partial x}(1,t) + \gamma u(1,t) &= T_1 \end{aligned}$$

という境界条件を考えることにする。仮想格子点を導入して、中心差分近似することで、それぞれ

$$\begin{aligned} \frac{U^m_{-1} - U^m_1}{2h} + \gamma U^m_0 &= T_0, \\ \frac{U^m_{N+1} - U^m_{N-1}}{2h} + \gamma U^m_N &= T_1 \end{aligned}$$

という差分方程式を得る。これから

$$U^m_{-1} = U^m_1 + 2h(T_0 - \gamma U^m_0), \tag{1.17}$$

$$U^m_{N+1} = U^m_{N-1} + 2h(T_1 - \gamma U^m_N). \tag{1.18}$$

さて、熱方程式を差分近似して得られる差分方程式 (1.4) が $i=0$ でも成り立つとして

$$(1+2\theta\lambda)U^{n+1}_0 - \theta\lambda(U^{n+1}_1 + U^{n+1}_{-1}) = [1-2(1-\theta)\lambda]\,U^n_0 + (1-\theta)\lambda(U^n_1 + U^n_{-1}).$$

ここで $m = n, n+1$ の場合の (1.17) を代入すると、

$$[1+2\theta\lambda]\,U^{n+1}_0 - 2\theta\lambda U^{n+1}_1 - 2\theta\lambda h(T_0 - \gamma U^{n+1}_0) = [1-2(1-\theta)\lambda]\,U^n_0 + 2(1-\theta)\lambda U^n_1 + 2(1-\theta)\lambda h(T_0 - \gamma U^n_0).$$

整理すると

$$[1+2\theta\lambda(1+\gamma h)]\,U^{n+1}_0 - 2\theta\lambda U^{n+1}_1 = [1-2(1-\theta)\lambda(1+\gamma h)]\,U^n_0 + 2(1-\theta)\lambda U^n_1 + 2\lambda h T_0.$$

同様にして熱方程式を差分近似して得られる差分方程式が $i=N$ で成り立つとした式に、$m = n, n+1$ の場合の (1.18) を代入して整理すると、

$$[1+2\theta\lambda(1+\gamma h)]\,U^{n+1}_N - 2\theta\lambda U^{n+1}_{N-1} = [1-2(1-\theta)\lambda(1+\gamma h)]\,U^n_N + 2(1-\theta)\lambda U^n_{N-1} + 2\lambda h T_1.$$

これから

$$A\boldsymbol{U}^{n+1} = \boldsymbol{b}^n$$

となる。ただし、

$$\begin{aligned}
A &= \begin{pmatrix}
1+2\theta\lambda(1+\gamma h) & -2\theta\lambda & & & & \\
-\theta\lambda & 1+2\theta\lambda & -\theta\lambda & & & \\
& \ddots & \ddots & \ddots & & \\
& & -\theta\lambda & 1+2\theta\lambda & -\theta\lambda & \\
& & & -2\theta\lambda & 1+2\theta\lambda(1+\gamma h)
\end{pmatrix}, \\
\boldsymbol{U}^{n+1} &= \left(U_0^{n+1}, \cdots, U_N^{n+1}\right)^T, \\
\boldsymbol{b}^n &= \begin{pmatrix}
\left[1-2(1-\theta)\lambda(1+\gamma h)\right]U_0^n + 2(1-\theta)\lambda U_1^n \\
\left[1-2(1-\theta)\lambda\right]U_1^n + (1-\theta)\lambda(U_2^n + U_0^n) \\
\vdots \\
\left[1-2(1-\theta)\lambda\right]U_i^n + (1-\theta)\lambda(U_{i+1}^n + U_{i-1}^n) \\
\vdots \\
\left[1-2(1-\theta)\lambda\right]U_{N-1}^n + (1-\theta)\lambda(U_N^n + U_{N-2}^n) \\
\left[1-2(1-\theta)\lambda(1+\gamma h)\right]U_N^n + 2(1-\theta)\lambda U_{N-1}^n
\end{pmatrix} + \begin{pmatrix}
2\lambda h T_0 \\ 0 \\ \vdots \\ 0 \\ 2\lambda h T_1
\end{pmatrix}.
\end{aligned}$$

なお、$\theta = 1$ と選んだ場合、A と $\boldsymbol{b}^n$ は特に簡単になる。参考まで、ここに掲げておく。

$$\begin{aligned}
A &= \begin{pmatrix}
1+2\lambda(1+\gamma h) & -2\lambda & & & & \\
-\lambda & 1+2\lambda & -\lambda & & & \\
& \ddots & \ddots & \ddots & & \\
& & -\lambda & 1+2\lambda & -\lambda & \\
& & & -2\lambda & 1+2\lambda(1+\gamma h)
\end{pmatrix}, \\
\boldsymbol{b}^n &= \begin{pmatrix}
U_0^n \\ \vdots \\ U_i^n \\ \vdots \\ U_N^n
\end{pmatrix} + \begin{pmatrix}
2\lambda h T_0 \\ 0 \\ \vdots \\ 0 \\ 2\lambda h T_1
\end{pmatrix}.
\end{aligned}$$

1.5.2 「万能」プログラム

Robin の境界条件という場合 $\gamma > 0$ なのだが、形式的に $\gamma = 0$ とおくといわゆる Neumann 境界条件になるので、無理なく一つのプログラムを使うことができる。

そこで Robin の条件に代えて、Dirichlet 境界条件

$$u(0,t) = \alpha, \quad u(1,t) = \beta$$

も考えてみよう。この場合は $A, \boldsymbol{b}^n$ を次の $\widetilde{A}, \widetilde{\boldsymbol{b}}^n$ に変えればよい[6]。

$$\widetilde{A} = \begin{pmatrix} 1 & 0 & & & \\ -\theta\lambda & 1+2\theta\lambda & -\theta\lambda & & \\ & \ddots & \ddots & \ddots & \\ & & -\theta\lambda & 1+2\theta\lambda & -\theta\lambda \\ & & & 0 & 1 \end{pmatrix},$$

$$\widetilde{\boldsymbol{b}}^n = \begin{pmatrix} \alpha \\ [1-2(1-\theta)\lambda]\,U_1^n + (1-\theta)\lambda(U_2^n + U_0^n) \\ \vdots \\ [1-2(1-\theta)\lambda]\,U_i^n + (1-\theta)\lambda(U_{i+1}^n + U_{i-1}^n) \\ \vdots \\ [1-2(1-\theta)\lambda]\,U_{N-1}^n + (1-\theta)\lambda(U_N^n + U_{N-2}^n) \\ \beta \end{pmatrix}.$$

A と $\widetilde{A}, \boldsymbol{b}^n$ と $\widetilde{\boldsymbol{b}}^n$ はほとんど同じなので、無理なく一つのプログラムで処理することができる。

```
/*
 * heat1g-v2.c 1 次元熱方程式, 境界条件は Robin (含む Neumann) と Dirichlet
 *
 * Robin 境界条件に取り組む機会を与えてくれた A.Y. に感謝
 * 2005 年 3 月 10 日 作成
 * 2006 年 1 月 2 日　注釈を書き足す
 * 2024 年 8 月 21 日 URL など注釈を書き直す
 *
 *   u_t(x,t)=u_{xx}(x,t) ((x,t) ∈ (0,1) × (0, ∞))
 *   -u_x(0,t)+γ u(0,t)=T0  (t ∈ (0, ∞)) または u(0,t)=α  (t ∈ (0, ∞))
 *    u_x(1,t)+γ u(1,t)=T1  (t ∈ (0, ∞)) または u(1,t)=β  (t ∈ (0, ∞))
 *   u(x,0)=u0(x)          (x ∈ [0,1])
 *
 *  差分方程式の説明は次の文書に書き足した (2024 年 8 月 21 日時点で第 1 章 5 節)。
 *  『熱方程式に対する差分法 I --- 区間における熱方程式 ---』
 *   https://m-katsurada.sakura.ne.jp/labo/text/heat-fdm-1.pdf
 *
 *  このプログラム自体は
 *   curl -O https://m-katsurada.sakura.ne.jp/program/fdm/heat1g-v2.c
 *  とすれば入手できる
 */

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

double exact(double x, double t);
double u0(double x);
void trilu(int n, double *al, double *ad, double *au);
void trisol(int n, double *al, double *ad, double *au, double *b);

/* left, right は != 0 だと Robin 境界条件
 *
 *  ∂ u
```

[6]少々強引ではある。この場合の連立 1 次方程式は、係数行列を簡単に対称化することは出来ないようであるし。遊戯かもしれない。

```
 *  ---- + γ u=T
 *  ∂ n
 *
 * を課す（γ=0 だと Neumann 境界条件)。== 0 だと Dirichlet 境界条件
 *
 *  u=α (x=0), u=β (x=1)
 *
 * を課す。
 */

int left = 0, right = 0;
double pi;

int main(void)
{
  int i,Nx,n,nMax;
  double *u, *uu;
  double *ad,*al,*au;
  double x,h,lambda,theta,tau,gamma,t,error,maxerror;
  double T0,T1,alpha,beta;
  double Tmax = 0.1;

  pi = 4 * atan(1.0);
  gamma = 1;

  if (left)
    T0 = 0;
  else
    alpha = 0;
  if (right)
    T1 = 0;
  else
    beta = 0;

  printf("Nx="); scanf("%d", &Nx);
  h = 1.0 / Nx;
  lambda = 0.5;
  theta = 0.5;
  tau = lambda * h * h;

  u = malloc(sizeof(double) * (Nx+1));
  uu = malloc(sizeof(double) * (Nx+1));
  ad = malloc(sizeof(double) * (Nx+1));
  au = malloc(sizeof(double) * (Nx+1));
  al = malloc(sizeof(double) * (Nx+1));

  /* 係数行列を作る */
  for (i = 1; i < Nx; i++) {
    ad[i] = 1 + 2 * theta * lambda;
    au[i] = al[i] = - theta * lambda;
  }

  if (left) {
    ad[0] = 1 + 2 * theta * lambda * (1 + gamma * h);
    au[0] = - 2 * theta * lambda;
  }
  else {
    ad[0] = 1; au[0] = 0;
  }
  if (right) {
```

```
    ad[Nx] = 1 + 2 * theta * lambda * (1 + gamma * h);
    al[Nx] = - 2 * theta * lambda;
  }
  else {
    ad[Nx] = 1; al[Nx] = 0;
  }
  /* LU 分解 */
  trilu(Nx + 1, al, ad, au);

  /* 初期条件 */
  for (i = 0; i <= Nx; i++)
    u[i] = u0(i * h);

  /* 繰り返し */
  nMax = Tmax / tau + 0.5;
  for (n = 1; n <= nMax; n++) {
    for (i = 1; i < Nx; i++)
      uu[i] = (1-2*(1-theta)*lambda)*u[i]+(1-theta)*lambda*(u[i+1]+u[i-1]);
    if (left)
      uu[0] = (1 - 2 * (1 - theta) * lambda * (1 + gamma * h)) * u[0]
               + 2 * (1 - theta) * lambda * u[1]
               + 2 * h * lambda * T0;
    else
      uu[0] = U0;
    if (right)
      uu[Nx]  = (1 - 2 * (1 - theta) * lambda * (1 + gamma * h)) * u[Nx]
                 + 2 * (1 - theta) * lambda * u[Nx - 1]
                 + 2 * h * lambda * T1;
    else
      uu[Nx] = U1;

    trisol(Nx + 1, al, ad, au, uu);

    for (i = 0; i <= Nx; i++)
      u[i] = uu[i];
    t = n * tau;
#ifdef VERBOSE
    printf("%f\n", t);
    for (i = 0; i <= Nx; i++)
      printf("%5.2f ", u[i]);
    printf("\n");
    for (i = 0; i <= Nx; i++)
      printf("%5.2f ", exact(i * h, t));
    printf("\n");
#endif
    maxerror = 0;
    for (i = 0; i <= Nx; i++) {
      error = fabs(exact(i * h, t) - u[i]);
      if (error > maxerror)
        maxerror = error;
    }
    printf("error=%e\n", maxerror);
  }
}

#define lambda1 2.028757838110434
#define lambda2 4.913180439434883

double exact(double x, double t)
{
```

```
  if (left) {
    /* まだ厳密解を知りません */
  }
  else {
    if (right)
      return sin(lambda1 * x) * exp(- lambda1 * lambda1 * t)
        + sin(lambda2 * x) * exp(- lambda2 * lambda2 * t);
    else
      return sin(pi * x) * exp(- pi * pi * t)
        + sin(2 * pi * x) * exp(- 4 * pi * pi * t);
  }
}

double u0(double x)
{
  return exact(x, 0.0);
}

/* 3 項方程式（係数行列が三重対角である連立 1 次方程式のこと）Ax=b を解く
 *
 *   入力
 *     n: 未知数の個数
 *     al,ad,au: 連立 1 次方程式の係数行列
 *       (al: 対角線の下側 i.e. 下三角部分 (lower part)
 *        ad: 対角線       i.e. 対角部分 (diagonal part)
 *        au: 対角線の上側 i.e. 上三角部分 (upper part)
 *        つまり
 *
 *         ad[0] au[0]   0   ................. 0
 *         al[1] ad[1] au[1]   0  ............. 0
 *           0   al[2] ad[2] au[2]  0 ......... 0
 *                         ...................
 *                         al[n-2] ad[n-2] au[n-2]
 *                           0     al[n-1] ad[n-1]

 *        al[i] = A_{i,i-1}, ad[i] = A_{i,i}, au[i] = A_{i,i+1},
 *        al[0], au[n-1] は意味がない)
 *
 *     b: 連立 1 次方程式の右辺の既知ベクトル
 *        (添字は 0 から。i.e. b[0],b[1],...,b[n-1] にデータが入っている。)
 *   出力
 *     al,ad,au: 入力した係数行列を LU 分解したもの
 *     b: 連立 1 次方程式の解
 *   能書き
 *     一度 call すると係数行列を LU 分解したものが返されるので、
 *     以後は同じ係数行列に関する連立 1 次方程式を解くために、
 *     関数 trisol() が使える。
 *   注意
 *     Gauss の消去法を用いているが、ピボットの選択等はしていな
 *     いので、ピボットの選択をしていないので、係数行列が正定値である
 *     などの適切な条件がない場合は結果が保証できない。
 */

/* 三重対角行列の LU 分解 (pivoting なし) */
void trilu(int n, double *al, double *ad, double *au)
{
  int i, nm1 = n - 1;
  /* 前進消去 (forward elimination) */
  for (i = 0; i < nm1; i++) {
    al[i + 1] /= ad[i];
```

```
    ad[i + 1] -= au[i] * al[i + 1];
  }
}

/* LU 分解済みの三重対角行列を係数に持つ 3 項方程式を解く */
void trisol(int n, double *al, double *ad, double *au, double *b)
{
  int i, nm1 = n - 1;
  /* 前進消去 (forward elimination) */
  for (i = 0; i < nm1; i++) b[i + 1] -= b[i] * al[i + 1];
  /* 後退代入 (backward substitution) */
  b[nm1] /= ad[nm1];
  for (i = n - 2; i >= 0; i--) b[i] = (b[i] - au[i] * b[i + 1]) / ad[i];
}
```

1.5.3 Robin 境界条件の場合の厳密解

(工事中 — というよりも忘れないように書きかけておく)

まず、同次境界条件

$$-u_x(0,t)+\gamma_1 u(0,t)=0,\quad u_x(1,t)+\gamma_2 u(1,t)=0\quad (t\in(0,\infty))$$

のもとでの厳密解を Fourier の方法で求める。$\gamma_1,\gamma_2>0$ である。

対応する固有値問題は

$$X''(x)=-\lambda X(x),\quad -X'(0)+\gamma_1 X(0)=0,\quad X'(1)+\gamma_2 X(1)=0$$

である。ここで λ は定数であるが、よくある議論で $\lambda>0$ であることが分かる (「微分方程式 2」2 の講義ノート [2] を見よ)。この微分方程式の一般解は

$$X(x)=A\cos\sqrt{\lambda}x+B\sin\sqrt{\lambda}x\quad (A,\,B\text{ は任意定数})$$

である。境界条件に代入して

$$-\lambda B+\gamma_1 A=0,\quad \lambda(-A\sin\lambda+B\cos\lambda)+\gamma_2(A\cos\lambda+B\sin\lambda)=0.$$

すなわち

$$\begin{pmatrix}\gamma_1 & -\sqrt{\lambda}\\ -\sqrt{\lambda}\sin\sqrt{\lambda}+\gamma_2\cos\sqrt{\lambda} & \sqrt{\lambda}\cos\sqrt{\lambda}+\gamma_2\sin\sqrt{\lambda}\end{pmatrix}\begin{pmatrix}A\\B\end{pmatrix}=\begin{pmatrix}0\\0\end{pmatrix}.$$

非自明解を持つために

$$\begin{aligned}0&=\begin{vmatrix}\gamma_1 & -\sqrt{\lambda}\\ -\sqrt{\lambda}\sin\sqrt{\lambda}+\gamma_2\cos\sqrt{\lambda} & \sqrt{\lambda}\cos\sqrt{\lambda}+\gamma_2\sin\sqrt{\lambda}\end{vmatrix}\\ &=(\gamma_1\gamma_2-\lambda)\sin\sqrt{\lambda}+(\gamma_1+\gamma_2)\sqrt{\lambda}\cos\sqrt{\lambda}.\end{aligned}\tag{1.19}$$

これは $\sqrt{\gamma_1\gamma_2}\notin\dfrac{\pi}{2}+\pi\mathbf{Z}$ のとき、

$$\tan\sqrt{\lambda}=\frac{(\gamma_1+\gamma_2)\sqrt{\lambda}}{\lambda-\gamma_1\gamma_2}\tag{1.20}$$

に同値である。一方 $\sqrt{\gamma_1\gamma_2} \in \dfrac{\pi}{2} + \pi\mathbf{Z}$ のときは、

$$\lambda = \gamma_1\gamma_2 \quad \text{または} \quad \tan\sqrt{\lambda} = \frac{(\gamma_1+\gamma_2)\sqrt{\lambda}}{\lambda - \gamma_1\gamma_2}.$$

(1.20) の場合、正の解全体は

$$\lambda_1 < \lambda_2 < \cdots \to \infty$$

と並べることが出来、

$$\lim_{n\to\infty} \left| \lambda_n - \left(n - \frac{1}{2}\right)^2 \pi^2 \right| = 0$$

であることが分かる。$\lambda = \lambda_n$ のとき、(1.20) の解は、

$$\begin{pmatrix} A \\ B \end{pmatrix} = t \begin{pmatrix} \sqrt{\lambda} \\ \gamma_1 \end{pmatrix} \quad (t \neq 0).$$

対応する固有関数は

$$X(x) = \sqrt{\lambda}\cos\sqrt{\lambda}x + \gamma_1 \sin\sqrt{\lambda}x.$$

固有値の近似値を求め、固有関数のグラフの概形を描くのは簡単である。

Mathematica で遊ぶ

```
gamma1=1
gamma2=1
solnear[x0_]:=FindRoot[Tan[x]-(gamma1+gamma2)*x/(x-gamma1*gamma2)==0,
 {x,x0},WorkingPrecision->20,AccuracyGoal->40]
rootlambda[1]= x /. solnear[1.5]
rootlambda[n_]:=rootlambda[n]=x /.  solnear[rootlambda[n-1]+Pi]
Table[rootlambda[n],{n,10}]
eigenfunc[n_]:=Plot[rootlambda[n]*Cos[rootlambda[n]*x]
                    +gamma1*Sin[rootlambda[n]*x],{x,0,1}]
```

γ_1, γ_2 が小さい時に Neumann 境界条件に近いか、γ_1, γ_2 が大きい時に Dirichlet 境界条件に近いか？ などなど、研究に値すると思う。

1.6 周期境界条件

うっかりしていたのだが、**周期境界条件**と呼ばれる条件

$$u(0,t) = u(1,t), \quad u_x(0,t) = u_x(1,t) \quad (t \in (0,\infty)) \tag{1.21}$$

も良く出て来る。

1.6.1 なぜ周期境界条件と呼ぶか

(準備中)

1.6.2　差分方程式の作り方

素朴な方法

(工事中)

$$U_0^{n+1} = U_N^{n+1}, \quad \frac{U_1^{n+1} - U_0^{n+1}}{h} = \frac{U_N^{n+1} - U_{N-1}^{n+1}}{h}.$$

周期数列に対する差分方程式を用いる方法

(工事中)

n を固定したとき、$\{U_i^{n+1}\}$ が i について、周期 N の周期数列である:

$$U_{i+N}^{n+1} = U_i^{n+1} \quad (i \in \mathbf{Z},\, n \geq 0)$$

ことを用いる。以下の考え方は円盤・円環領域での Laplacian の近似に通じるものがある。

独立な成分として、$U_0^{n+1}, U_1^{n+1}, \cdots, U_{N-1}^{n+1}$ という N 個が取れる。N 個の方程式を課すのが自然で、$i = 0, 1, \cdots, N-1$ について

$$(1 + 2\theta\lambda)U_i^{n+1} - \theta\lambda(U_{i+1}^{n+1} + U_{i-1}^{n+1}) = (1 - 2(1-\theta)\lambda)U_i^n - (1-\theta)\lambda(U_{i+1}^n + U_{i-1}^n)$$

を採用する。ただし $1 \leq i \leq N-2$ については、直接この方程式で大丈夫だが、$i = 0$ については、$U_{i-1}^{n+1} = U_{-1}^{n+1}$ を U_{N-1}^{n+1} で置き換えた

$$(1 + 2\theta\lambda)U_0^{n+1} - \theta\lambda(U_1^{n+1} + U_{N-1}^{n+1}) = (1 - 2(1-\theta)\lambda)U_0^n - (1-\theta)\lambda(U_1^n + U_{N-1}^n)$$

を、$i = N-1$ については、$U_{i+1}^{n+1} = U_N^{n+1}$ を U_0^{n+1} で置き換えた

$$(1 + 2\theta\lambda)U_{N-1}^{n+1} - \theta\lambda(U_0^{n+1} + U_{N-2}^{n+1}) = (1 - 2(1-\theta)\lambda)U_{N-1}^n - (1-\theta)\lambda(U_0^n + U_{N-2}^n)$$

を用いる。

こうして

$$\mathcal{A}U^{(n+1)} = b^{(n)}$$

を得る。ただし

$$A = \begin{pmatrix} 1+2\theta\lambda & -\theta\lambda & & & -\theta\lambda \\ -\theta\lambda & 1+2\theta\lambda & -\theta\lambda & & \\ & \ddots & \ddots & \ddots & \\ & & -\theta\lambda & 1+2\theta\lambda & -\theta\lambda \\ -\theta\lambda & & & -\theta\lambda & 1+2\theta\lambda \end{pmatrix}$$

この係数行列は三重対角行列ではないが、比較的簡単に LU 分解できる。

時間が欲しいなあ…せっかくつかみかけたものがこぼれ落ちていくような…

第2章　2次元領域における熱伝導方程式

2.1　Fourier の方法の限界

Ω を $\mathbf{R}^d$ における領域[1]とするとき、Ω における熱伝導方程式の初期値境界値問題 (境界条件は同次 Dirichlet 境界条件)

$$u_t(x,t) = \triangle u(x,t) \quad ((x,t) \in \Omega \times (0,\infty)), \tag{2.1}$$

$$u(x,t) = 0 \quad ((x,t) \in \Gamma \times (0,\infty)), \tag{2.2}$$

$$u(x,0) = f(x) \quad (x \in \overline{\Omega}) \tag{2.3}$$

を考えよう。ここで Γ は Ω の境界[2] $\partial\Omega$ で、十分滑らかであるとする。$\overline{\Omega}$ は Ω の閉包 $[a,b] \times [c,d]$ である。また、f は $\overline{\Omega}$ 上定義された与えられた関数である。

すでに $d = 1$, $\Omega = (0,1)$ の場合には、「微分方程式 2」で学んでいる。一口にまとめると

1 次元熱伝導方程式は Fourier の方法で解ける

となる。

ところで、(残念ながら) 多次元領域 (いいかえると $d \geq 2$) の場合には、Fourier の方法は万能とは言えない。なぜそうなのかを、以下に見てみよう。

まず、固有値問題が現われるところまでは、1 次元の場合と同じである。熱方程式と境界条件を満す $u(x,t) = \zeta(x)\eta(t)$ の形の関数 (で恒等的には 0 でないもの) を満すことを目標にすえる。境界条件に代入することによって、

$$\zeta(x) = 0 \quad (x \in \Gamma)$$

が得られる。一方、熱方程式に代入すると

$$\zeta(x)\eta'(t) = \triangle\zeta(x)\eta(t)$$

となるから

$$\frac{\eta'(t)}{\eta(t)} = \frac{\triangle\zeta(x)}{\zeta(x)}$$

が得られる。この等式の値は定数となるので、それを $-\lambda$ とおくと[3]、

$$\eta'(t) = -\lambda\eta(t),$$

$$\triangle\zeta(x) = -\lambda\zeta(x).$$

[1]領域とは、連結開集合のことである。$\mathbf{R}^1$ においては、領域=開区間であるが、$\mathbf{R}^d$ $(d \geq 2)$ においては、開区間でない領域はたくさんある。

[2]例えば、$d = 1$, $\Omega = (0,1)$ であれば $\partial\Omega = \{0,1\}$. ∂ は本来は境界多様体を表わす記号であり、境界多様体と位相空間の部分集合の境界とは異なる概念であるが、滑らかな境界を持つ開集合の場合には、両者は一致する。

[3]負号 $-$ をつけるのは、世の中の習慣に合わせるため。$\triangle$ の固有値は 負 (Neumann 境界条件の場合は、非正) になるので、符号を変えて $-\triangle$ を考える (固有値は正となる) 習慣が広まっている。正値でないと分数巾などを考えることもできないし。

こうして、$\zeta = \zeta(x)$ については、$-\triangle$ に関する固有値問題

$$-\triangle\zeta = \lambda\zeta \quad (\text{in } \Omega), \tag{2.4}$$

$$\zeta = 0 \quad (\text{on } \Gamma), \tag{2.5}$$

$$\zeta \not\equiv 0 \tag{2.6}$$

が導かれた。

空間の次元 d が 1 に等しい場合は、この固有値問題を定数係数線形常微分方程式に対する特性根の方法で解くことができて、スイスイと話が進んだのだが、$d \geq 2$ の場合は、この方程式は偏微分方程式であって、簡単ではない。

結論のみ述べると、この固有値問題には、$d = 1$ の場合と同様の解が存在する (藤田他 [4] 参照)。その解 ζ_n, λ_n $(n \in \mathbf{N})$ を用いれば、Fourier の方法の議論は $n = 1$ の場合と (一応は) 同様に進めることができるが、実は固有値問題の解の存在は抽象的に保証されるだけで、一般には解は具体的に求められない、という限界がある。

> 多次元の固有値問題は特別な場合をのぞいて具体的には解けない！
> $\longrightarrow$ 多次元の **Fourier** の方法には限界がある

もっとも、この事情 (解の存在が保証されるだけで、解の具体的な表示式は得られない) は、Fourier の方法を適用する場合に限ったことではなくて、他のどんな方法[4]を用いた場合も同様である。

余談 2.1.1 一般の領域で 固有値、固有関数がどうなっているか知りたい場合、数値計算する手もある。例えば菊地&山本 [1] には、正方形領域における Laplacian の固有値・固有関数を差分法で計算するプログラムが載っている (1997 年度卒研の丹羽功・山田英二 [8] も参考になる)。また有限要素法で計算するプログラムを書くのも難しくはない (1997 年度卒研の高藤康孝 [9], 1998 年度卒研の鈴木康大 [10])。 ■

余談 2.1.2 どうも 藤田他 [4] は読みにくいので、簡単に要点を書いておく[5]。多次元領域 Ω における $-\triangle$ (境界条件は Dirichlet) の固有値は、正の無限大に発散する実数の無限列

$$0 < \lambda_1 < \lambda_2 \leq \lambda_3 \leq \lambda_4 \leq \cdots \to \infty$$

をなす。対応する固有関数を φ_k とおくと、$\forall w \in L^2(\Omega)$ は、

$$w = \sum_{k=1}^{\infty} c_k w_k \quad (\text{in } L^2(\Omega))$$

と展開できる。$\{\varphi_k\}_{k\in\mathbf{N}}$ は正規直交系であるように選べるから、その場合は

$$c_k = \int_\Omega w(x)\overline{\varphi_k(x)}dx$$

で係数 $\{c_k\}$ が定められる。ゆえに熱伝導方程式の解は

$$u(x,t) = \sum_{k=1}^{\infty} c_k e^{-\lambda_k t}\varphi_k(x), \quad c_k = \int_\Omega f(x)\overline{\varphi_k(x)}dx$$

[4]熱伝導方程式の解法としては、(1) 積分方程式を解いて Green 関数を (やや間接的に) 構成する、(2) 作用素の半群理論を用いる、などの方法がある。前者については、例えば伊藤 [5], [6] を見よ (ただし、この本では Green 関数のことを基本解と呼んでいる)。後者については、例えば増田 [7] を見ると良い。

[5]これは俣野 [11] からの引用だが、この本に証明が書いてあるわけではない。

であり、漸近評価

$$(\forall \delta > 0) \quad (\exists C > 0) \quad (\forall t \geq \delta) \quad \text{s.t.} \quad |u(x,t)| \leq Ce^{-\lambda_1 t}$$

が成り立つ。熱伝導方程式の Green 関数は

$$U(x,y,t) = \sum_{k=1}^{\infty} e^{-\lambda_k t} \varphi_k(x) \varphi_k(y)$$

であり、さらに $\triangle$ の Green 関数は

$$G(x,y) = -\int_0^{\infty} U(x,y,t)\, dt = -\sum_{k=1}^{\infty} \frac{1}{\lambda_k} \varphi_k(x) \varphi_k(y)$$

と表わすことができる。 ■

注意 2.1.3 この節では、解を求める方法、解の存在を示す方法について述べたが、古典解の一意性等については、空間 1 次元の場合とほとんど同様である。すなわち、最大値原理が成立し、それから古典解の一意性、正値性、順序保存性などが導かれる。例えば Protter and Weinberger [12] を参照せよ。 ■

2.2 長方形領域における初期値境界値問題

前節では Fourier の方法の限界について述べたが、いくつかの簡単だが重要な場合に、Fourier の方法はそのパワーを発揮する。ここでは長方形領域における初期値境界値問題の解を紹介しよう。

(この文書の古い版では、円盤領域における初期値境界値問題の解について言及してあったが、それについては、桂田 [13] を見よ。)

Ω が 2 次元空間内の長方形領域 $(0,L) \times (0,H)$ の場合には、さらなる変数分離によって、固有値問題

$$\begin{aligned}
&\frac{\partial^2 \zeta}{\partial x^2} + \frac{\partial^2 \zeta}{\partial y^2} = -\lambda \zeta \quad (0 < x < L,\, 0 < y < H), \\
&\zeta(0,y) = \zeta(L,y) = 0, \quad \zeta(x,0) = \zeta(x,H) = 0 \quad (0 \leq x \leq L,\, 0 \leq y \leq H), \\
&\zeta \not\equiv 0
\end{aligned}$$

が解ける。実際

$$\zeta(x,y) = X(x)Y(y)$$

とすると、まず境界条件から

$$X(0) = X(L) = Y(0) = Y(H) = 0.$$

微分方程式に代入して、

$$X''(x)Y(y) + X(x)Y''(y) = -\lambda X(x)Y(y).$$

ゆえに

$$\frac{X''(x)}{X(x)} = -\lambda - \frac{Y''(y)}{Y(y)}.$$

この式の値は定数であることが分るから、それを μ とおくと、

$$X''(x) = \mu X(x), \quad Y''(y) = -(\lambda + \mu)Y(y).$$

X と μ, Y と $\lambda + \mu$ という二組のそれぞれが 1 次元の固有値問題の解ということで、

$$(\exists m \in \mathbf{N}) \quad \mu = -\left(\frac{m\pi}{L}\right)^2, \quad X(x) = C \sin \frac{m\pi x}{L},$$

$$(\exists n \in \mathbf{N}) \quad (\lambda + \mu) = \left(\frac{n\pi}{H}\right)^2, \quad Y(y) = C' \sin \frac{n\pi y}{H}.$$

ゆえに

$$(\exists (m, n) \in \mathbf{N} \times \mathbf{N}) \quad \lambda = \lambda_{mn}, \quad \zeta(x, y) = C\zeta_{mn}(x, y) \quad (C \text{ は任意定数}).$$

ただし

$$\zeta_{mn}(x, y) := \sin \frac{m\pi x}{L} \sin \frac{n\pi y}{H}, \quad \lambda_{m,n} = \left[\left(\frac{m}{L}\right)^2 + \left(\frac{n}{H}\right)^2\right] \pi^2 \quad (m,\, n \in \mathbf{N}).$$

これから熱伝導方程式と境界条件を満す特解として

$$b_{m,n} e^{-((m/L)^2 + (n/H)^2)\pi^2 t} \sin \frac{m\pi x}{L} \sin \frac{n\pi y}{H} \quad (m,\, n \in \mathbf{N})$$

が得られる (ここで $b_{m,n}$ は任意定数)。これから任意の $\{b_{mn}\}$ について、

$$u(x, y, t) := \sum_{m,n=1}^{\infty} b_{m,n} e^{-((m/L)^2 + (n/H)^2)\pi^2 t} \sin \frac{m\pi x}{L} \sin \frac{n\pi y}{H}$$

は熱伝導方程式と境界条件を満す (ただし級数がまともな収束をすることは仮定する)。後は、適当な $b_{m,n}$ を取ることによって、初期条件を満すように、すなわち

$$\sum_{m,n=1}^{\infty} b_{m,n} \sin \frac{m\pi x}{L} \sin \frac{n\pi y}{H} = f(x, y) \quad ((x, y) \in [0, L] \times [0, H])$$

が成り立つように出来ればよいが、これは以下のようにして可能である。

簡単のため、f は C^1 級で、$f = 0$ on Γ を満たす関数とする。

$y \in [0, H]$ を任意に固定して考えると、$f(\cdot, y)\colon x \mapsto f(x, y)$ は $[0, L]$ 上定義され、境界で 0 となる C^1-級関数である。ゆえに Fourier 正弦級数展開できる。つまり

$$f(x, y) = \sum_{m=1}^{\infty} B_m(y) \sin \frac{m\pi x}{L} \quad (x \in [0, L] \text{ につき一様に収束}), \quad B_m(y) := \frac{2}{L} \int_0^L f(x, y) \sin \frac{m\pi x}{L} dx.$$

ところで、$y \mapsto B_m(y)$ は $[0, H]$ 上定義された C^1- 級関数で、境界で 0 となることが容易に分るから、やはり Fourier 正弦級数展開可能である。すなわち

$$B_m(y) := \sum_{n=1}^{\infty} b_{m,n} \sin \frac{n\pi y}{H} \quad (y \in [0, H] \text{ につき一様に収束}), \quad b_{m,n} := \frac{2}{H} \int_0^H B_m(y) \sin \frac{n\pi y}{H} dy.$$

まとめると、

$$f(x, y) = \sum_{m=1}^{\infty} B_m(y) \sin \frac{m\pi x}{L} = \sum_{m=1}^{\infty} \sum_{n=1}^{\infty} b_{m,n} \sin \frac{n\pi y}{H} \sin \frac{m\pi x}{L}, \quad (\text{どんな収束？ })$$

$$b_{m,n} = \frac{2}{H} \int_0^H B_m(y) \sin \frac{n\pi y}{H} dy = \frac{2}{H} \int_0^H \left(\frac{2}{L} \int_0^L f(x, y) \sin \frac{m\pi x}{L} dx\right) \sin \frac{n\pi y}{H} dy$$

$$= \frac{4}{LH} \int_0^H \int_0^L f(x, y) \sin \frac{m\pi x}{L} \sin \frac{n\pi y}{H} dx\, dy.$$

こうして、次の解の公式が得られた。

$$u(x,y,t)=\sum_{m,n=1}^{\infty} b_{m,n}e^{-((m/L)^2+(n/H)^2)\pi^2 t}\sin\frac{m\pi x}{L}\sin\frac{n\pi y}{H},$$
$$b_{m,n}=\frac{4}{LH}\int_0^H\int_0^L f(x,y)\sin\frac{m\pi x}{L}\sin\frac{n\pi y}{H}dxdy.$$

ごちゃごちゃしているので、$L=H=1$ の場合を書いておく ($n=1$ の場合と比較してみよう)。

$$u(x,y,t)=\sum_{m,n=1}^{\infty} b_{m,n}e^{-(m^2+n^2)\pi^2 t}\sin m\pi x\sin n\pi y,\quad b_{m,n}=4\int_0^1\int_0^1 f(x,y)\sin m\pi x\sin n\pi y\,dx\,dy.$$

注意 2.2.1 (変数分離の形の固有関数を探せば十分であること) 上の議論で、$\overline{\Omega}$ で C^1-級で、$\Gamma=\partial\Omega$ で 0 となる任意の関数が $\sin\frac{m\pi x}{L}\sin\frac{n\pi y}{H}$ で展開されることが証明されているので、初期値境界値問題の解の構成には一応は成功しているが、固有値問題については、固有関数が x の関数と y の関数の積の形のものしか考えていないので、解決したとは言えない。

そこで $\triangle\zeta=-\lambda\zeta$ (in Ω), $\zeta=0$ (on Γ), $\zeta\not\equiv 0$ を満たす ζ が $\overline{\Omega}$ で十分なめらかと仮定して[6] 追求してみよう。まず

$$\zeta(x,y)=\sum_{m,n=1}^{\infty} b_{m,n}\sin\frac{m\pi x}{L}\sin\frac{n\pi y}{H}$$

と Fourier 級数展開し、方程式に代入すると

$$-\sum_{m,n=1}^{\infty} b_{m,n}\left[\left(\frac{m\pi}{L}\right)^2+\left(\frac{n\pi}{H}\right)^2\right]\sin\frac{m\pi x}{L}\sin\frac{n\pi y}{H}=-\lambda\sum_{m,n=1}^{\infty} b_{m,n}\sin\frac{m\pi x}{L}\sin\frac{n\pi y}{H}.$$

移項して

$$\sum_{m,n=1}^{\infty} b_{m,n}\left[\left(\frac{m\pi}{L}\right)^2+\left(\frac{n\pi}{H}\right)^2-\lambda\right]\sin\frac{m\pi x}{L}\sin\frac{n\pi y}{H}=0.$$

これから

$$\forall m\in\mathbf{N},\quad \forall n\in\mathbf{N},\quad b_{m,n}\left[\left(\frac{m\pi}{L}\right)^2+\left(\frac{n\pi}{H}\right)^2-\lambda\right]=0.$$

$\zeta\not\equiv 0$ としてあるので、少なくとも一組の (m,n) については $b_{m,n}\neq 0$ である。$b_{m,n}\neq 0$ となる (m,n) については、

$$(\star)\qquad \lambda=\left(\frac{m\pi}{L}\right)^2+\left(\frac{n\pi}{H}\right)^2=\lambda_{mn}.$$

ゆえに固有値は上で求めたもので尽きていることがわかる。また 1 つの λ に対して、(☆) を満たす (m,n) は有限個しか存在しないことは明らかである ($\because$ $|m|\le\frac{L}{\pi}\sqrt{\lambda}$, $|n|\le\frac{H}{\pi}\sqrt{\lambda}$ であるから)。さらに

$$\zeta(x,y)=\sum_{(m,n)\text{ は }(\star)\text{ を満たす}} b_{m,n}\sin\frac{m\pi x}{L}\sin\frac{n\pi y}{H}\in\operatorname{span}\left\{\sin\frac{m\pi x}{L}\sin\frac{n\pi y}{H};\,m,\,n\text{ は }(\star)\text{ を満たす}\right\}$$

であるから、固有空間の基底も (変数分離の形のものだけで) 求まっていることになる。

[6] もしも 1 変数 (1 次元) であれば、ζ が $-\zeta''(x)=\lambda\zeta(x)$ を満たすことから、ζ が無限回微分可能であることが導かれる (分かりますか？)。多変数の場合は、それほど簡単ではないが…

例えば $L = H = 1$ のとき、$\lambda = 5\pi^2 = (1^2 + 2^2)\pi^2$ は固有値で、$\lambda = (m^2 + n^2)\pi^2$ を満たす (m, n) は $(1, 2)$, $(2, 1)$ の 2 つである。そして

$$\zeta(x, y) = C_1 \sin \pi x \sin 2\pi y + C_2 \sin 2\pi x \sin \pi y \quad (C_1,\ C_2 \text{ は同時に } 0 \text{ とはならない任意の定数})$$

が対応する固有関数を与える。 ■

課題　同次 Neumann 境界条件を課した初期値境界値問題の解の公式を求めよ (証明が出来ればそれに越したことはないが、とりあえず公式を求めるだけでよい)。

ヒント　C^1-級の関数 $f\colon \mathbf{R} \times \mathbf{R} \to \mathbf{R}$ が

$$f(x + pL, y + qH) = f(x, y) \quad ((x, y) \in \mathbf{R}^2,\ p, q \in \mathbf{Z})$$

という周期性を持つならば、$\Omega := [0, L] \times [0, H]$ として、

$$\begin{aligned} f(x, y) = \frac{1}{4}a_{00} + \frac{1}{2}\left(\sum_{n=1}^{\infty} a_{0n} \cos\frac{2n\pi y}{H} + \sum_{m=1}^{\infty} a_{m0} \cos\frac{2m\pi x}{L} + \sum_{n=1}^{\infty} b_{0n} \sin\frac{2n\pi y}{H} + \sum_{m=1}^{\infty} c_{m0} \sin\frac{2m\pi x}{L}\right) \\ + \sum_{m,n=1}^{\infty}\left(a_{mn} \cos\frac{2m\pi x}{L}\cos\frac{2n\pi y}{H} + b_{mn} \cos\frac{2m\pi x}{L}\sin\frac{2n\pi y}{H} + c_{mn} \sin\frac{2m\pi x}{L}\cos\frac{2n\pi y}{H}\right. \\ \left. + d_{mn} \sin\frac{m\pi x}{L}\sin\frac{n\pi y}{H}\right), \end{aligned}$$

と展開できる (どういう収束?)。ここで係数は次式で与えられる。

$$\begin{aligned} a_{mn} &= \frac{4}{LH}\iint_{\Omega} f(x, y) \cos\frac{2m\pi x}{L}\cos\frac{2n\pi y}{H}\, dx\, dy, \\ b_{mn} &= \frac{4}{LH}\iint_{\Omega} f(x, y) \cos\frac{2m\pi x}{L}\sin\frac{2n\pi y}{H}\, dx\, dy, \\ c_{mn} &= \frac{4}{LH}\iint_{\Omega} f(x, y) \sin\frac{2m\pi x}{L}\cos\frac{2n\pi y}{H}\, dx\, dy, \\ d_{mn} &= \frac{4}{LH}\iint_{\Omega} f(x, y) \sin\frac{2m\pi x}{L}\sin\frac{2n\pi y}{H}\, dx\, dy. \end{aligned}$$

特に $f\colon \Omega \to \mathbf{R}$ が C^1-級かつ $f = 0$ on $\partial\Omega$ を満すならば、(周期 $2L$, $2H$ として考えて)

$$f(x, y) = \sum_{m,n=1}^{\infty} d_{mn} \sin\frac{m\pi x}{L}\sin\frac{n\pi y}{H},$$

$$d_{mn} = \frac{4}{LH}\iint_{[0,L]\times[0,H]} f(x, y) \sin\frac{m\pi x}{L}\sin\frac{n\pi y}{H}\, dx\, dy$$

展開できる。

第3章　2次元長方形領域における熱伝導方程式に対する差分法

3.1　Target 問題 (Dirichlet 境界条件)

この章では、Ω は $\mathbf{R}^2$ の開区間 $(a,b)\times(c,d)$ であるとし、$\Gamma=\partial\Omega$ とする。このとき、Ω における熱伝導方程式の初期値境界値問題

$$\frac{\partial u}{\partial t}(x,y,t)=\triangle u(x,y,t)\quad(t>0,\,(x,y)\in\Omega) \tag{3.1}$$

$$u(x,y,t)=\alpha(x,y)\quad(t>0,\,(x,y)\in\Gamma) \tag{3.2}$$

$$u(x,y,0)=f(x,y)\quad((x,y)\in\overline{\Omega}) \tag{3.3}$$

を考える。ただし

- u は $\overline{\Omega}\times[0,\infty)$ を定義域とする未知関数:
$$u\colon\overline{\Omega}\times[0,\infty)\ni(x,y,t)\longmapsto u(x,y,t)\in\mathbf{R}.$$
- α は Γ 上定義された既知関数[1]。
- f は $\overline{\Omega}$ 上定義された既知関数。
- $\triangle$ は $\mathbf{R}^2$ における Laplace 作用素 (Laplacian):
$$\triangle:=\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}.$$

既に述べたように、空間 1 次元の場合と異なり、開区間は領域としてはかなり特別なものであるので、残念ながら上の問題は十分一般的であるとは言えない。が、それでもこの問題を自在に扱えるだけの知識があれば、実り多い実験が遂行できるであろう。

以下、差分スキームと共に、プログラム例を示す。これらのプログラムを利用するにあたっては、付録の A 「サンプルプログラムについて」を見て欲しい。

3.2　陽解法 (前進 Euler 法)

領域 Ω を座標軸に平行な格子線で x 軸、y 軸方向にそれぞれ N_x 等分, N_y 等分し、格子点を (x_i,y_j) と表わす。すなわち

$$h_x:=\frac{b-a}{N_x},\quad h_y:=\frac{d-c}{N_y},$$

[1] $d=1$ のときは、領域は開区間に他ならず、その境界は二点からなる集合であったので、境界値とは二つの実数値であったが、$d\geq 2$ では境界値は関数として扱う必要がある。

さらに

$$x_i := a + ih_x \quad (i = 0, 1, \cdots, N_x),$$
$$y_j := c + jh_y \quad (j = 0, 1, \cdots, N_y)$$

とおく。

N_x を x 軸方向の分割数、N_y を y 軸方向の分割数、h_x を x 軸方向の刻み幅、h_y を y 軸方向の刻み幅と呼ぶ。

次に $\tau > 0$ を固定し、

$$t_n := n\tau \quad (n = 0, 1, 2, \cdots)$$

とおく。

後のために便利なので、

$$\lambda_x := \frac{\tau}{h_x^2}, \quad \lambda_y := \frac{\tau}{h_y^2}, \quad \lambda := \lambda_x + \lambda_y$$

とおく。

我々は (x_i, y_j, t_n) における u の値、すなわち

$$u_{i,j}^n := u(x_i, y_j, t_n) \quad (i = 0, 1, \cdots, N_x;\ j = 0, 1, \cdots, N_y;\ n = 0, 1, \cdots)$$

の近似値

$$U_{i,j}^n \quad (i = 0, 1, \cdots, N_x;\ j = 0, 1, \cdots, N_y;\ n = 0, 1, \cdots)$$

を求めることを目標にすえる。

$u_t(x_i, y_j, t_n)$ を

$$u_t(x_i, y_j, t_n) \fallingdotseq \frac{u_{i,j}^{n+1} - u_{i,j}^n}{\tau}$$

と前進差分近似し、$u_{xx}(x_i, y_j, t_n)$ と $u_{yy}(x_i, y_j, t_n)$ をそれぞれ

$$u_{xx}(x_i, y_j, t_n) \fallingdotseq \frac{u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n}{h_x^2},$$
$$u_{yy}(x_i, y_j, t_n) \fallingdotseq \frac{u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n}{h_y^2}$$

と 2 階中心差分近似することを考えると、

$$\frac{U_{i,j}^{n+1} - U_{i,j}^n}{\tau} = \frac{U_{i+1,j}^n - 2U_{i,j}^n + U_{i-1,j}^n}{h_x^2} + \frac{U_{i,j+1}^n - 2U_{i,j}^n + U_{i,j-1}^n}{h_y^2}$$

なる差分方程式を得る。両辺に τ をかけてから、移項すると

$$U_{i,j}^{n+1} = (1 - 2\lambda_x - 2\lambda_y)U_{i,j}^n + \lambda_x(U_{i+1,j}^n + U_{i-1,j}^n) + \lambda_y(U_{i,j+1}^n + U_{i,j-1}^n)$$
$$(i = 1, 2, \cdots, N_x - 1;\ j = 1, 2, \cdots, N_y - 1;\ n = 0, 1, 2, \cdots)$$

が導かれる。

一方、境界条件からは

$$U_{0,j}^n = \alpha(a, y_j), \quad U_{N_x,j}^n = \alpha(b, y_j) \quad (j = 0, 1, \cdots, N_y;\ n = 1, 2, \cdots) \tag{3.4}$$
$$U_{i,0}^n = \alpha(x_i, c), \quad U_{i,N_y}^n = \alpha(x_i, d) \quad (i = 0, 1, \cdots, N_x;\ n = 1, 2, \cdots) \tag{3.5}$$

を得る。ここで (3.4) と (3.5) には重複があるため (角点に対応している)、一つの n につき、方程式の個数は

$$2(N_x + N_y)$$

であることを注意しておこう。

初期条件からは、もちろん

$$U^0_{i,j} = f(x_i, y_j) \quad (i = 0, 1, \cdots, N_x;\ j = 0, 1, \cdots, N_y)$$

を得る。

まとめ 3 重添字を持つ未知数列 $\{U^n_{i,j}; i = 0, 1, \cdots, N_x, j = 0, 1, \cdots, N_y, n = 0, 1, 2, \cdots\}$ を定めるための差分方程式として

$$U^{n+1}_{i,j} = (1 - 2\lambda_x - 2\lambda_y)U^n_{i,j} + \lambda_x(U^n_{i+1,j} + U^n_{i-1,j}) + \lambda_y(U^n_{i,j+1} + U^n_{i,j-1}) \tag{3.6}$$
$$(i = 1, 2, \cdots, N_x - 1;\ j = 1, 2, \cdots, N_y - 1;\ n = 0, 1, 2, \cdots),$$

$$U^{n+1}_{i,j} = \alpha(x_i, y_j) \tag{3.7}$$
$$((i, j) \in \{0, N_x\} \times \{0, 1, \ldots, N_y\} \cup \{0, 1, \cdots, N_x\} \times \{0, N_y\};\ n = 0, 1, \cdots),$$

$$U^0_{i,j} = f(x_i, y_j) \quad (i = 0, 1, \cdots, N_x;\ j = 0, 1, \cdots, N_y) \tag{3.8}$$

を考える。

上の方程式は一意可解である。言い換えると、(3.6), (3.7), (3.8) によって、$\{U^n_{i,j}\}$ は一通りに定まる。そのことは空間 1 次元の場合と同様にして分かる。

$0 < \lambda \leq 1/2$ であるときに、差分スキームが安定であり、$N \to \infty$ のとき差分解が偏微分方程式の厳密解に収束することは、空間 1 次元の場合とまったく同様である。

ここで説明したアルゴリズムによるプログラムが

https://m-katsurada.sakura.ne.jp/program/fdm/heat2d-e.c

から入手できる。

入手&コンパイル (cglsc が必要)

```
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/heat2d-e.c
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/smallmatrix.h
cglsc heat2d-e.c
./heat2d-e
```

色々尋ねられるが、例えば

```
Nx, Ny: 100 100
Tmax: 1
τ: 1e-5
Δt: 0.001
```

3.3 陰解法

3.3.1 領域を格子に分割する

Ω を座標軸に平行な格子線で x 軸, y 軸方向にそれぞれ N_x 等分、 N_y 等分し、格子点を (x_i, y_j) と表す。すなわち

$$h_x := \frac{b-a}{N_x}, \quad h_y := \frac{d-c}{N_y}$$

として

$$x_i := a + ih_x \quad (i = 0, 1, \cdots, N_x), \quad y_j := c + jh_y \quad (j = 0, 1, \cdots, N_y)$$

とおく。

一方 $\tau > 0$ を固定して、

$$t_n := n\tau \quad (n = 0, 1, 2, \cdots)$$

とおく。

我々の目標は (x_i, y_j, t_n) における u の値の近似値を求めることである。

以下現れる様々な方程式で、二重添字 (i, j) の範囲を明確にするため、

$$\begin{aligned} \omega &:= \{(i,j); 1 \le i \le N_x - 1,\ 1 \le j \le N_y - 1\} \\ \overline{\omega} &:= \{(i,j); 0 \le i \le N_x,\ 0 \le j \le N_y\} \\ \gamma &:= \overline{\omega} \setminus \omega \end{aligned}$$

とおき[23]、さらに

$$\begin{aligned} u_{i,j}^n &:= u(x_i, y_j, t_n) \quad ((i,j) \in \overline{\omega};\ n = 0, 1, 2, \cdots) \\ f_{i,j} &:= f(x_i, y_j) \quad ((i,j) \in \overline{\omega}) \\ \alpha_{i,j} &:= \alpha(x_i, y_j) \quad ((i,j) \in \gamma). \end{aligned}$$

とおく。

後でしばしば、方程式や未知数の個数を勘定することになる。その準備として、集合 $\omega, \overline{\omega}, \gamma$ の要素の個数を数えると、

$$\begin{aligned} \sharp\omega &= (N_x - 1)(N_y - 1), \\ \sharp\overline{\omega} &= (N_x + 1)(N_y + 1), \\ \sharp\gamma &= 2(N_x + N_y) \end{aligned}$$

となる。ただし集合 A に属する要素の個数を $\sharp A$ と表した。$\sharp\overline{\omega}$ は以下何回も登場するので、

$$N := \sharp\overline{\omega} = (N_x + 1)(N_y + 1)$$

とおこう。

後でよく出て来るので

$$\lambda_x := \frac{\tau}{h_x^2}, \quad \lambda_y := \frac{\tau}{h_y^2}$$

とおく。

---

[2] これは、この文書オリジナルな記法だが、それぞれ $\Omega, \overline{\Omega}, \Gamma$ に属する格子点の添字に対応すると言えば、筆者の気持は分かってもらえると信じる。

[3] $A \setminus B = \{x; x \in A$ かつ $x \notin B\}$.

3.3.2　偏微分方程式を離散化して出来る差分方程式

$u_{i,j}^n$ の近似 $U_{i,j}^n$ を求めることを目標にする。$0 \leq \theta \leq 1$ なるパラメーター θ を導入して、熱伝導方程式をいわゆる θ-法で離散化すると

$$\begin{aligned}\frac{U_{i,j}^{n+1}-U_{i,j}^n}{\tau} &= (1-\theta)\left(\frac{U_{i+1,j}^n-2U_{i,j}^n+U_{i-1,j}^n}{h_x^2}+\frac{U_{i,j+1}^n-2U_{i,j}^n+U_{i,j-1}^n}{h_y^2}\right)\\ &\quad+\theta\left(\frac{U_{i+1,j}^{n+1}-2U_{i,j}^{n+1}+U_{i-1,j}^{n+1}}{h_x^2}+\frac{U_{i,j+1}^{n+1}-2U_{i,j}^{n+1}+U_{i,j-1}^{n+1}}{h_y^2}\right)\end{aligned}$$

なる差分方程式を得る。添字 (i,j) の範囲は通常[4] $(i,j)\in\omega$ である。この式の両辺に τ をかけて移項して整理すると、

$$\begin{aligned}&[1+2\theta(\lambda_x+\lambda_y)]U_{i,j}^{n+1}-\theta[\lambda_x(U_{i+1,j}^{n+1}+U_{i-1,j}^{n+1})+\lambda_y(U_{i,j+1}^{n+1}+U_{i,j-1}^{n+1})]\\ &\quad=[1-2(1-\theta)(\lambda_x+\lambda_y)]U_{i,j}^n+(1-\theta)[\lambda_x(U_{i+1,j}^n+U_{i-1,j}^n)+\lambda_y(U_{i,j+1}^n+U_{i,j-1}^n)]\\ &\qquad((i,j)\in\omega;\ n=0,1,2,\cdots).\end{aligned}\tag{3.9}$$

3.3.3　差分解の決定 — 連立方程式の行列、ベクトル表現

一方、境界条件からは

$$U_{i,j}^{n+1}=\alpha_{i,j}\quad((i,j)\in\gamma;\ n=0,1,2,\cdots),\tag{3.10}$$

初期条件からは

$$U_{i,j}^0=f_{i,j}\quad((i,j)\in\overline{\omega})\tag{3.11}$$

を得る。

(3.9), (3.10), (3.11) から 3 重添字を持つ数列 $\{U_{i,j}^n;(i,j)\in\overline{\omega},n=0,1,2,\cdots\}$ は一意に決定されることを以下で示す。

そのため、まず $n=0,1,2,\cdots$ に対して

$$U^n:=\{U_{i,j}^n\}_{(i,j)\in\overline{\omega}}$$

とおく (二重添字を持つ有限数列)。各 n に対して U^n は N 次元のベクトルとみなせる。空間 1 次元の時と同様に、 n が小さい方から順に U^n を求めて行く。

U^0 は (3.11) で一意に定まる。

U^n が求まっているとき、(3.9), (3.10) は N 個の未知数に関する N 個の 1 次方程式になっている[5]。よって、

$$N\text{ 次正方行列}\cdot N\text{ 次元未知ベクトル}=N\text{ 次元既知ベクトル}$$

の形の連立 1 次方程式に表される (行の個数 = 方程式の個数=未知数の個数 = 列の個数)。

この方程式を実際に書き下すためには、$U^n=\{U_{i,j}^n\}_{(i,j)\in\overline{\omega}}$ を 1 次元的に並べなければならないが、そのためには $\overline{\omega}$ から $\{0,1,2,\cdots,N-1\}$ への全単射を一つ選べばよい。例えば

$$\varphi(i,j):=j(N_x+1)+i$$

[4] 仮想格子点などを導入しなければ、という意味である。

[5] (3.9) は、$\sharp\omega$ 個、(3.10) は $\sharp\gamma$ 個、合わせて $\sharp\omega+\sharp\gamma=\sharp\overline{\omega}=N$ 個。

とおくと、

$$\varphi\colon \overline{\omega} \longrightarrow \{0,1,2,\cdots,N-1\}$$

は全単射になる[6] (図を描いて φ の意味を考えよ[7])。ちなみに φ の逆写像 φ^{-1} は $\ell \in \{0,1,2,\cdots,N-1\}$ に対して

$$\varphi^{-1}(\ell) = (\ell \bmod (N_x+1), \ell \operatorname{div} (N_x+1))$$

となる[8]。ここで $a \bmod b$ は a を b で割った余り[9]、$a \operatorname{div} b$ は a を b で割った商[10]を表すとする[11]。

$$U^n_\ell := U^n_{i,j}, \quad \ell = \varphi(i,j) \qquad ((i,j) \in \overline{\omega},\ n = 0,1,2,\cdots)$$

によって、U^n_ℓ $(0 \le \ell \le N-1,\ n \ge 0)$ を定めて

$$\vec{U}^{n+1} := \begin{pmatrix} U^{n+1}_0 \\ U^{n+1}_1 \\ \vdots \\ U^{n+1}_{N-1} \end{pmatrix}$$

とおいて $\vec{U}^{n+1} \in \mathbf{R}^N$ を作り、(3.9), (3.10) を $\vec{U}^{n+1}$ についての連立 1 次方程式とみなそう。

$(i,j) \in \omega$ とするとき、$\ell = \varphi(i,j)$ とおくと、

$$\varphi(i+1,j) = \ell+1, \quad \varphi(i-1,j) = \ell-1, \quad \varphi(i,j+1) = \ell+m, \quad \varphi(i,j-1) = \ell-m$$

となる。ただし $m := N_x+1$ とした。これから

$$U^{n+1}_{i,j} = U^{n+1}_\ell, \quad U^{n+1}_{i+1,j} = U^{n+1}_{\ell+1}, \quad U^{n+1}_{i-1,j} = U^{n+1}_{\ell-1}, \quad U^{n+1}_{i,j+1} = U^{n+1}_{\ell+m}, \quad U^{n+1}_{i,j-1} = U^{n+1}_{\ell-m}.$$

すると、この (i,j) についての (3.9) は、

$$[1+2\theta(\lambda_x+\lambda_y)]U^{n+1}_\ell - \theta[\lambda_x(U^{n+1}_{\ell+1}+U^{n+1}_{\ell-1}) + \lambda_y(U^{n+1}_{\ell+m}+U^{n+1}_{\ell-m})] = b^n_\ell$$

と表せる。ただし、

(3.12) $\quad b^n_\ell :=$ (3.9) の右辺

$$= [1-2(1-\theta)(\lambda_x+\lambda_y)]U^n_{i,j} + (1-\theta)[\lambda_x(U^n_{i+1,j}+U^n_{i-1,j}) + \lambda_y(U^n_{i,j+1}+U^n_{i,j-1})]$$

とおいた。また、この方程式は

$$\begin{array}{ccccccccccccc} (0,\cdots,0, & -\theta\lambda_y, & 0,\cdots,0, & -\theta\lambda_x, & 1+2\theta(\lambda_x+\lambda_y), & -\theta\lambda_x, & 0,\cdots,0, & -\theta\lambda_y, & 0,\cdots,0)\vec{U}^{n+1} = b^n_\ell \\ & \uparrow & & \uparrow & \uparrow & \uparrow & & \uparrow & \\ & \ell-m & & \ell-1 & \ell & \ell+1 & & \ell+m & \end{array}$$

と、

---

[6] 要するに、重複も抜けもなく、きれいに番号が付きましたね、ということ。

[7] $(0,0)$, $(1,0)$, $(2,0)$, …, $(N_x,0)$, $(0,1)$, $(1,1)$, …, $(N_x,1)$, …, $(0,N_y)$, $(1,N_y)$, …, (N_x,N_y) に順に 0, 1, …, $N-1$ という番号を振る。誰か TEX に貼込める図を描いてくれませんか？

[8] よく分からない場合は、$N_x = 5$, $N_y = 4$ の場合に図を描いて、格子点に番号 ℓ をふり、ℓ から (i,j) がどう求まるか眺めながら考えよ。図を描くと簡単に分かる。最初から図を描いておけばよいようだが、我々はプログラムを書く必要があるので、やりたいことを式で表現しなければならない。

[9] C 言語ならば `a % b` と書くところ。

[10] C 言語ならば、(`a`, `b` が整数型の式として) 単に `a / b` と書くところ。しかし、これは数学における / の意味とは食い違うので、この C 流を採用するわけにはいかないだろう。

[11] 「a を b で割った商が q, 余りが r」を "$a \operatorname{div} b = q$, $a \bmod b = r$" と書くということ。これは $a = bq+r$, $0 \le r < b$ と同値 (というよりもこれが定義)。

横ベクトル × 縦ベクトル = 実数

の形にも書ける。

一方、境界条件の方は

$$\alpha_\ell := \alpha_{i,j}, \qquad \ell = \varphi(i,j)$$

とおくことにより、

$$U_\ell^{n+1} = \alpha_\ell \quad (\ell \in \varphi(\gamma))$$

と表せ、

$$(0,\cdots,0,\ \underset{\uparrow}{1},\ 0,\cdots,0)\vec{U}^{n+1} = \alpha_\ell \qquad \ell$$

とも書ける。

以上をまとめると

$$A\vec{U}^{n+1} = \vec{b} \tag{3.13}$$

の形の方程式が立てられる。ここで A は N 次正方行列で、第 $\ell \in \varphi(\omega)$ 行は

$$(0,\cdots,0,\ \underset{\ell-m}{-\theta\lambda_y},\ 0,\cdots,0,\ \underset{\ell-1}{-\theta\lambda_x},\ \underset{\ell}{1+2\theta(\lambda_x+\lambda_y)},\ \underset{\ell+1}{-\theta\lambda_x},\ 0,\cdots,0,\ \underset{\ell+m}{-\theta\lambda_y},\ 0,\cdots,0)$$

の形で、第 $\ell \in \varphi(\gamma)$ 行は

$$(0,\cdots,0,\ \underset{\ell}{1},\ 0,\cdots,0)$$

の形になっている。また $\vec{b}$ は $\mathbf{R}^N$ の既知要素であり、第 $\ell \in \varphi(\omega)$ 成分は (3.12) で定義される b_ℓ^n であり、第 $\ell \in \varphi(\gamma)$ 成分は α_ℓ に等しい。

(3.13) の具体例を二つ示そう。$N_x = N_y = 2$ の場合は

$$\begin{pmatrix} 1 & & & & & & & & \\ & 1 & & & & & & & \\ & & 1 & & & & & & \\ & & & 1 & & & & & \\ 0 & -\theta\lambda_y & 0 & -\theta\lambda_x & 1+2\theta(\lambda_x+\lambda_y) & -\theta\lambda_x & 0 & -\theta\lambda_y & 0 \\ & & & & & 1 & & & \\ & & & & & & 1 & & \\ & & & & & & & 1 & \\ & & & & & & & & 1 \end{pmatrix} \begin{pmatrix} U_0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \\ U_6 \\ U_7 \\ U_8 \end{pmatrix} = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ b_4 \\ \alpha_5 \\ \alpha_6 \\ \alpha_7 \\ \alpha_8 \end{pmatrix}$$

となる[12]。ここで U_i, b_i は、本当はそれぞれ U_i^{n+1}, b_i^n と書くべきだが、うるさいので右肩の $n+1$, n は省略した。最初の行を 0 番と数えるとき、第 4 行のみが θ, λ_x, λ_y の登場する行になっているが、これは 4 番目の格子点だけが、内部に属する格子点であること (つまり $\varphi(\omega) = \{4\}$) から明らかである。

[12] $\omega = \{(1,1)\}$, $\varphi(\omega) = \{4\}$ であるから、第 4 行[13]以外は境界上にある格子点に対応している。

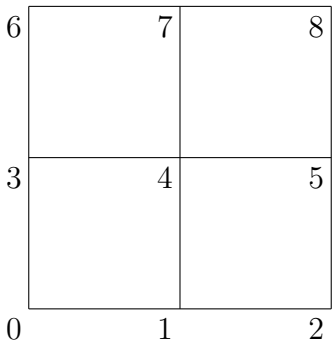


図 3.1: $N_x = 2$, $N_y = 2$ の場合の $\varphi(i,j)$

また $N_x = N_y = 3$ の場合は

$$\beta_0 = 1 + 2\theta(\lambda_x + \lambda_y), \quad \beta_1 = -\theta\lambda_x, \quad \beta_2 = -\theta\lambda_y$$

とおくと、

$$\begin{pmatrix}
1 & & & & & & & & & & & & & & & \\
 & 1 & & & & & & & & & & & & & & \\
 & & 1 & & & & & & & & & & & & & \\
 & & & 1 & & & & & & & & & & & & \\
 & & & & 1 & & & & & & & & & & & \\
\beta_2 & & & \beta_1 & \beta_0 & \beta_1 & & & \beta_2 & & & & & & & \\
 & \beta_2 & & & \beta_1 & \beta_0 & \beta_1 & & & \beta_2 & & & & & & \\
 & & & & & & 1 & & & & & & & & & \\
 & & & & & & & 1 & & & & & & & & \\
 & & & & \beta_2 & & & \beta_1 & \beta_0 & \beta_1 & & & \beta_2 & & & \\
 & & & & & \beta_2 & & & \beta_1 & \beta_0 & \beta_1 & & & \beta_2 & & \\
 & & & & & & & & & & 1 & & & & & \\
 & & & & & & & & & & & 1 & & & & \\
 & & & & & & & & & & & & 1 & & & \\
 & & & & & & & & & & & & & 1 & & \\
 & & & & & & & & & & & & & & 1 &
\end{pmatrix}
\begin{pmatrix} U_0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \\ U_6 \\ U_7 \\ U_8 \\ U_9 \\ U_{10} \\ U_{11} \\ U_{12} \\ U_{13} \\ U_{14} \\ U_{15} \end{pmatrix}
=
\begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ b_5 \\ b_6 \\ \alpha_7 \\ \alpha_8 \\ b_9 \\ b_{10} \\ \alpha_{11} \\ \alpha_{12} \\ \alpha_{13} \\ \alpha_{14} \\ \alpha_{15} \end{pmatrix}$$

となる[14]。

これ以外の場合にも、このように連立 1 次方程式 (3.13) を組み立てることは容易である。

しかし、問題はこれで終ったわけではない。まず

連立 1 次方程式 (3.13) は一意に解けるか?

幸い、この問に対する答は Yes. なぜならば、この方程式の係数行列は狭義の優対角行列であるから[15] 正則である (『応用数学』第 7 章「発展系の数値解析」 [14] で、空間 1 次元の場合の差分方程式の場合に同様の論法を用いた)。

[14] $\omega = \{(1,1),(2,1),(1,2),(2,2)\}$, $\varphi(\omega) = \{5,6,9,10\}$ である。

[15] $|\beta_0| - 2|\beta_1| - 2|\beta_2| = 1 > 0$.

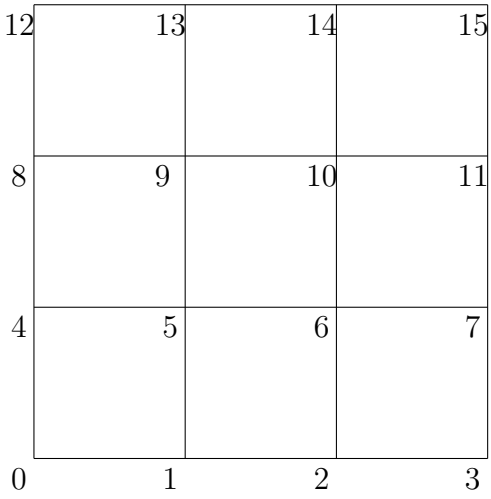


図 3.2: $N_x = 3$, $N_y = 3$ の場合の $\varphi(i, j)$

3.3.4 対称な連立 1 次方程式への変形

連立 1 次方程式 (3.13) をどうやって解くか?

という問題が残っている。上の方程式の係数行列を見ると、ほとんど[16]の行は

$$(0 \cdots 0\ \beta_2\ 0 \cdots 0\ \beta_1\ \beta_0\ \beta_1\ 0 \cdots 0\ \beta_2\ 0 \cdots 0)$$

であるから、「対称行列に近い」ことが分かる。連立 1 次方程式や固有値問題においては

対称な問題は、対称な問題向けの手法を用いて、効率的に解くべきである

という鉄則がある[17]。実はもともとの偏微分方程式の初期値境界値問題 (3.1), (3.2), (3.3) は対称な問題であるため、離散化して得られた連立 1 次方程式 (3.13) も「ほとんど対称」になったのである。ここは頑張って、方程式 (3.13) に修正を加えて、(完全な) 対称行列を係数とする連立 1 次方程式に書き換えるべきである。

[16] N_x, N_y が大きいとき、$\sharp\omega \gg \sharp\gamma$.
[17] 「対称な問題を解くのは science であるが、非対称な問題を解くのは art である」という言葉がある。

一般的な話をする前に、$N_x = N_y = 3$ の場合で論じよう。

$$
\begin{pmatrix}
1 & & & & & & & & & & & & & & & \\
 & 1 & & & & & & & & & & & & & & \\
 & & 1 & & & & & & & & & & & & & \\
 & & & 1 & & & & & & & & & & & & \\
 & & & & 1 & & & & & & & & & & & \\
\boxed{\beta_2} & & & \boxed{\beta_1} & \beta_0 & \beta_1 & & & \beta_2 & & & & & & & \\
 & \boxed{\beta_2} & & & \beta_1 & \beta_0 & \boxed{\beta_1} & & & \beta_2 & & & & & & \\
 & & & & & & 1 & & & & & & & & & \\
 & & & & & & & 1 & & & & & & & & \\
 & & & & \beta_2 & & & \boxed{\beta_1} & \beta_0 & \beta_1 & & & \boxed{\beta_2} & & & \\
 & & & & & \beta_2 & & & \beta_1 & \beta_0 & \boxed{\beta_1} & & & \boxed{\beta_2} & & \\
 & & & & & & & & & & 1 & & & & & \\
 & & & & & & & & & & & 1 & & & & \\
 & & & & & & & & & & & & 1 & & & \\
 & & & & & & & & & & & & & 1 & & \\
 & & & & & & & & & & & & & & 1 &
\end{pmatrix}
\begin{pmatrix} U_0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \\ U_6 \\ U_7 \\ U_8 \\ U_9 \\ U_{10} \\ U_{11} \\ U_{12} \\ U_{13} \\ U_{14} \\ U_{15} \end{pmatrix}
=
\begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ b_5 \\ b_6 \\ \alpha_7 \\ \alpha_8 \\ b_9 \\ b_{10} \\ \alpha_{11} \\ \alpha_{12} \\ \alpha_{13} \\ \alpha_{14} \\ \alpha_{15} \end{pmatrix}
$$

において $\boxed{}$ でくくった成分は、相棒 (対角線に関して対称の位置にある成分) が 0 であるのに、自分自身は 0 でないので、対称性を壊していると言える。ところが、この成分とかけあわせる U_ℓ は実は境界値に対応しているので、既知である。これを利用するため、上の方程式を

$$
\begin{pmatrix}
1 & & & & & & & & & & & & & & & \\
 & 1 & & & & & & & & & & & & & & \\
 & & 1 & & & & & & & & & & & & & \\
 & & & 1 & & & & & & & & & & & & \\
 & & & & 1 & & & & & & & & & & & \\
 & & & & & \beta_0 & \beta_1 & & \beta_2 & & & & & & & \\
 & & & & & \beta_1 & \beta_0 & & & \beta_2 & & & & & & \\
 & & & & & & & 1 & & & & & & & & \\
 & & & & & & & & 1 & & & & & & & \\
 & & & & & \beta_2 & & & & \beta_0 & \beta_1 & & & & & \\
 & & & & & & \beta_2 & & & \beta_1 & \beta_0 & & & & & \\
 & & & & & & & & & & & 1 & & & & \\
 & & & & & & & & & & & & 1 & & & \\
 & & & & & & & & & & & & & 1 & & \\
 & & & & & & & & & & & & & & 1 & \\
 & & & & & & & & & & & & & & & 1
\end{pmatrix}
\begin{pmatrix} U_0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \\ U_6 \\ U_7 \\ U_8 \\ U_9 \\ U_{10} \\ U_{11} \\ U_{12} \\ U_{13} \\ U_{14} \\ U_{15} \end{pmatrix}
=
\begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ b_5 \\ b_6 \\ \alpha_7 \\ \alpha_8 \\ b_9 \\ b_{10} \\ \alpha_{11} \\ \alpha_{12} \\ \alpha_{13} \\ \alpha_{14} \\ \alpha_{15} \end{pmatrix}
-
\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \beta_2 U_1 + \beta_1 U_4 \\ \beta_2 U_2 + \beta_1 U_7 \\ 0 \\ 0 \\ \beta_1 U_8 + \beta_2 U_{13} \\ \beta_1 U_{11} + \beta_2 U_{14} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}
$$

のように変形する[18]。これはめでたく対称な方程式になった! 右辺は実は

$$\begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ b_5 \\ b_6 \\ \alpha_7 \\ \alpha_8 \\ b_9 \\ b_{10} \\ \alpha_{11} \\ \alpha_{12} \\ \alpha_{13} \\ \alpha_{14} \\ \alpha_{15} \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \beta_2\alpha_1 + \beta_1\alpha_4 \\ \beta_2\alpha_2 + \beta_1\alpha_7 \\ 0 \\ 0 \\ \beta_1\alpha_8 + \beta_2\alpha_{13} \\ \beta_1\alpha_{11} + \beta_2\alpha_{14} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

であり、既知の値からすぐに計算できることに注意しよう。

以上の操作を一般的に論じよう。$(i,j) \in \varpi$ として、$\ell = \varphi(i,j)$ とおく。

1. $(i,j) \in \gamma$ の場合は ℓ 番目の方程式は
$$U_\ell = \alpha_\ell$$
であるから、これは対角成分以外は 0 なので書き換えの必要はない。

2. $(i,j) \in \omega$ としよう。対応する差分方程式は
$$\beta_2 U_{\ell-m} + \beta_1 U_{\ell-1} + \beta_0 U_\ell + \beta_1 U_{\ell+1} + \beta_2 U_{\ell+m} = b_\ell$$
である。つまり A の (p,q) 成分を a_{pq} とするとき、
$$\sum_{q=0}^{N-1} a_{\ell q} U_q = b_\ell, \tag{3.14}$$
ただし
$$\begin{aligned} a_{\ell,\ell-m} &= \beta_2, \quad a_{\ell,\ell-1} = \beta_1, \quad a_{\ell,\ell} = \beta_0, \quad a_{\ell,\ell+1} = \beta_1, \quad a_{\ell,\ell+m} = \beta_2, \\ a_{\ell,q} &= 0 \quad (q \neq \ell-m, \ell-1, \ell, \ell+1, \ell+m) \end{aligned}$$
である。non-zero 成分 $a_{\ell,\ell-m}$, $a_{\ell,\ell-1}$, $a_{\ell,\ell}$, $a_{\ell,\ell+1}$, $a_{\ell,\ell+m}$ と対称な位置にある成分をチェックする。

   (a) $a_{\ell,\ell-m} = a_{\ell-m,\ell}$ か? $a_{\ell-m,\ell} = a_{\ell-m,(\ell-m)+m}$ であるから、これは $\ell - m \in \varphi(\omega)$ か?という条件で置き換えられる。(もし $\ell - m \in \varphi(\omega)$ ならば $a_{\ell-m,(\ell-m)+m} = \beta_2 = a_{\ell,\ell-m}$.)

[18] この部分の操作を、方程式の左辺の項を右辺に移項したものであると説明してきたが、ある学生に (拡大係数行列の) 「基本変形」と言われた。なるほど、そうも言えるし、その方が分かりやすいかもしれない。

(b) $a_{\ell,\ell-1} = a_{\ell-1,\ell}$ か? $a_{\ell-1,\ell} = a_{\ell-1,(\ell-1)+1}$ であるから、これは $\ell - 1 \in \varphi(\omega)$ か?という条件で置き換えられる。(もし $\ell - 1 \in \varphi(\omega)$ ならば $a_{\ell-1,(\ell-1)+1} = \beta_1 = a_{\ell,\ell-1}$.)

(c) $a_{\ell,\ell} = a_{\ell,\ell}$ は当り前。

(d) $a_{\ell,\ell+1} = a_{\ell+1,\ell}$ か? $a_{\ell+1,\ell} = a_{\ell+1,(\ell+1)-1}$ であるから、これは $\ell + 1 \in \varphi(\omega)$ か?という条件で置き換えられる。

(e) $a_{\ell,\ell+m} = a_{\ell+m,\ell}$ か? $a_{\ell+m,\ell} = a_{\ell+m,(\ell+m)-m}$ であるから、これは $\ell + m \in \varphi(\omega)$ か?という条件で置き換えられる。

言い換えると、$(i, j) \in \omega$ について、

(a) (i, j) の左右上下 $(i \pm 1, j)$, $(i, j \pm 1)$ がやはり ω に含まれている場合 (直観的には (i, j) が γ から離れている場合)、第 $\ell = \varphi(i, j)$ 行は何も書き換える必要はない。

(b) (i, j) の左右上下 $(i \pm 1, j)$, $(i, j \pm 1)$ の少なくとも一つが ω に含まれない (γ に含まれる) 場合は[19]、対称性を崩している要素があるので、第 $\ell = \varphi(i, j)$ 行は書き換えが必要である。以下の 4 つの条件は排反というわけではないことに注意。

i. $j = 1$ の場合、$(i, j - 1) \in \gamma$, すなわち $\ell - m \in \varphi(\gamma)$ なので

$$a_{\ell,\ell-m} = \beta_2, \quad a_{\ell-m,\ell} = 0$$

そこで (3.14) において、 $a_{\ell,\ell-m}U_{\ell-m}$ を右辺に移項する。

ii. $i = 1$ の場合、$(i - 1, j) \in \gamma$, すなわち $\ell - 1 \in \varphi(\gamma)$ なので

$$a_{\ell,\ell-1} = \beta_1, \quad a_{\ell-1,\ell} = 0$$

そこで (3.14) において、$a_{\ell,\ell-1}U_{\ell-1}$ を右辺に移項する。

iii. $i = N_x - 1$ の場合、$(i + 1, j) \in \gamma$, すなわち $\ell + 1 \in \varphi(\gamma)$ なので

$$a_{\ell,\ell+1} = \beta_1, \quad a_{\ell+1,\ell} = 0$$

そこで (3.14) において、 $a_{\ell,\ell+1}U_{\ell+1}$ を右辺に移項する。

iv. $j = N_y - 1$ の場合、$(i, j + 1) \in \gamma$, すなわち $\ell + m \in \varphi(\gamma)$ なので

$$a_{\ell,\ell+m} = \beta_2, \quad a_{\ell+m,\ell} = 0$$

そこで (3.14) において、 $a_{\ell,\ell+m}U_{\ell+m}$ を右辺に移項する。

3.3.5 連立 1 次方程式を組み立てる手順のまとめ

前節までに説明した手順をまとめておこう[20]。実際にコンパイルして動くプログラムのソースを掲げる[21]。ここに掲げるプログラムは説明用であって、大変に素朴な書き方をしている (実行効率に大きな問題がある)。係数行列が帯(おび)行列 (band matrix) であることをまったく利用していない。

[19]格子点で言うと、境界から 1 段ずれているところにあるものである。
[20]アルゴリズムの紹介にはプログラムを使うな、という意見も根強いが、読みやすく書けば構わないだろう。
[21]https://m-katsurada.sakura.ne.jp/program/ から入手可能である。

注意 3.3.1 (decomp(), solve() について) 以下に紹介するプログラムのいくつかで decomp(), solve() という関数を用いている。それぞれ行列を LU 分解するための関数と LU 分解を利用して連立 1 次方程式を解くための関数である。Forsythe and Moler の著書のプログラムを参考にして作成したものである。代替物で置き換えた方がよいでしょうか… ■

注意 3.3.2 (安定性条件) θ 法の安定性条件は、1 次元とほとんど同様に

$$\lambda \leq \frac{1}{2(1-\theta)}$$

であることが証明できる (後述)。ただし

$$\lambda := \lambda_x + \lambda_y, \quad \lambda_x := \frac{\tau}{h_x^2}, \quad \lambda_x := \frac{\tau}{h_y^2}.$$

τ について解くと

$$\tau \leq \frac{1}{2(1-\theta)} \times \left(\frac{1}{h_x^2} + \frac{1}{h_y^2} \right)^{-1}$$

となる。 ■

```
 1  /*
 2   * heat2d-i-naive.c --- 2 次元熱方程式を陰解法で素朴に解く (version 3.1)
 3   * https://m-katsurada.sakura.ne.jp/program/fdm/heat2d-i-naive.c
 4   * https://m-katsurada.sakura.ne.jp/program/fdm/smallmatrix.h
 5   * https://m-katsurada.sakura.ne.jp/program/linear/lu.c
 6   * https://m-katsurada.sakura.ne.jp/program/linear/lu.h
 7   * To get these files
 8   *     curl -O https://m-katsurada.sakura.ne.jp/program/fdm/heat2d-i-naive.c
 9   *     curl -O https://m-katsurada.sakura.ne.jp/program/fdm/smallmatrix.h
10   *     curl -O https://m-katsurada.sakura.ne.jp/program/linear/lu.c
11   *     curl -O https://m-katsurada.sakura.ne.jp/program/linear/lu.c
12   * To compile
13   *     cglsc heat2d-i-naive.c lu.c
14   *
15   */
16
17  #include <stdio.h>
18  #include <stdlib.h>
19  #include <math.h>
20  #ifdef OLD
21  #include <matrix.h>
22  #else
23  #include "smallmatrix.h"
24  #endif
25  #ifndef G_DOUBLE
26  #define G_DOUBLE
27  #endif
28  #include <glsc.h>
29  #include "lu.h"
30
31  #define phi(i,j) (j)*m+(i)
32
33  int main(void)
34  {
35      double a, b, c, d;
36      int N_x, N_y, m, N, i, j, p, q, L, n, nMax;
```

```
37      matrix Uk, A;
38      double *B, *vector_U, cond;
39      int *iwork, skip;
40      double h_x, h_y, lambda_x, lambda_y, lambda, lambda_limit, tau, Tmax, dt;
41      double f(double, double), alpha(double, double);
42      double theta, beta_0, beta_1, beta_2, beta_00, beta_10, beta_20;
43      double x, y, t;
44
45      /* 問題を考える区間 [a,b] × [c,d] */
46      a = 0.0; b = 1.0; c = 0.0; d = 1.0;
47
48      /* 区間の分割数 */
49      printf("Nx, Ny: "); scanf("%d %d", &N_x, &N_y);
50
51      m = N_x + 1;
52      N = (N_x + 1) * (N_y + 1);
53      /* 空間の刻み幅 */
54      h_x = (b - a) / N_x;
55      h_y = (d - c) / N_y;
56
57      /* 行列、ベクトルを記憶する変数のメモリー割り当て */
58      if ((Uk = new_matrix(N_x + 1, N_y + 1)) == NULL) {
59          fprintf(stderr, "数列 U^k を記憶する領域の確保に失敗\n");
60          exit(1);
61      }
62      if ((A = new_matrix(N, N)) == NULL) {
63          fprintf(stderr, "係数行列 A を記憶する領域の確保に失敗\n");
64          exit(1);
65      }
66      if ((B = malloc(sizeof(double) * N)) == NULL) {
67          fprintf(stderr, "B を記憶する領域の確保に失敗\n");
68          exit(1);
69      }
70      if ((vector_U = malloc(sizeof(double) * N)) == NULL) {
71          fprintf(stderr, "vector_U を記憶する領域の確保に失敗\n");
72          exit(1);
73      }
74      if ((iwork = malloc(sizeof(int) * N)) == NULL) {
75          fprintf(stderr, "iwork を記憶する領域の確保に失敗\n");
76          exit(1);
77      }
78
79      /* θ法の重みの決定 */
80      printf("θ (0 ≦ θ ≦ 1): "); scanf("%lf", &theta);
81
82      if (theta == 1.0) {
83          printf("τ: "); scanf("%lf", &tau);
84      } else {
85          printf("τ (≦%g ≡最大値ノルムに関する安定性条件を満たすτの上限): ",
86                 0.5 / (1 - theta) / (1 / (h_x * h_x) + 1 / (h_y * h_y)));
87          scanf("%lf", &tau);
88      }
89
90      lambda_x = tau / (h_x * h_x);
91      lambda_y = tau / (h_y * h_y);
92      lambda = lambda_x + lambda_y;
93
94      /* 最大値ノルムに関する安定性を満たすλの上限 */
95      lambda_limit = 1.0 / (2.0 * (1.0 - theta));
96
```

```
 97      if (lambda > lambda_limit)
 98          printf("注意: λ=%g>1/2(1-θ) となっています。\n", lambda);
 99      else
100          printf("λ=%g\n", lambda);
101
102      /* 初期値の設定 */
103      for (i = 0; i <= N_x; i++) {
104        x = a + i * h_x;
105        for (j = 0; j <= N_y; j++)
106          Uk[i][j] = f(x, c + j * h_y);
107      }
108
109      /* 連立 1 次方程式に現れる係数 */
110      beta_0 = 1.0 + 2.0 * theta * lambda;
111      beta_1 = - theta * lambda_x;
112      beta_2 = - theta * lambda_y;
113
114      beta_00 = 1.0 - 2.0 * (1.0 - theta) * lambda;
115      beta_10 = (1.0 - theta) * lambda_x;
116      beta_20 = (1.0 - theta) * lambda_y;
117
118      /* 係数行列の作成 */
119      /* まず 0 クリア */
120      for (p = 0; p < N; p++) {
121          for (q = 0; q < N; q++)
122              A[p][q] = 0.0;
123          A[p][p] = 1.0;
124      }
125      for (i = 1; i < N_x; i++)
126          for (j = 1; j < N_y; j++) {
127              L = phi(i, j);
128              A[L][L - m] = beta_2;
129              A[L][L - 1] = beta_1;
130              A[L][L]     = beta_0;
131              A[L][L + 1] = beta_1;
132              A[L][L + m] = beta_2;
133          }
134
135      /* 対称化するための作業 1 */
136      for (j = 1; j < N_y; j++) {
137          /* (1,j) */
138          L = phi(1, j);
139          A[L][L - 1] = 0.0;
140          /* (N_x-1,j) */
141          L = phi(N_x - 1, j);
142          A[L][L + 1] = 0.0;
143      }
144
145      /* 対称化するための作業 2 */
146      for (i = 1; i < N_x; i++) {
147          /* (i,1) */
148          L = phi(i, 1);
149          A[L][L - m] = 0.0;
150          /* (i,N_y-1) */
151          L = phi(i, N_y - 1);
152          A[L][L + m] = 0.0;
153      }
154
155      /* 連立 1 次方程式の係数行列を表示する */
156      if (N < 20) {
```

```
157         printf("素朴に作った連立 1 次方程式の行列\n");
158         for (p = 0; p < N; p++) {
159           for (q = 0; q < N; q++)
160             printf(" %4.1f", A[p][q]);
161           printf("\n");
162         }
163       }
164
165       printf("備考: 1+2 θ λ=%4.1f, -θ λ x=%5.1f, -θ λ y=%5.1f\n",
166              beta_0, beta_1, beta_2);
167
168       printf("Tmax: "); scanf("%lf", &Tmax);
169       printf("Δ t: ");  scanf("%lf", &dt);
170       skip = rint(dt / tau);
171       if (skip == 0) {
172           skip = 1;
173       }
174       dt = skip * tau;
175
176       nMax = rint(Tmax / tau);
177
178       /* グラフィックス・ライブラリィ GLSC の呼び出し */
179       g_init("Meta", 250.0, 160.0);
180       g_device(G_BOTH);
181       g_def_scale(0,
182                   0.0, 1.0, 0.0, 1.0,
183                   30.0, 70.0, 100.0, 72.0);
184       g_def_scale(4,
185                   -1.0, 1.0, -1.0, 1.0,
186                   30.0, 30.0, 100.0, 100.0);
187       g_def_line(0, G_BLACK, 0, G_LINE_SOLID);
188       g_sel_scale(0);
189
190       g_cls();
191   #ifdef OLD
192       g_hidden2(1.0, 1.0, 0.4, -1.0, 1.0, 5.0, 25.0, 20.0, 20.0, 20.0,
193                 150.0, 100.0, Uk, N_x + 1, N_y + 1,
194                 1, G_SIDE_NONE, 2, 1);
195   #else
196       g_hidden(1.0, 1.0, 0.4, -1.0, 1.0, 5.0, 25.0, 20.0, 20.0, 20.0,
197                150.0, 100.0, &Uk[0][0], N_x + 1, N_y + 1,
198                1, G_SIDE_NONE, 2, 1);
199   #endif
200       /* 係数行列 LU 分解 */
201       decomp(N, A, &cond, iwork, B);
202       if (cond + 1 == cond) {
203           /* 条件数が大きければ、計算をあきらめる */
204           printf("MATRIX IS SINGULAR TO WORKING PRECISION\n");
205           return 0;
206       }
207
208       /* 時間に関するループ */
209       for (n = 1; n <= nMax; n++) {
210
211           /* まず、素朴な連立 1 次方程式の右辺を用意する */
212           /* 内部の格子点 */
213           for (i = 1; i < N_x; i++)
214               for (j = 1; j < N_y; j++) {
215                   L = phi(i, j);
216                   B[L] = beta_00 * Uk[i][j]
```

```
217                     + beta_10 * (Uk[i + 1][j] + Uk[i - 1][j])
218                     + beta_20 * (Uk[i][j + 1] + Uk[i][j - 1]);
219             }
220         /* 下の辺、上の辺にある格子点（角の点も含める）*/
221         for (i = 0; i <= N_x; i++) {
222             x = a + i * h_x;
223             /* (i, 0) */
224             L = phi(i, 0);
225             B[L] = alpha(x, c);
226             /* (i, N_y) */
227             L = phi(i, N_y);
228             B[L] = alpha(x, d);
229         }
230         /* 左の辺、右の辺にある格子点（角の点は含めない）*/
231         for (j = 1; j < N_y; j++) {
232             y = c + j * h_y;
233             /* (0, j) */
234             L = phi(0, j);
235             B[L] = alpha(a, y);
236             /* (N_x, j) */
237             L = phi(N_x, j);
238             B[L] = alpha(b, y);
239         }
240
241         /* 対称化する */
242         /* 対称化するための作業 1 */
243         for (j = 1; j < N_y; j++) {
244             y = c + j * h_y;
245             /* (1,j) のとき a_{l,l-1}U_{l-1}=β 1 U_{l-1} を移項する */
246             L = phi(1, j);
247             B[L] -= beta_1 * alpha(a, y);
248             /* (N_x-1,j) のとき a_{l,l+1}U_{l+1}=β 1 U_{l+1} を移項する */
249             L = phi(N_x - 1, j);
250             B[L] -= beta_1 * alpha(b, y);
251         }
252         /* 対称化するための作業 2 */
253         for (i = 1; i < N_x; i++) {
254             x = a + i * h_x;
255             /* (i,1) のとき a_{l,l-m}U_{l-m}=β 2 U_{l-m} を移項する */
256             L = phi(i, 1);
257             B[L] -= beta_2 * alpha(x, c);
258             /* (i,N_y-1) のとき a_{l,l+m}U_{l+m}=β 2 U_{l+m} を移項する */
259             L = phi(i, N_y - 1);
260             B[L] -= beta_2 * alpha(x, d);
261         }
262
263         /* A vector_U = B を解く */
264         solve(N, A, B, iwork);
265         /* */
266         for (i = 0; i <= N_x; i++)
267             for (j = 0; j <= N_y; j++)
268                 Uk[i][j] = B[phi(i,j)];
269
270         /* データを数値で表示 */
271         if (n % skip == 0) {
272 #ifdef  PRINT
273             for (i = 0; i <= N_x; i++) {
274                 for (j = 0; j <= N_y; j++)
275                     printf(" %5.1f", Uk[i][j]);
276                 printf("\n");
```

```
277             }
278 #endif
279
280             /* 鳥瞰図を描く */
281             g_cls();
282 #ifdef OLD
283             g_hidden2(1.0, 1.0, 0.4, -1.0, 1.0, 5.0, 25.0, 20.0, 20.0, 20.0,
284                       150.0, 100.0, Uk, N_x + 1, N_y + 1,
285                       1, G_SIDE_NONE, 2, 1);
286 #else
287             g_hidden(1.0, 1.0, 0.4, -1.0, 1.0, 5.0, 25.0, 20.0, 20.0, 20.0,
288                      150.0, 100.0, &Uk[0][0], N_x + 1, N_y + 1,
289                      1, G_SIDE_NONE, 2, 1);
290 #endif
291         }
292     }
293     /* マウスでクリックされるのを待つ */
294     g_sleep(-1.0);
295     /* ウィンドウを消す */
296     g_term();
297
298     return 0;
299 }
300
301 /* 初期値 */
302 double f(double x, double y)
303 {
304     /* ピラミッド型の関数 */
305     if (y > 0.5)
306         y = 1 - y;
307     if (x > 0.5)
308         x = 1 - x;
309     if (y < x)
310         return 5 * y;
311     else
312         return 5 * x;
313 }
314
315 /* 境界値 */
316 double alpha(double x, double y)
317 {
318     /* 同次 Dirichlet 境界条件 */
319     return 0.0;
320 }
```

入手＆コンパイル (cglsc が必要)

```
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/heat2d-i-naive.c
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/smallmatrix.h
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/lu.c
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/lu.h
cglsc heat2d-i-naive.c lu.c
./heat2d-i-naive
```

色々尋ねられるが、例えば

```
Nx, Ny: 50 50
θ (0≦θ≦1): 0.5
τ (≦0.0002 ≡最大値ノルムに関する安定性条件を満たすτの上限): 0.0001
λ=0.5
備考: 1+2 θ λ= 1.5, -θ λ x= -0.1, -θ λ y= -0.1
Tmax: 1
Δt: 0.001
```

(動作チェック: 2024/8/22)

3.3.6 係数行列が帯行列であることを利用したプログラム

(ここは記録として残しておくが、中途半端なので読むときはカットして構わない。)

『Gauss の消去法と LU 分解』の章で紹介した帯行列向け LU 分解用関数 `bandlu()`, `bandsol()` を利用するサンプル・プログラム https://m-katsurada.sakura.ne.jp/program/fdm/heat2d-i-band3.c を紹介しておく (プログラム・リストは省略)。

入手＆コンパイル (cglsc が必要)

```
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/heat2d-i-band3.c
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/smallmatrix.h
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/luband3.c
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/luband3.h
cglsc heat2d-i-band3.c luband3.c
```

(動作チェック: 2024/8/22)

実はこの熱方程式を解くプログラムはさらに改良が可能である。それは差分方程式の係数行列が対称であることをまだ利用していないからである。

3.3.7 境界上の格子点における値を未知数に含めないプログラム

(1.3「非同次 Dirichlet 境界条件」を見ることを勧める。)

境界上の格子点における値 U_{ij}^n $((i,j) \in \gamma)$ も未知数として方程式に組み込んだが、これは境界条件からすぐに分るので、これを除いた連立 1 次方程式を立てることもできる。それには、$U^n =$

$\{U^n_{i,j}\}_{(i,j)\in\omega}$ を 1 次元的に並べなければならないが、そのためには ω から $\{0,1,2,\cdots,N'-1\}$ への全単射を一つ選べばよい ($N' := (N_x-1)(N_y-1)$)。例えば

$$\psi(i,j) := (j-1)(N_x-1)+i-1$$

とおくと、

$$\psi\colon \omega \longrightarrow \{0,1,2,\cdots,N'-1\}$$

は全単射になる。逆写像は $\ell \in \{0,1,2,\cdots,N'-1\}$ に対して

$$\psi^{-1}(\ell) = (\ell \bmod (N_x-1)+1, \ell \operatorname{div} (N_x-1)+1)$$

となる。この ψ を用いて

$$U^n_\ell := U^n_{i,j}, \quad \ell = \psi(i,j) \quad ((i,j)\in\omega,\, n=0,1,2,\cdots)$$

によって U^n_ℓ を定めて、

$$\vec{U}^n := \begin{pmatrix} U^n_0 \\ U^n_1 \\ \vdots \\ U^n_{N'-1} \end{pmatrix}$$

とおく。今度は

$$m' := N_x - 1$$

とおくと、$\ell = \psi(i,j)$ とするとき、

$$\psi(i+1,j) = \ell+1, \quad \psi(i-1,j) = \ell-1, \quad \psi(i,j+1) = \ell+m', \quad \psi(i,j-1) = \ell-m'$$

となる。これを用いて

$$\beta_2 U^{n+1}_{i,j-1} + \beta_1 U^{n+1}_{i-1,j} + \beta_0 U^{n+1}_{i,j} + \beta_1 U^{n+1}_{i+1,j} + \beta_2 U^{n+1}_{i,j+1} = b_\ell \quad (1\le i\le N_x-1,\, 1\le j\le N_y-1)$$

を書き換えることになる。

結局、連立 1 次方程式は以下のようになる。

$$A\vec{U}^{n+1} = \vec{b}^n + \vec{c}^n.$$

ここで

$$A = \left(\begin{array}{cccc|cccc|cccc|cccc}
\beta_0 & \beta_1 & & & \beta_2 & & & & & & & & & & & \\
\beta_1 & \ddots & \ddots & & & \beta_2 & & & & & & & & & & \\
 & \ddots & & \beta_1 & & & \ddots & & & & & & & & & \\
 & & \beta_1 & \beta_0 & & & & \beta_2 & & & & & & & & \\
\hline
\beta_2 & & & & \ddots & & & & \ddots & & & & & & & \\
 & \beta_2 & & & & \ddots & & & & \ddots & & & & & & \\
 & & \ddots & & & & \ddots & & & & \ddots & & & & & \\
 & & & \beta_2 & & & & \ddots & & & & \ddots & & & & \\
\hline
 & & & & \ddots & & & & \ddots & & & & \beta_2 & & & \\
 & & & & & \ddots & & & & \ddots & & & & \beta_2 & & \\
 & & & & & & \ddots & & & & \ddots & & & & \ddots & \\
 & & & & & & & \ddots & & & & \ddots & & & & \beta_2 \\
\hline
 & & & & & & & & \beta_2 & & & & \beta_0 & \beta_1 & & \\
 & & & & & & & & & \beta_2 & & & \beta_1 & \ddots & \ddots & \\
 & & & & & & & & & & \ddots & & & \ddots & & \beta_1 \\
 & & & & & & & & & & & \beta_2 & & & \beta_1 & \beta_0
\end{array}\right),$$

$\vec{b}^n = (b_0, b_1, \ldots, b_{N'-1})^T$.

また $\vec{c}^n = (c_0^n, c_1^n, \ldots, c_{N'-1}^n)^T$ は、最初に $\vec{c}^n = \vec{0}$ としておいてから、以下の (1)〜(4) を行って計算できる。

(1) $\ell = \psi(i, 1)$ $(1 \le i \le N_x - 1)$ に対して

$$c_\ell^n \leftarrow c_\ell^n - \beta_2 \alpha_{i,0}.$$

(2) $\ell = \psi(i, N_y - 1)$ $(1 \le i \le N_x - 1)$ に対して

$$c_\ell^n \leftarrow c_\ell^n - \beta_2 \alpha_{i,N_y}.$$

(3) $\ell = \psi(1, j)$ $(1 \le j \le N_y - 1)$ に対して

$$c_\ell^n \leftarrow c_\ell^n - \beta_1 \alpha_{0,j}.$$

(4) $\ell = \psi(N_x - 1, j)$ $(1 \le j \le N_y - 1)$ に対して

$$c_\ell^n \leftarrow c_\ell^n - \beta_1 \alpha_{N_x,j}.$$

$N_x = N_y = 3$ の場合は、

$$\left(\begin{array}{cc|cc}
\beta_0 & \beta_1 & \beta_2 & 0 \\
\beta_1 & \beta_0 & 0 & \beta_2 \\
\hline
\beta_2 & 0 & \beta_0 & \beta_1 \\
0 & \beta_2 & \beta_1 & \beta_0
\end{array}\right)
\left(\begin{array}{c} U_0 \\ U_1 \\ U_2 \\ U_3 \end{array}\right)$$

以下手書きノートの内容を書き足すつもり。(と書いたけれど、ノートはどこかに行ってしまった。)

サンプル・プログラム

https://m-katsurada.sakura.ne.jp/program/fdm/heat2d-i-naive2.c

を公開している。

コンパイルの仕方 (cglsc が必要)

```
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/heat2d-i-naive2.c
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/smallmatrix.h
curl -O https://m-katsurada.sakura.ne.jp/program/linear/lu.c
curl -O https://m-katsurada.sakura.ne.jp/program/linear/lu.h
cglsc heat2d-i-naive2.c lu.c
```

(動作チェック: 2024/8/22)

A の各ブロックは $m' = N_x - 1$ 次の正方行列で、それが縦横ともに $N_y - 1$ 個並んでいる。ゆえに

$$A = \begin{pmatrix} A' & \beta_2 I & & & \\ \beta_2 I & A' & \beta_2 I & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_2 I & A' & \beta_2 I \\ & & & \beta_2 I & A' \end{pmatrix}, \quad A' = \begin{pmatrix} \beta_0 & \beta_1 & & & \\ \beta_1 & \beta_0 & \beta_1 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_1 & \beta_0 & \beta_1 \\ & & & \beta_1 & \beta_0 \end{pmatrix}. \tag{3.15}$$

実行結果 (係数行列に注目)

```
% cglsc heat2d-i-naive2.c lu.c
% ./heat2d-i-naive2
Nx, Ny: 4 4
θ (0 ≦θ≦ 1): 0.5
τ (≦ 0.03125 ＝最大値ノルムに関する安定性条件を満たすτの上限): 0.01
λ=0.32
連立 1 次方程式の行列
   1.32  -0.08      0  -0.08      0      0      0      0      0
  -0.08   1.32  -0.08      0  -0.08      0      0      0      0
      0  -0.08   1.32      0      0  -0.08      0      0      0
  -0.08      0      0   1.32  -0.08      0  -0.08      0      0
      0  -0.08      0  -0.08   1.32  -0.08      0  -0.08      0
      0      0  -0.08      0  -0.08   1.32      0      0  -0.08
      0      0      0  -0.08      0      0   1.32  -0.08      0
      0      0      0      0  -0.08      0  -0.08   1.32  -0.08
      0      0      0      0      0  -0.08      0  -0.08   1.32
備考: 1+2 θλ=1.32, -θλ x= -0.08, -θλ y= -0.08
Tmax: 1
Δ t: 0.01
% ./heat2d-i-naive2
Nx, Ny: 5 5
θ (0 ≦θ≦ 1): 0.5
τ (≦ 0.02 ＝最大値ノルムに関する安定性条件を満たすτの上限): 0.01
λ=0.5
連立 1 次方程式の行列
    1.5 -0.125      0      0 -0.125      0      0      0      0      0      0      0      0      0      0      0
 -0.125    1.5 -0.125      0      0 -0.125      0      0      0      0      0      0      0      0      0      0
      0 -0.125    1.5 -0.125      0      0 -0.125      0      0      0      0      0      0      0      0      0
      0      0 -0.125    1.5      0      0      0 -0.125      0      0      0      0      0      0      0      0
 -0.125      0      0      0    1.5 -0.125      0      0 -0.125      0      0      0      0      0      0      0
      0 -0.125      0      0 -0.125    1.5 -0.125      0      0 -0.125      0      0      0      0      0      0
      0      0 -0.125      0      0 -0.125    1.5 -0.125      0      0 -0.125      0      0      0      0      0
      0      0      0 -0.125      0      0 -0.125    1.5      0      0      0 -0.125      0      0      0      0
      0      0      0      0 -0.125      0      0      0    1.5 -0.125      0      0 -0.125      0      0      0
      0      0      0      0      0 -0.125      0      0 -0.125    1.5 -0.125      0      0 -0.125      0      0
      0      0      0      0      0      0 -0.125      0      0 -0.125    1.5 -0.125      0      0 -0.125      0
      0      0      0      0      0      0      0 -0.125      0      0      0    1.5 -0.125      0      0      0
      0      0      0      0      0      0      0      0 -0.125      0      0 -0.125    1.5 -0.125      0      0
      0      0      0      0      0      0      0      0      0 -0.125      0      0 -0.125    1.5 -0.125      0
      0      0      0      0      0      0      0      0      0      0 -0.125      0      0 -0.125    1.5 -0.125
      0      0      0      0      0      0      0      0      0      0      0 -0.125      0      0 -0.125    1.5
備考: 1+2 θλ= 1.5, -θλ x=-0.125, -θλ y=-0.125
Tmax: 1
Δ t: 0.01
%
(動作チェック: 2024/8/22)
```

3.3.8 Kronecker 積を使った表現

前項に現われた係数行列 A は、行列の Kronecker 積[22]を用いると簡潔に書けることを示そう。

$$m' = N_x - 1, \quad \ell' = N_y - 1, \quad N = (N_x - 1)(N_y - 1)$$

とおく。

(3.15) における A は、

$$a = \theta\lambda_x, \quad b = \theta\lambda_y$$

とおくと、次のようになる:

$$\left(\begin{array}{ccc|ccc|ccc|ccc}
1+2(a+b) & -a & & -b & & & & & & & & \\
-a & \ddots & \ddots & & -b & & & & & & & \\
 & \ddots & & & & \ddots & & & & & & \\
 & & -a & & & & -b & & & & & \\
 & -a & 1+2(a+b) & & & & & & & & & \\
\hline
-b & & & \ddots & & & \ddots & & & & & \\
 & -b & & & \ddots & & & \ddots & & & & \\
 & & \ddots & & & \ddots & & & \ddots & & & \\
 & & -b & & & \ddots & & & \ddots & & & \\
\hline
 & & & \ddots & & & \ddots & & & -b & & \\
 & & & & \ddots & & & \ddots & & & -b & \\
 & & & & & \ddots & & & \ddots & & & -b \\
 & & & & & \ddots & & & \ddots & & & \\
\hline
 & & & & & & -b & & & 1+2(a+b) & -a & \\
 & & & & & & & -b & & -a & \ddots & \ddots \\
 & & & & & & & & \ddots & & \ddots & -a \\
 & & & & & & & & -b & & -a & 1+2(a+b)
\end{array}\right)$$

となる。各ブロックは $m' = N_x - 1$ 次の正方行列で、それが縦横ともに $\ell' = N_y - 1$ 個並んでいる。

$k \in \mathbf{N}$ に対して、

$$I_k = k \text{ 次単位行列}, \quad O_k = k \text{ 次零行列}, \quad J_k = \begin{pmatrix} 0 & 1 & & \\ 1 & 0 & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & 0 \end{pmatrix} \in \mathbf{R}^{k\times k}$$

と書くことにすると、

$$A = [1 + 2\theta(\lambda_x + \lambda_y)]\, I_{m'\ell'} - \theta\lambda_x \begin{pmatrix} J_{m'} & & & \\ & J_{m'} & & \\ & & \ddots & \\ & & & J_{m'} \end{pmatrix} - \theta\lambda_y \begin{pmatrix} O_{m'} & I_{m'} & & \\ I_{m'} & O_{m'} & \ddots & \\ & \ddots & \ddots & I_{m'} \\ & & I_{m'} & O_{m'} \end{pmatrix}$$

と書ける。

Kronecker 積の記法を用いると

$$A = [1 + 2\theta(\lambda_x + \lambda_y)]\, I_{\ell'm'} - \theta\lambda_x I_{\ell'} \otimes J_{m'} - \theta\lambda_y J_{\ell'} \otimes I_{m'},$$

[22]Kronecker 積について、例えば伊理 [15] を見よ。

あるいは ($I_{\ell' m'} = I_{\ell'} \otimes I_{m'}$ であるので)

$$A = [1 + 2\theta(\lambda_x + \lambda_y)]\, I_{\ell'} \otimes I_{m'} - \theta\lambda_x I_{\ell'} \otimes J_{m'} - \theta\lambda_y J_{\ell'} \otimes I_{m'}.$$

ℓ', m' よりも N_x, N_y が見えた方が良いかもしれない。

$$A = [1 + 2\theta(\lambda_x + \lambda_y)]\, I_{N_y-1} \otimes I_{N_x-1} - \theta\lambda_x I_{N_y-1} \otimes J_{N_x-1} - \theta\lambda_y J_{N_y-1} \otimes I_{N_x-1}.$$

観察

- 大部分の行は 0 でない成分が 5 個ある。こういう行は上下左右の格子点が領域内部に含まれている格子点 (x_i, y_j) に対応している。
- 対角線の両隣は大部分が $-\theta\lambda_x$ であるが、左右いずれかの格子点が境界に属するような格子点に対応する行については、境界に対応するところに 0 が現われる。
- 対角線から m だけ離れたところは、$-\theta\lambda_y$ となる。これは例外がない。意外なようだが、上下いずれかの格子点が境界に属するような行は行列の上と下の部分で、対応する成分が行列の外に出てしまうのである。

上は、$\ell = \psi(i, j) = (j - 1)(N_x - 1) + i - 1$ という番号づけを行った場合であるが、$\ell = (i - 1)(N_y - 1) + j - 1$ という番号づけを行った場合には

$$A = [1 + 2\theta(\lambda_x + \lambda_y)]\, I_{N_x-1} \otimes I_{N_y-1} - \theta\lambda_x J_{N_x-1} \otimes I_{N_y-1} - \theta\lambda_y I_{N_x-1} \otimes J_{N_y-1}$$

となる (Kronecker 積の順序が変わっただけである)。

MATLAB には Kronecker 積を計算する `kron()` という命令があり、それを用いると A を作るのは簡単である。しかし工夫をしないと効率の高いプログラムにはならないであろう。(2024/8/22 加筆: これを書いたときから、MATLAB の利用についてもそれなりに経験を積んでいて、`kron()` を利用して大丈夫と考えるようになっている。すでに別文書を公開中である (桂田 [16], [17], [18], [19])。そのうち、こちらの記述を直すかもしれない。)

3 次元の場合

"2 次元と同様にして"、直方体 $[a, b] \times [c, d] \times [e, f]$ を $N_x \times N_y \times N_z$ 等分するとき、

$$k' := N_z - 1, \quad h_z := \frac{f - e}{N_z}, \quad \lambda_z := \frac{\tau}{h_z^2}$$

とおくと、連立 1 次方程式の係数行列は

$$A = [1 + 2\theta(\lambda_x + \lambda_y + \lambda_z)]\, I_k \otimes I_\ell \otimes I_m - \theta\lambda_x I_{k'} \otimes I_\ell \otimes J_m - \theta\lambda_y I_{k'} \otimes J_\ell \otimes I_m - \theta\lambda_z J_{k'} \otimes I_\ell \otimes I_m.$$

ただし、格子点 (x_i, y_j, z_k) を 1 次元的に並べるときの番号づけを

$$\psi(i, j, k) = (i - 1) + (j - 1)(N_x - 1) + (k - 1)(N_x - 1)(N_y - 1)$$

とする (おっと、k という字がだぶった…そのため片方を k' としてあります。i, j と ℓ, m だったのを i, j, k と k', ℓ, m とした訳です)。

Kronecker 積による表現の応用として、§4.6 「ADI 法の安定性解析」と §4.7 「θ 法の安定性解析」をあげておく。

3.3.9 対称帯行列向け LU 分解の利用 heat2d-i.c (このレベルで一番効率的なプログラム)

§3.3.7, §3.3.8 の $A = (a_{ij})$ は、対称かつ半バンド幅が m, すなわち

$$|i - j| > m \quad \Longrightarrow \quad a_{ij} = 0$$

となる行列である。

以下に掲げるのは、『連立 1 次方程式 I』 (桂田 [20]) の 5.8 節で紹介した symbandlu.c[23]を利用したプログラム heat2d-i.c である。

コンパイルの仕方 (cglsc が必要)

```
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/heat2d-i.c
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/smallmatrix.h
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/symbandlu.c
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/symbandlu.h
cglsc heat2d-i.c symbandlu.c
```

このプログラムは効率的なので、Nx, Ny を大きくしても十分なスピードで動く。

```
Nx, Ny: 100 100
θ (0≦θ≦1): 0.5
τ (≦5e-05 ≡最大値ノルムに関する安定性条件を満たすτの上限): 0.0001
注意: λ=2>1/2(1-θ) となっています。
備考: 1+2 θ λ= 3.0, -θ λ x= -0.5, -θ λ y= -0.5
Tmax: 1
Δt: 0.005
```

(動作チェック: 2024/8/22)

[23]実対称帯行列を係数とする連立 1 次方程式を解くための関数の symbandlu(), synbandsolv() がある。

駆け足で symbandlu.c の説明

詳しくは、桂田 [20] を見てもらいたいが、要点を二つほど書いておく。

1. すべての首座小行列式の値が 0 でない行列 A は、単位下三角行列 L と上三角行列 U の積 $A = LU$ として表されるが、A が対称である場合、$L = (\ell_{ij})$ の対角線より下の成分は

$$\ell_{ji} = u_{ij}/u_{ii}$$

により簡単に求められるので、L を計算したり、ましてや記憶しておく必要はない。

2. 行列 $A = (a_{ij})$ が半バンド幅 m である、すなわち

$$|i-j| > m \quad \Longrightarrow \quad a_{ij} = 0$$

を満たし、さらに対称である場合、a_{ij} $(1 \le i \le n;\, i \le j \le i+m)$ だけ記憶しておけば十分である。そこで n 行 $m+1$ 列の行列 $A' = (a'_{ij})$ $(1 \le i \le n,\, 0 \le j \le m)$ を

$$a_{ij} = a'_{i,j-i}$$

で定め、この A' を記憶すればよい。

heat2d-i.c

```
 1  /*
 2   * heat2d-i.c --- 2 次元熱方程式を陰解法で解く
 3   *                境界にある格子点における値は未知数に含めない方法
 4   *                係数行列が対称帯行列であることを利用して解く
 5   *                (GLSC, matrix library を利用)
 6   * https://m-katsurada.sakura.ne.jp/program/fdm/heat2d-i.c
 7   * https://m-katsurada.sakura.ne.jp/program/fdm/smallmatrix.h
 8   * https://m-katsurada.sakura.ne.jp/program/fdm/symbandlu.c
 9   * https://m-katsurada.sakura.ne.jp/program/fdm/symbandlu.h
10   *
11   * To get these files
12         curl -O https://m-katsurada.sakura.ne.jp/program/fdm/heat2d-i.c
13         curl -O https://m-katsurada.sakura.ne.jp/program/fdm/smallmatrix.h
14         curl -O https://m-katsurada.sakura.ne.jp/program/fdm/symbandlu.c
15         curl -O https://m-katsurada.sakura.ne.jp/program/fdm/symbandlu.h
16   *
17   * To compile
18         cglsc heat2d-i.c symbandlu.c
19   * or
20   *     ccmg heat2d-i.c symbandlu.c -DOLD
21   *
22   */
23
24  #include <stdio.h>
25  #include <stdlib.h>
26  #include <math.h>
27  #ifdef OLD
28  #include <matrix.h>
29  #else
30  #include "smallmatrix.h"
31  #endif
32  #ifndef G_DOUBLE
```

```
33  #define G_DOUBLE
34  #endif
35  #include <glsc.h>
36  #include "symbandlu.h"
37
38  #define psi(i,j) (((j)-1)*mm+(i)-1)
39
40  int main()
41  {
42      double a, b, c, d;
43      int N_x, N_y, mm, NN, i, j, p, q, L, n, nMax;
44      matrix Uk, A;
45      double *B, *vector_U;
46      int *iwork, skip;
47      double h_x, h_y, lambda_x, lambda_y, lambda, lambda_limit, tau, Tmax, dt;
48      double f(double, double), alpha(double, double);
49      double theta, beta_0, beta_1, beta_2, beta_00, beta_10, beta_20;
50      double x, y, t;
51
52      /* 問題を考える区間 [a,b] × [c,d] */
53      a = 0.0; b = 1.0; c = 0.0; d = 1.0;
54
55      /* 区間の分割数 */
56      printf("Nx, Ny: "); scanf("%d %d", &N_x, &N_y);
57
58      mm = N_x - 1;
59      NN = (N_x - 1) * (N_y - 1);
60      /* 空間の刻み幅 */
61      h_x = (b - a) / N_x;
62      h_y = (d - c) / N_y;
63
64      /* 行列、ベクトルを記憶する変数のメモリー割り当て */
65      if ((Uk = new_matrix(N_x + 1, N_y + 1)) == NULL) {
66          fprintf(stderr, "数列 U^k を記憶する領域の確保に失敗\n");
67          exit(1);
68      }
69      if ((A = new_matrix(NN, mm + 1)) == NULL) {
70          fprintf(stderr, "係数行列 A を記憶する領域の確保に失敗\n");
71          exit(1);
72      }
73      if ((B = malloc(sizeof(double) * NN)) == NULL) {
74          fprintf(stderr, "B を記憶する領域の確保に失敗\n");
75          exit(1);
76      }
77      if ((vector_U = malloc(sizeof(double) * NN)) == NULL) {
78          fprintf(stderr, "vector_U を記憶する領域の確保に失敗\n");
79          exit(1);
80      }
81      if ((iwork = malloc(sizeof(int) * NN)) == NULL) {
82          fprintf(stderr, "iwork を記憶する領域の確保に失敗\n");
83          exit(1);
84      }
85
86      /* θ法の重みの決定 */
87      printf("θ (0 ≦ θ ≦ 1): "); scanf("%lf", &theta);
88
89      if (theta == 1.0) {
90          printf("τ: "); scanf("%lf", &tau);
91      } else {
92          printf("τ (≦%g ≡最大値ノルムに関する安定性条件を満たすτの上限): ",
```

```
 93                0.5 / (1 - theta) / (1 / (h_x * h_x) + 1 / (h_y * h_y)));
 94          scanf("%lf", &tau);
 95      }
 96
 97      lambda_x = tau / (h_x * h_x);
 98      lambda_y = tau / (h_y * h_y);
 99      lambda = lambda_x + lambda_y;
100
101      /* 最大値ノルムに関する安定性を満たすλの上限 */
102      lambda_limit = 1.0 / (2.0 * (1.0 - theta));
103
104      if (lambda > lambda_limit)
105          printf("注意: λ=%g>1/2(1-θ) となっています。\n", lambda);
106      else
107          printf("λ=%g\n", lambda);
108
109      /* 初期値の設定 */
110      for (i = 0; i <= N_x; i++) {
111        x = a + i * h_x;
112        for (j = 0; j <= N_y; j++)
113          Uk[i][j] = f(x, c + j * h_y);
114      }
115
116      /* 連立 1 次方程式に現れる係数 */
117      beta_0 = 1.0 + 2.0 * theta * lambda;
118      beta_1 = -theta * lambda_x;
119      beta_2 = -theta * lambda_y;
120
121      beta_00 = 1.0 - 2.0 * (1.0 - theta) * lambda;
122      beta_10 = (1.0 - theta) * lambda_x;
123      beta_20 = (1.0 - theta) * lambda_y;
124
125      /* 係数行列の作成 */
126      /* まず 0 クリア */
127      for (p = 0; p < NN; p++) {
128          for (q = 0; q <= mm; q++)
129              A[p][q] = 0.0;
130      }
131      for (i = 1; i < N_x; i++)
132          for (j = 1; j < N_y; j++) {
133              L = psi(i, j);
134       /*
135              if (j != 1) A[L][L - mm] = beta_2;
136              if (i != 1) A[L][L -  1] = beta_1;
137       */
138              A[L][0] = beta_0;     /* A[L][L] = beta_0; */
139              if (i != N_x - 1)
140                 A[L][1] = beta_1;  /* A[L][L +  1] = beta_1; */
141              if (j != N_y - 1)
142                 A[L][mm] = beta_2; /* A[L][L + mm] = beta_2; */
143          }
144
145      /* 連立 1 次方程式の係数行列を表示する */
146      if (NN < 20) {
147        printf("連立 1 次方程式の行列\n");
148        for (p = 0; p < NN; p++) {
149          for (q = 0; q <= mm; q++)
150            printf(" %4.1f", A[p][q]);
151          printf("\n");
152        }
```

```
153     }
154
155     printf("備考: 1+2 θ λ=%4.1f, -θ λ x=%5.1f, -θ λ y=%5.1f\n",
156            beta_0, beta_1, beta_2);
157
158     printf("Tmax: "); scanf("%lf", &Tmax);
159     printf("Δ t: ");  scanf("%lf", &dt);
160     skip = rint(dt / tau);
161     if (skip == 0) {
162        skip = 1;
163     }
164     dt = skip * tau;
165
166     nMax = rint(Tmax / tau);
167
168     /* グラフィックス・ライブラリィ GLSC の呼び出し */
169     g_init("Meta", 250.0, 160.0);
170     g_device(G_BOTH);
171     g_def_scale(0,
172                 0.0, 1.0, 0.0, 1.0,
173                 30.0, 70.0, 100.0, 72.0);
174     g_def_scale(4,
175                 -1.0, 1.0, -1.0, 1.0,
176                 30.0, 30.0, 100.0, 100.0);
177     g_def_line(0, G_BLACK, 0, G_LINE_SOLID);
178     g_sel_scale(0);
179
180     g_cls();
181     g_hidden2(1.0, 1.0, 0.4, -1.0, 1.0, 5.0, 25.0, 20.0, 20.0, 20.0,
182               150.0, 100.0, Uk, N_x + 1, N_y + 1,
183               1, G_SIDE_NONE, 2, 1);
184
185     /* 係数行列 LU 分解 */
186     symbandlu(A, NN, mm+1);
187
188     /* 時間に関するループ */
189     for (n = 1; n <= nMax; n++) {
190
191         /* まず、素朴な連立 1 次方程式の右辺を用意する */
192         /* 第 n ステップにおける値の寄与 */
193         for (i = 1; i < N_x; i++)
194             for (j = 1; j < N_y; j++) {
195                 L = psi(i,j);
196                 B[L] = beta_00 * Uk[i][j]
197                     + beta_10 * (Uk[i + 1][j] + Uk[i - 1][j])
198                     + beta_20 * (Uk[i][j + 1] + Uk[i][j - 1]);
199             }
200         /* 第 n+1 ステップの境界上の格子点における値（既知）の寄与 */
201         for (j = 1; j < N_y; j++) {
202             y = c + j * h_y;
203             /* (1,j) のとき */
204             L = psi(1, j);
205             B[L] -= beta_1 * alpha(a, y);
206             /* (N_x-1,j) のとき */
207             L = psi(N_x - 1, j);
208             B[L] -= beta_1 * alpha(b, y);
209         }
210         for (i = 1; i < N_x; i++) {
211             x = a + i * h_x;
212             /* (i,1) のとき */
```

```
213             L = psi(i, 1);
214             B[L] -= beta_2 * alpha(x, c);
215             /* (i,N_y-1) のとき */
216             L = psi(i, N_y - 1);
217             B[L] -= beta_2 * alpha(x, d);
218         }
219
220         /* A vector_U = B を解く */
221         symbandsolve(A, B, NN, mm+1);
222         /* */
223         for (i = 1; i < N_x; i++)
224             for (j = 1; j < N_y; j++)
225                 Uk[i][j] = B[psi(i,j)];
226
227         /* 下の辺、上の辺にある格子点（角の点も含める）*/
228         for (i = 0; i <= N_x; i++) {
229             x = a + i * h_x;
230             Uk[i][0]   = alpha(x, c);
231             Uk[i][N_y] = alpha(x, d);
232         }
233         /* 左の辺、右の辺にある格子点（角の点は含めない）*/
234         for (j = 1; j < N_y; j++) {
235             y = c + j * h_y;
236             Uk[0][j]   = alpha(a, y);
237             Uk[N_x][j] = alpha(b, y);
238         }
239
240         /* データを数値で表示 */
241         if (n % skip == 0) {
242 #ifdef  PRINT
243             for (i = 0; i <= N_x; i++) {
244                 for (j = 0; j <= N_y; j++)
245                     printf(" %5.1f", Uk[i][j]);
246                 printf("\n");
247             }
248 #endif
249             /* 鳥瞰図を描く */
250             g_cls();
251             g_hidden2(1.0, 1.0, 0.4, -1.0, 1.0, 5.0, 25.0, 20.0, 20.0, 20.0,
252                       150.0, 100.0, Uk, N_x + 1, N_y + 1,
253                       1, G_SIDE_NONE, 2, 1);
254         }
255     }
256     /* マウスでクリックされるのを待つ */
257     g_sleep(-1.0);
258     /* ウィンドウを消す */
259     g_term();
260
261     return 0;
262 }
263
264 /* 初期値 */
265 double f(double x, double y)
266 {
267     /* ピラミッド型の関数 */
268     if (y > 0.5)
269         y = 1 - y;
270     if (x > 0.5)
271         x = 1 - x;
272     if (y < x)
```

```
273           return 5 * y;
274       else
275           return 5 * x;
276   }
277
278   /* 境界値 */
279   double alpha(double x, double y)
280   {
281       /* 同次 Dirichlet 境界条件 */
282       return 0.0;
283   }
```

このプログラム (と symbandlu.c など) も以下のページから入手できる。

https://m-katsurada.sakura.ne.jp/program/fdm/

3.3.10 MATLAB プログラム

MATLAB プログラムを作ってある。そのうち、この文書とマージするつもりであるが、現在は別文書にしてある。

桂田 [16] (2015), [17] (2024), [19] (2024) を見よ。それぞれ同次 Ditichlet 境界条件の場合、非同次 Ditichlet 境界条件の場合、非同次 Neumann 境界条件の場合を説明してある。

この問題は一応解決した。

3.4 Neumann 境界条件の場合

Dirichlet 境界条件の代りに Neumann 境界条件を課した初期値境界値問題を考えよう。

3.4.1 境界条件の設定

Neumann 境界値 $\frac{\partial u}{\partial n}$ は、長方形領域 $\Omega = (a,b) \times (c,d)$ の 4 つの角点 $((a,c), (a,d), (b,c), (b,d))$ においては普通の意味では定義されない。以下では、解 u が領域の閉包 $\overline{\Omega}$ で C^1 級 (すなわち、u が $\overline{\Omega}$ のある開近傍まで C^1 級に拡張できる) と仮定して、角点において $\frac{\partial u}{\partial x}$, $\frac{\partial u}{\partial y}$ が分かっている場合に近似解を求めることにする。

もともと $\frac{\partial u}{\partial x}(a,y)$ $(y \in (c,d))$, $\frac{\partial u}{\partial x}(b,y)$ $(y \in (c,d))$, $\frac{\partial u}{\partial y}(x,c)$ $(x \in (a,b))$, $\frac{\partial u}{\partial y}(x,d)$ $(x \in (a,b))$ は (角点以外の点における $\frac{\partial u}{\partial n}$ であるから) 与えられているので、その極限が分かるというのは不自然な仮定ではないであろう。

せっかくだから、境界条件が時間に依存するように一般化しよう。

$$-\frac{\partial u}{\partial x}(a,y,t), \frac{\partial u}{\partial x}(b,y,t) \quad (c \le y \le d,\ t \ge 0),$$

$$-\frac{\partial u}{\partial y}(x,c,t), \frac{\partial u}{\partial y}(x,d,t) \quad (a \le x \le b,\ t \ge 0)$$

を計算する関数 (それぞれ `b_l(x,y,t)`, `b_r(x,y,t)`, `b_b(x,y,t)`, `b_t(x,y,t)` と名付ける) が与えられているとする。

角点以外の $\partial\Omega$ の点 (x,y) と $t \geq 0$ に対して

$$\Phi(x,y,t) := \begin{cases} \mathrm{b\_l}(x,y,t) & (x=a,\ c<y<d) \\ \mathrm{b\_r}(x,y,t) & (x=b,\ c<y<d) \\ \mathrm{b\_b}(x,y,t) & (y=c,\ a<x<b) \\ \mathrm{b\_t}(x,y,t) & (y=d,\ a<x<b) \end{cases}$$

とおく。いわゆる Neumann 境界値である。

3.4.2 差分方程式の導出

さて、

$$\alpha = -\theta\lambda_x, \quad \beta = -\theta\lambda_y, \quad \gamma = 1 + 2\theta(\lambda_x + \lambda_y),$$
$$\alpha' = (1-\theta)\lambda_x, \quad \beta' = (1-\theta)\lambda_y, \quad \gamma' = 1 - 2(1-\theta)(\lambda_x + \lambda_y),$$

とおくと、熱方程式を差分近似して出来た差分方程式は、

$$\begin{aligned} &\gamma U_{i,j}^{n+1} + \alpha(U_{i+1,j}^{n+1} + U_{i-1,j}^{n+1}) + \beta(U_{i,j+1}^{n+1} + U_{i,j-1}^{n+1}) \\ &\quad = \gamma' U_{i,j}^{n} + \alpha'(U_{i+1,j}^{n} + U_{i-1,j}^{n}) + \beta'(U_{i,j+1}^{n} + U_{i,j-1}^{n}) \end{aligned} \tag{3.16}$$

表せる。この式に仮想格子点における値が現れる場合に、境界条件を用いてそれを消去することを考える。

(1) (長方形の左辺に注目している格子点がある場合, 角点は除く) $i=0,\ 1 \leq j \leq N_y - 1$ のとき

$$\frac{U_{1,j}^{n+1} - U_{-1,j}^{n+1}}{2h_x} = \frac{\partial u}{\partial x}(x_0, y_j, t_{n+1}) = -\Phi(x_0, y_j, t_{n+1}),$$
$$\frac{U_{1,j}^{n} - U_{-1,j}^{n}}{2h_x} = \frac{\partial u}{\partial x}(x_0, y_j, t_{n}) = -\Phi(x_0, y_j, t_{n})$$

より

$$U_{-1,j}^{n+1} = U_{1,j}^{n+1} + 2h_x\Phi_{0,j}^{n+1}, \quad U_{-1,j}^{n} = U_{1,j}^{n} + 2h_x\Phi_{0,j}^{n}.$$

これらを差分方程式 (3.16) に代入して

$$\begin{aligned} &\gamma U_{0,j}^{n+1} + \alpha(U_{1,j}^{n+1} + U_{1,j}^{n+1} + 2h_x\Phi_{0,j}^{n+1}) + \beta(U_{0,j+1}^{n+1} + U_{0,j-1}^{n+1}) \\ &\quad = \gamma' U_{0,j}^{n} + \alpha'(U_{1,j}^{n} + U_{1,j}^{n} + 2h_x\Phi_{0,j}^{n}) + \beta'(U_{0,j+1}^{n} + U_{0,j-1}^{n}) \end{aligned}$$

整理すると

$$\begin{aligned} &\gamma U_{0,j}^{n+1} + 2\alpha U_{1,j}^{n+1} + \beta(U_{0,j+1}^{n+1} + U_{0,j-1}^{n+1}) \\ &\quad = \gamma' U_{0,j}^{n} + 2\alpha' U_{1,j}^{n} + \beta'(U_{0,j+1}^{n} + U_{0,j-1}^{n}) + 2h_x\left[\alpha'\Phi_{0,j}^{n} - \alpha\Phi_{0,j}^{n+1}\right]. \end{aligned}$$

以下のプログラムでは、係数行列を対称にするために、この方程式の両辺に $\frac{1}{2}$ をかけた次の方程式を用いている。

$$\begin{aligned} &\frac{\gamma}{2} U_{0,j}^{n+1} + \alpha U_{1,j}^{n+1} + \frac{\beta}{2}(U_{0,j+1}^{n+1} + U_{0,j-1}^{n+1}) \\ &\quad = \frac{\gamma'}{2} U_{0,j}^{n} + \alpha' U_{1,j}^{n} + \frac{\beta'}{2}(U_{0,j+1}^{n} + U_{0,j-1}^{n}) + h_x\left[\alpha'\Phi_{0,j}^{n} - \alpha\Phi_{0,j}^{n+1}\right]. \end{aligned}$$

(2) (長方形の右辺に注目している格子点がある場合, 角点は除く) $i = N_x$, $1 \le j \le N_y - 1$ のとき。

$$\frac{U^{n+1}_{N_x+1,j} - U^{n+1}_{N_x-1,j}}{2h_x} = \frac{\partial u}{\partial x}(x_{N_x}, y_j, t_{n+1}) = \Phi^{n+1}_{N_x,j},$$
$$\frac{U^{n}_{N_x+1,j} - U^{n}_{N_x-1,j}}{2h_x} = \frac{\partial u}{\partial x}(x_{N_x}, y_j, t_{n+1}) = \Phi^{n}_{N_x,j}$$

より

$$U^{n+1}_{N_x+1,j} = U^{n+1}_{N_x-1,j} + 2h_x\Phi^{n+1}_{N_x,j}, \quad U^{n}_{N_x+1,j} = U^{n}_{N_x-1,j} + 2h_x\Phi^{n}_{N_x,j}.$$

これらを差分方程式 (3.16) に代入して

$$\gamma U^{n+1}_{N_x,j} + \alpha(U^{n+1}_{N_x-1,j} + U^{n+1}_{N_x-1,j} + 2h_x\Phi^{n+1}_{N_x,j}) + \beta(U^{n+1}_{N_x,j+1} + U^{n+1}_{N_x,j-1})$$
$$= \gamma' U^{n}_{N_x,j} + \alpha'(U^{n}_{N_x-1,j} + U^{n}_{N_x-1,j} + 2h_x\Phi^{n}_{N_x,j}) + \beta'(U^{n}_{N_x,j+1} + U^{n}_{N_x,j-1})$$

整理すると

$$\gamma U^{n+1}_{N_x,j} + 2\alpha U^{n+1}_{N_x-1,j} + \beta(U^{n+1}_{N_x,j+1} + U^{n+1}_{N_x,j-1})$$
$$= \gamma' U^{n}_{N_x,j} + 2\alpha' U^{n}_{N_x-1,j} + \beta'(U^{n}_{N_x,j+1} + U^{n}_{N_x,j-1}) + 2h_x\left[\alpha'\Phi^{n}_{N_x,j} - \alpha\Phi^{n+1}_{N_x,j}\right].$$

両辺に $\frac{1}{2}$ をかけて

$$\frac{\gamma}{2} U^{n+1}_{N_x,j} + \alpha U^{n+1}_{N_x-1,j} + \frac{\beta}{2}(U^{n+1}_{N_x,j+1} + U^{n+1}_{N_x,j-1})$$
$$= \frac{\gamma'}{2} U^{n}_{N_x,j} + \alpha' U^{n}_{N_x-1,j} + \frac{\beta'}{2}(U^{n}_{N_x,j+1} + U^{n}_{N_x,j-1}) + h_x\left[\alpha'\Phi^{n}_{N_x,j} - \alpha\Phi^{n+1}_{N_x,j}\right].$$

(3) (長方形の下辺に注目している格子点がある場合, 角点は除く) $j = 0$, $1 \le i \le N_x - 1$ のとき。

$$\frac{U^{n+1}_{i,1} - U^{n+1}_{i,-1}}{2h_y} = \frac{\partial u}{\partial y}(x_i, y_0, t_{n+1}) = -\Phi^{n+1}_{i,0},$$
$$\frac{U^{n}_{i,1} - U^{n}_{i,-1}}{2h_y} = \frac{\partial u}{\partial y}(x_i, y_0, t_{n}) = -\Phi^{n}_{i,0}$$

より

$$U^{n+1}_{i,-1} = U^{n+1}_{i,1} + 2h_y\Phi^{n+1}_{i,0}, \quad U^{n}_{i,-1} = U^{n}_{i,1} + 2h_y\Phi^{n}_{i,0}.$$

これらを差分方程式 (3.16) に代入して

$$\gamma U^{n+1}_{i,0} + \alpha(U^{n+1}_{i+1,0} + U^{n+1}_{i-1,0}) + \beta(U^{n+1}_{i,1} + U^{n+1}_{i,1} + 2h_y\Phi^{n+1}_{i,0})$$
$$= \gamma' U^{n}_{i,0} + \alpha'(U^{n}_{i+1,0} + U^{n}_{i-1,0}) + \beta'(U^{n}_{i,1} + U^{n}_{i,1} + 2h_y\Phi^{n}_{i,0})$$

整理すると

$$\gamma U^{n+1}_{i,0} + \alpha(U^{n+1}_{i+1,0} + U^{n+1}_{i-1,0}) + 2\beta U^{n+1}_{i,1}$$
$$= \gamma' U^{n}_{i,0} + \alpha'(U^{n}_{i+1,0} + U^{n}_{i-1,0}) + 2\beta' U^{n}_{i,1} + 2h_y(\beta'\Phi^{n}_{i,0} - \beta\Phi^{n+1}_{i,0}).$$

両辺に $\frac{1}{2}$ をかけて

$$\frac{\gamma}{2} U^{n+1}_{i,0} + \frac{\alpha}{2}(U^{n+1}_{i+1,0} + U^{n+1}_{i-1,0}) + \beta U^{n+1}_{i,1}$$
$$= \frac{\gamma'}{2} U^{n}_{i,0} + \frac{\alpha'}{2}(U^{n}_{i+1,0} + U^{n}_{i-1,0}) + \beta' U^{n}_{i,1} + h_y(\beta'\Phi^{n}_{i,0} - \beta\Phi^{n+1}_{i,0}).$$

(4) (長方形の上辺に注目している格子点がある場合, 角点は除く) $j = N_y$, $1 \le i \le N_x - 1$ のとき。

$$\frac{U^{n+1}_{i,N_y+1} - U^{n+1}_{i,N_y-1}}{2h_y} = \frac{\partial u}{\partial y}(x_i, y_{N_y}, t_{n+1}) = \Phi^{n+1}_{i,N_y}, \quad \frac{U^{n}_{i,N_y+1} - U^{n}_{i,N_y-1}}{2h_y} = \Phi^{n}_{i,N_y}$$

より

$$U^{n+1}_{i,N_y+1} = U^{n+1}_{i,N_y-1} + 2h_y\Phi^{n+1}_{i,N_y}, \quad U^{n}_{i,N_y+1} = U^{n}_{i,N_y-1} + 2h_y\Phi^{n}_{i,N_y}.$$

これらを差分方程式 (3.16) に代入して

$$\begin{aligned}&\gamma U^{n+1}_{i,N_y} + \alpha(U^{n+1}_{i+1,N_y} + U^{n+1}_{i-1,N_y}) + \beta(U^{n+1}_{i,N_y-1} + U^{n+1}_{i,N_y-1} + 2h_y\Phi^{n+1}_{i,N_y})\\ &\quad = \gamma' U^{n}_{i,N_y} + \alpha'(U^{n}_{i+1,N_y} + U^{n}_{i-1,N_y}) + \beta'(U^{n}_{i,N_y-1} + U^{n}_{i,N_y-1} + 2h_y\Phi^{n}_{i,N_y})\end{aligned}$$

整理すると

$$\begin{aligned}&\gamma U^{n+1}_{i,N_y} + \alpha(U^{n+1}_{i+1,N_y} + U^{n+1}_{i-1,N_y}) + 2\beta U^{n+1}_{i,N_y-1}\\ &\quad = \gamma' U^{n}_{i,N_y} + \alpha'(U^{n}_{i+1,N_y} + U^{n}_{i-1,N_y}) + 2\beta' U^{n}_{i,N_y-1} + 2h_y[\beta'\Phi^{n}_{i,N_y} - \beta\Phi^{n+1}_{i,N_y}].\end{aligned}$$

両辺に $\frac{1}{2}$ をかけて

$$\begin{aligned}&\frac{\gamma}{2} U^{n+1}_{i,N_y} + \frac{\alpha}{2}(U^{n+1}_{i+1,N_y} + U^{n+1}_{i-1,N_y}) + \beta U^{n+1}_{i,N_y-1}\\ &\quad = \frac{\gamma'}{2} U^{n}_{i,N_y} + \frac{\alpha'}{2}(U^{n}_{i+1,N_y} + U^{n}_{i-1,N_y}) + \beta' U^{n}_{i,N_y-1} + h_y[\beta'\Phi^{n}_{i,N_y} - \beta\Phi^{n+1}_{i,N_y}].\end{aligned}$$

仮想格子点での値をどう表すか

自分自身が混乱して、符号を間違えたこともあったので、補足的な注意を述べておく。外向き単位法線方向の微分の意味を思い出せば

$$U_{\text{仮想格子点}} = U_{\text{内側の格子点}} + 2h\frac{\partial u}{\partial n}(\text{境界上の格子点})$$

が成立することが納得できるはず。

以下、角の点での差分方程式を検討する。

左下の角点

$$\frac{\partial u}{\partial x}(x_0, y_0, t_n) \fallingdotseq \frac{u^n_{1,0} - u^n_{-1,0}}{2h_x} \quad \text{より} \quad U^n_{-1,0} = U^n_{1,0} + 2h_x \texttt{b\_l}(x_0, y_0, t_n),$$

$$\frac{\partial u}{\partial y}(x_0, y_0, t_n) \fallingdotseq \frac{u^n_{0,1} - u^n_{0,-1}}{2h_y} \quad \text{より} \quad U^n_{0,-1} = U^n_{0,1} + 2h_y \texttt{b\_b}(x_0, y_0, t_n)$$

のような近似がしたい。

差分方程式

$$\gamma U^{n+1}_{i,j} + \alpha(U^{n+1}_{i+1,j} + U^{n+1}_{i-1,j}) + \beta(U^{n+1}_{i,j+1} + U^{n+1}_{i,j-1}) = \gamma' U^{n}_{i,j} + \alpha'(U^{n}_{i+1,j} + U^{n}_{i-1,j}) + \beta'(U^{n}_{i,j+1} + U^{n}_{i,j-1})$$

を、$i=0,\ j=0$ では次のように変形する。

$$\gamma U^{n+1}_{x_0,y_0} + \alpha(U^{n+1}_{1,0} + U^{n+1}_{1,0} + 2h_x\mathtt{b\_l}(x_0,y_0,t_{n+1})) + \beta(U^{n+1}_{0,1} + U^{n+1}_{0,1} + 2h_y\mathtt{b\_b}(x_0,y_0,t_{n+1}))$$
$$= \gamma' U^{n}_{x_0,y_0} + \alpha'(U^{n}_{1,0} + U^{n}_{1,0} + 2h_x\mathtt{b\_l}(x_0,y_0,t_n)) + \beta'(U^{n}_{0,1} + U^{n}_{0,1} + 2h_y\mathtt{b\_b}(x_0,y_0,t_n))$$

$$\gamma U^{n+1}_{x_0,y_0} + 2\alpha U^{n+1}_{1,0} + 2h_x\alpha\mathtt{b\_l}(x_0,y_0,t_{n+1}) + 2\beta U^{n+1}_{0,1} + 2h_y\beta\mathtt{b\_b}(x_0,y_0,t_{n+1})$$
$$= \gamma' U^{n}_{x_0,y_0} + 2\alpha' U^{n}_{1,0} + 2h_x\alpha'\mathtt{b\_l}(x_0,y_0,t_n) + 2\beta' U^{n}_{0,1} + 2h_y\beta'\mathtt{b\_b}(x_0,y_0,t_n)$$

$$\gamma U^{n+1}_{x_0,y_0} + 2\alpha U^{n+1}_{1,0} + 2\beta U^{n+1}_{0,1} = \gamma' U^{n}_{x_0,y_0} + 2\alpha' U^{n}_{1,0} + 2\beta' U^{n}_{0,1}$$
$$+ 2h_x(\alpha'\mathtt{b\_l}(x_0,y_0,t_n) - \alpha\mathtt{b\_l}(x_0,y_0,t_{n+1})) + 2h_y(\beta'\mathtt{b\_b}(x_0,y_0,t_n) - \beta\mathtt{b\_b}(x_0,y_0,t_{n+1}))$$

```
L = phi(0,0); x = a; y = c;
B[L] = gamma_p * Uk[0][0]
     + 2 * alpha_p * Uk[1][0] + 2 * beta_p * Uk[0][1]
     + 2h_x * (alpha_p * b_l(x,y,tn) - alpha * b_l(x,y,tnp1))
     + 2h_y * (beta_p * b_b(x,y,tn) - beta * b_b(x,y,tnp1));
```

左上の角点

$$\frac{\partial u}{\partial x}(x_0,y_{N_y},t_n) \fallingdotseq \frac{u^n_{1,N_y} - u^n_{-1,N_y}}{2h_x} \quad \text{より} \quad U^n_{-1,N_y} = U^n_{1,N_y} + 2h_x\mathtt{b\_l}(x_0,y_{N_y},t_n),$$

$$\frac{\partial u}{\partial y}(x_0,y_{N_y},t_n) \fallingdotseq \frac{u^n_{0,N_y+1} - u^n_{0,N_y-1}}{2h_y} \quad \text{より} \quad U^n_{0,N_y+1} = U^n_{0,N_y-1} + 2h_y\mathtt{b\_t}(x_0,y_{N_y},t_n)$$

差分方程式を $i=0,\ j=N_y$ では次のように変形する。

$$\gamma U^{n+1}_{0,N_y} + \alpha(U^{n+1}_{1,N_y} + U^{n+1}_{1,N_y} + 2h_x\mathtt{b\_l}(x_0,y_{N_y},t_{n+1})) + \beta(U^{n+1}_{0,N_y-1} + U^{n+1}_{0,N_y-1} + 2h_y\mathtt{b\_t}(x_0,y_{N_y},t_{n+1}))$$
$$= \gamma' U^{n}_{0,N_y} + \alpha'(U^{n}_{1,N_y} + U^{n}_{1,N_y} + 2h_x\mathtt{b\_l}(x_0,y_{N_y},t_n)) + \beta'(U^{n}_{0,N_y-1} + U^{n}_{0,N_y-1} + 2h_y\mathtt{b\_t}(x_0,y_{N_y},t_n))$$

$$\gamma U^{n+1}_{0,N_y} + 2\alpha U^{n+1}_{1,N_y} + 2\beta U^{n+1}_{0,N_y-1} = \gamma' U^{n}_{0,N_y} + 2\alpha' U^{n}_{1,N_y} + 2\beta' U^{n}_{0,N_y-1}$$
$$+ 2h_x(\alpha'\mathtt{b\_l}(x_0,y_{N_y},t_n) - \alpha\mathtt{b\_l}(x_0,y_{N_y},t_{n+1})) + 2h_y(\beta'\mathtt{b\_t}(x_0,y_{N_y},t_n) - \beta\mathtt{b\_t}(x_0,y_{N_y},t_{n+1}))$$

```
L = phi(0,N_y); x = a; y = d;
B[L] = gamma_p * Uk[0][N_y]
     + 2 * alpha_p * Uk[1][N_y] + 2 * beta_p * Uk[0][N_y - 1]
     + 2 * h_x * (alpha_p * b_l(x,y,tn) - alpha * b_l(x,y,tnp1))
     + 2 * h_y * (beta_p * b_t(x,y,tn) - beta * b_t(x,y,tnp1));
```

右下の角点

```
L = phi(N_x,0); x = b; y = c;
B[L] = gamma_p * Uk[N_x][0]
     + 2 * alpha_p * Uk[N_x - 1][0] + 2 * beta_p * Uk[N_x][1]
     + 2 * h_x * (alpha_p * b_r(x,y,tn) - alpha * b_r(x,y,tnp1))
     + 2 * h_y * (beta * b_b(x,y,tn) - beta_p * b_b(x,y,tnp1));
```

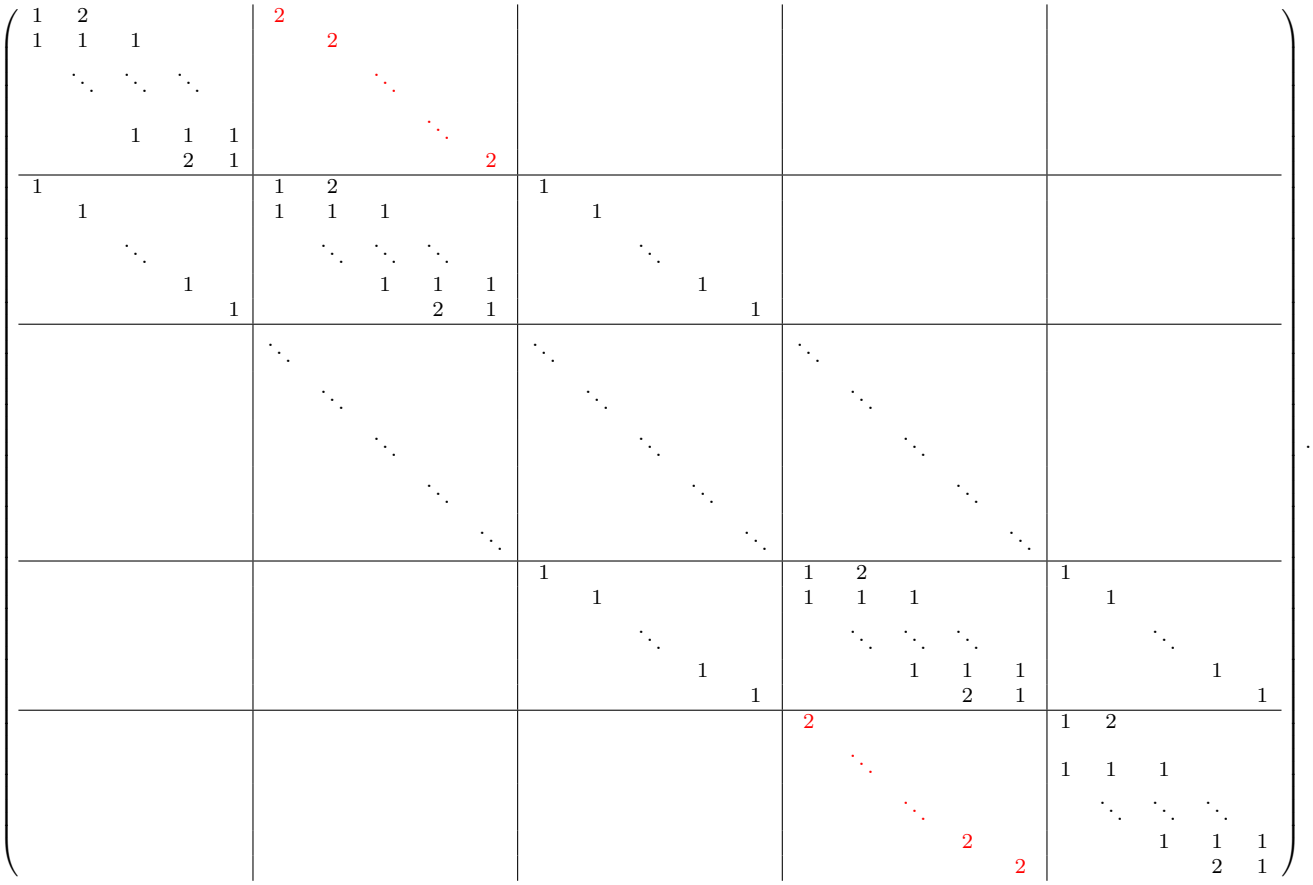
右上の角点

```
L = phi(N_x,N_y); x = b; y = d;
B[L] = gamma_p * Uk[N_x][N_y]
     + 2 * alpha_p * Uk[N_x - 1][N_y] + 2 * beta_p * Uk[N_x][N_y - 1]
     + 2 * h_x * (alpha_p * b_r(x,y,tn) - alpha * b_r(x,y,tnp1))
     + 2 * h_y * (beta_p * b_t(x,y,tn) - beta * b_t(x,y,tnp1));
```

以下に紹介するプログラムでは、係数行列が対称であるように連立 1 次方程式を修正している[24]。

3.4.3 係数行列の対称化の操作について (いくつかの行に $1/2$, $1/4$ をかける理由)

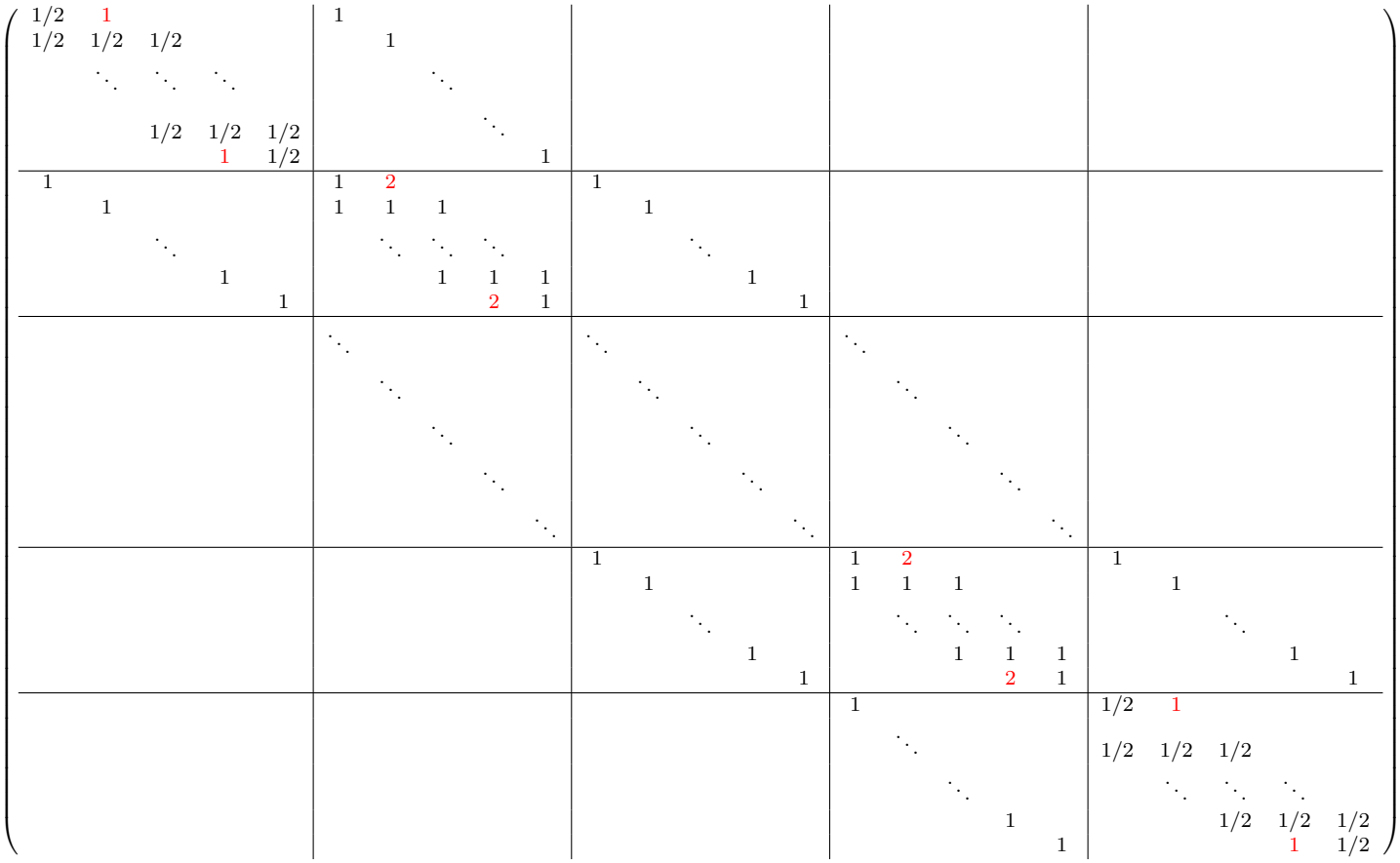
差分方程式 (3.16) において、仮想格子点での値を内側の格子点での値を用いて表すことで消去すると、係数が変化する。その方程式を行列表示した場合に、係数行列の各成分が消去前と比較して何倍になったかを表示すると、次のようになる。



[24] どのように修正すればよいかは、1998 年度卒研学生の深石君が卒業レポート課題中で解決してくれた。

対称行列に近いが、対角線に関して対称の位置にある成分と等しくないもの (2 倍になっているもの) が少数存在している。

まず、赤い部分の 2 を 1 にするために、最初の N_x+1 行と、最後の N_x+1 行を 1/2 倍すると (方程式に 1/2 をかける操作に相当する — 右辺にも 1/2 をかければ、同値変形となる)、次のようになる。



この行列の対角ブロック以外の部分は対称である。対角ブロックには 2 種類ある。最初と最後の対角ブロックは

$$\begin{pmatrix} 1/2 & 1 & & & \\ 1/2 & 1/2 & 1/2 & & \\ & \ddots & \ddots & \ddots & \\ & & 1/2 & 1/2 & 1/2 \\ & & & 1 & 1/2 \end{pmatrix},$$

それ以外の N_y-1 個の対角ブロックは

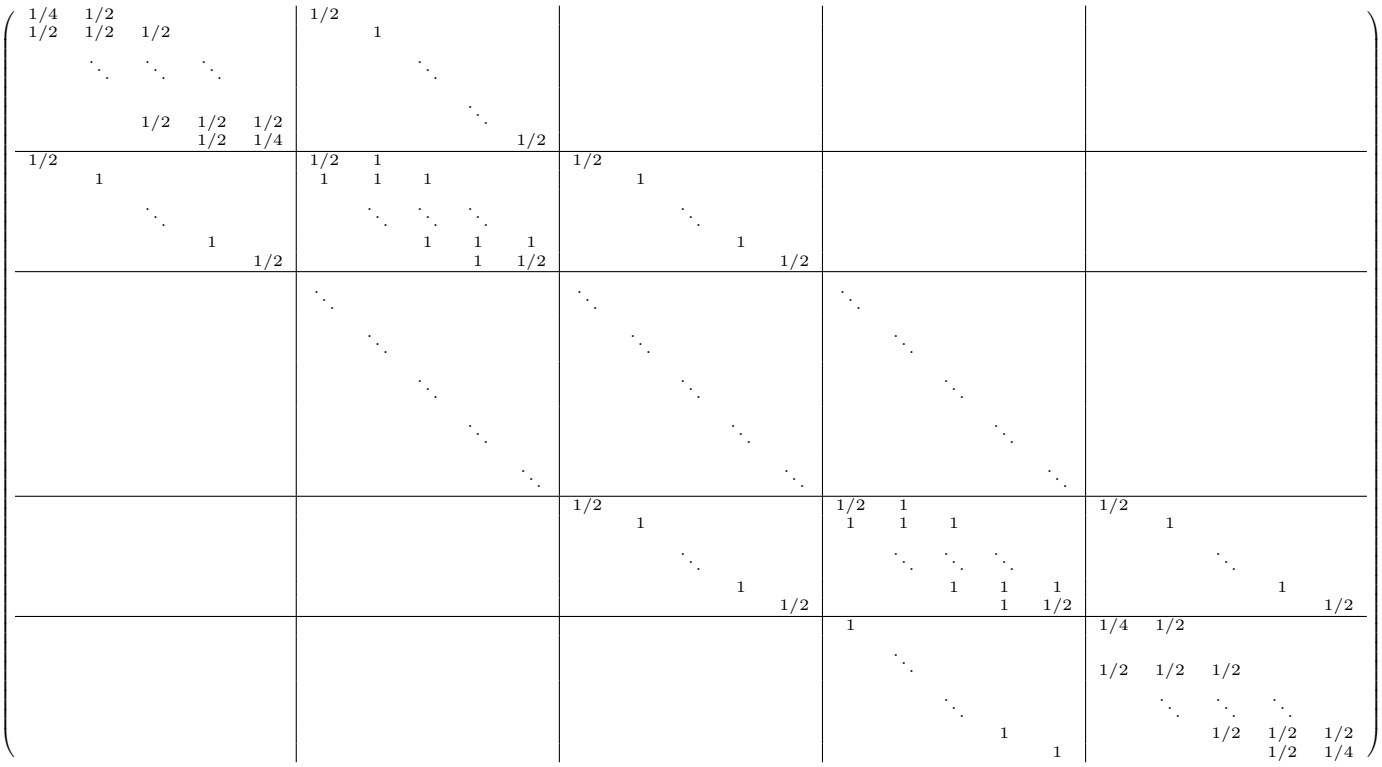
$$\begin{pmatrix} 1 & 2 & & & \\ 1 & 1 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 1 & 1 \\ & & & 2 & 1 \end{pmatrix}$$

である。これらすべてのブロックに対して、最初の行と最後の行を 1/2 倍すると、それぞれ

$$\begin{pmatrix} 1/4 & 1/2 & & & & \\ 1/2 & 1/2 & 1/2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & 1/2 & 1/2 & 1/2 & \\ & & & 1/2 & 1/4 \end{pmatrix}, \qquad \begin{pmatrix} 1/2 & 1 & & & & \\ 1 & 1 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & 1 & 1 & 1 & \\ & & & 1 & 1/2 \end{pmatrix}$$

となり、これらは対称である。

この、各対角ブロックの最初の行と最後の行を 1/2 倍することで、対角ブロック以外のブロックも変更されるが、その結果は



であり、対称行列である。

3.4.4 heat2n-i-naive2.c

以上で説明したことを素朴にコーディングすると、次のようなプログラムが得られる。連立 1 次方程式の係数行列は帯行列であるが、そのことは利用していないので、効率は高くない。帯行列であることを利用したプログラムは次項で与える。

```
 1  /*
 2   * heat2n-i-naive2.c --- 2 次元熱方程式 (Neumann 境界条件) を陰解法で素朴に解く
 3   *
 4   * http://nalab.mind.meiji.ac.jp/~mk/program/fdm/heat2n-i-naive2.c
 5   * http://nalab.mind.meiji.ac.jp/~mk/program/fdm/smallmatrix.h
 6   * http://nalab.mind.meiji.ac.jp/~mk/program/linear/lu.c
 7   * http://nalab.mind.meiji.ac.jp/~mk/program/linear/lu.h
 8   *
 9   * To compile
10   *     cglsc heat2n-i-naive.c lu.c
```

```
11   *
12   * このプログラムについては 1998 年度卒研の学生だった深石君に感謝します。
13   * このプログラムは効率的ではないので、N_x, N_y はあまり大きくしないこと。
14   * 50 程度にしておくことを勧めます。
15   */
16
17  #include <stdio.h>
18  #include <stdlib.h>
19  #include <math.h>
20  #ifdef OLD
21  #include <matrix.h>
22  #else
23  #include "smallmatrix.h"
24  #endif
25  #ifndef G_DOUBLE
26  #define G_DOUBLE
27  #endif
28  #include <glsc.h>
29  #include "lu.h"
30
31  // 格子点に番号をつける。
32  // いわゆる column major mode である。
33  // m = N_x + 1
34  // 行列の最初の m 行, 最後の m 行は j=0,N_y（長方形領域の下辺、上辺）に対応する。
35  #define phi(i,j) (j)*m+(i)
36
37  // Neumann 境界値
38  double b_t(double, double, double);
39  double b_b(double, double, double);
40  double b_l(double, double, double);
41  double b_r(double, double, double);
42
43  int main(void)
44  {
45      double a, b, c, d;
46      int N_x, N_y, m, N, i, j, p, q, L, n, nMax;
47      matrix Uk, A;
48      double *B, *vector_U, cond;
49      int *iwork, skip;
50      double h_x, h_y, lambda_x, lambda_y, lambda, lambda_limit, tau, Tmax, dt;
51      double f(double, double), Phi(double, double, double);
52      double theta, gamma, alpha, beta, gamma_p, alpha_p, beta_p;
53      double x, y, tnp1, tn;
54
55      /* 問題を考える区間 [a,b] × [c,d] */
56      a = 0.0; b = 1.0; c = 0.0; d = 1.0;
57
58      /* 区間の分割数 */
59      printf("Nx, Ny: "); scanf("%d %d", &N_x, &N_y);
60
61      m = N_x + 1;
62      N = (N_x + 1) * (N_y + 1);
63      /* 空間の刻み幅 */
64      h_x = (b - a) / N_x;
65      h_y = (d - c) / N_y;
66
67      /* 行列、ベクトルを記憶する変数のメモリー割り当て */
68      if ((Uk = new_matrix(N_x + 1, N_y + 1)) == NULL) {
69          fprintf(stderr, "数列 U^k を記憶する領域の確保に失敗\n");
70          exit(1);
```

```
71      }
72      if ((A = new_matrix(N, N)) == NULL) {
73          fprintf(stderr, "係数行列 A を記憶する領域の確保に失敗\n");
74          exit(1);
75      }
76      if ((B = malloc(sizeof(double) * N)) == NULL) {
77          fprintf(stderr, "B を記憶する領域の確保に失敗\n");
78          exit(1);
79      }
80      if ((vector_U = malloc(sizeof(double) * N)) == NULL) {
81          fprintf(stderr, "vector_U を記憶する領域の確保に失敗\n");
82          exit(1);
83      }
84      if ((iwork = malloc(sizeof(int) * N)) == NULL) {
85          fprintf(stderr, "iwork を記憶する領域の確保に失敗\n");
86          exit(1);
87      }
88
89      /* θ法の重みの決定 */
90      printf("θ (0≦θ≦1): "); scanf("%lf", &theta);
91
92      if (theta == 1.0) {
93          printf("τ: "); scanf("%lf", &tau);
94      } else {
95          printf("τ (≦%g ≡最大値ノルムに関する安定性条件を満たすτの上限): ",
96                 0.5 / (1 - theta) / (1 / (h_x * h_x) + 1 / (h_y * h_y)));
97          scanf("%lf", &tau);
98      }
99
100     lambda_x = tau / (h_x * h_x);
101     lambda_y = tau / (h_y * h_y);
102     lambda = lambda_x + lambda_y;
103
104     /* 最大値ノルムに関する安定性を満たすλの上限 */
105     lambda_limit = 1.0 / (2.0 * (1.0 - theta));
106
107     if (lambda > lambda_limit)
108         printf("注意: λ=%g>1/2(1-θ) となっています。\n", lambda);
109     else
110         printf("λ=%g\n", lambda);
111
112     /* 初期値の設定 */
113     for (i = 0; i <= N_x; i++) {
114       x = a + i * h_x;
115       for (j = 0; j <= N_y; j++)
116         Uk[i][j] = f(x, c + j * h_y);
117     }
118
119     /* 連立1次方程式に現れる係数
120        γ U_{ij}^{n+1}
121          +α (U_{i+1,j}^{n+1}+U_{i-1,j}^{n+1})
122          +β (U_{i,j+1}^{n+1}+U_{i,j-1}^{n+1})
123       =γ'U_{ij}^n
124          +α'(U_{i+1,j}^n+U_{i-1,j}^n)
125          +β'(U_{i,j+1}^n+U_{i,j-1}^n)
126      と書いたときのα, β, γ, α', β', γ' */
127     gamma = 1.0 + 2.0 * theta * lambda;
128     alpha = - theta * lambda_x;
129     beta  = - theta * lambda_y;
130
```

```
131      gamma_p = 1.0 - 2.0 * (1.0 - theta) * lambda;
132      alpha_p = (1.0 - theta) * lambda_x;
133      beta_p  = (1.0 - theta) * lambda_y;
134
135      /* 係数行列の作成 */
136      /* まず 0 クリア */
137      for (p = 0; p < N; p++) {
138          for (q = 0; q < N; q++)
139              A[p][q] = 0.0;
140      }
141      for (i = 0; i <= N_x; i++)
142          for (j = 0; j <= N_y; j++) {
143              L = phi(i, j);
144              if (j != 0)   A[L][L - m] = beta;
145              if (i != 0)   A[L][L - 1] = alpha;
146                            A[L][L    ] = gamma;
147              if (i != N_x) A[L][L + 1] = alpha;
148              if (j != N_y) A[L][L + m] = beta;
149          }
150
151      /* 左辺または右辺（角点を除く）*/
152      for (j = 1; j < N_y; j++) {
153          /* 左辺にある (0,j) */
154          L = phi(0,j);
155  #ifdef ORIGINAL
156          // 右の格子点の係数を 2 倍
157          A[L][L + 1] *= 2.0;
158  #else
159          // その格子点と、上と下の格子点の係数を 2 で割る。
160          // 左の格子点 (仮想格子点) の係数は行列にない。
161          A[L][L - m] /= 2.0; A[L][L] /= 2.0; A[L][L + m] /= 2.0;
162  #endif
163          /* 右辺にある (N_x,j) */
164          L = phi(N_x,j);
165  #ifdef ORIGINAL
166          // 左の格子点の係数を 2 倍
167          A[L][L - 1] *= 2.0;
168  #else
169          // その格子点と、上と下の格子点の係数を 2 で割る。
170          // 右の格子点 (仮想格子点) の係数は行列にない。
171          A[L][L - m] /= 2.0; A[L][L] /= 2.0; A[L][L + m] /= 2.0;
172  #endif
173      }
174
175      /* 下辺または上辺（角点を除く）*/
176      for (i = 1; i < N_x; i++) {
177          /* 下辺にある (i,0) */
178          L = phi(i,0);
179  #ifdef ORIGINAL
180          // 上の格子点の係数を 2 倍
181          A[L][L + m] *= 2.0;
182  #else
183          // その格子点と、左と右の格子点の係数を 2 で割る。
184          // 下の格子点 (仮想格子点) の係数は行列にない。
185          A[L][L - 1] /= 2.0; A[L][L] /= 2.0; A[L][L + 1] /= 2.0;
186  #endif
187          /* 上辺にある (i,N_y) */
188          L = phi(i,N_y);
189  #ifdef ORIGINAL
190          // 下の格子点の係数を 2 倍
```

```
191         A[L][L - m] *= 2.0;
192 #else
193         // その格子点と、左と右の格子点の係数を 2 で割る。
194         // 上の格子点 (仮想格子点) の係数は行列にない。
195         A[L][L - 1] /= 2.0; A[L][L] /= 2.0; A[L][L + 1] /= 2.0;
196 #endif
197     }
198     /* 角の点 */
199 #ifdef ORIGINAL
200     /* 左下の角点 */
201     L = phi(0,0);
202     A[L][L+1] *= 2.0; A[L][L+m] *= 2.0;
203     /* 左上の角点 */
204     L = phi(0,N_y);
205     A[L][L+1] *= 2.0; A[L][L-m] *= 2.0;
206     /* 右下の角点 */
207     L = phi(N_x,0);
208     A[L][L-1] *= 2.0; A[L][L+m] *= 2.0;
209     /* 右上の角点 */
210     L = phi(N_x,N_y);
211     A[L][L-1] *= 2.0; A[L][L-m] *= 2.0;
212 #else
213     /* 左下の角点 -- 右と上の格子点の係数を 2 で、その格子点の係数を 4 で割る */
214     L = phi(0,0);
215     A[L][L] /= 4.0; A[L][L+1] /= 2.0; A[L][L+m] /= 2.0;
216     /* 左上の角点 -- 右と下の格子点の係数を 2 で、その格子点の係数を 4 で割る */
217     L = phi(0,N_y);
218     A[L][L] /= 4.0; A[L][L+1] /= 2.0; A[L][L-m] /= 2.0;
219     /* 右下の角点 -- 左と上の格子点の係数を 2 で、その格子点の係数を 4 で割る */
220     L = phi(N_x,0);
221     A[L][L] /= 4.0; A[L][L-1] /= 2.0; A[L][L+m] /= 2.0;
222     /* 右上の角点 -- 左と下の格子点の係数を 2 で、その格子点の係数を 4 で割る */
223     L = phi(N_x,N_y);
224     A[L][L] /= 4.0; A[L][L-1] /= 2.0; A[L][L-m] /= 2.0;
225 #endif
226
227     /* 連立 1 次方程式の係数行列を表示する */
228     if (N < 20) {
229       printf("素朴に作った連立 1 次方程式の行列\n");
230       for (p = 0; p < N; p++) {
231         for (q = 0; q < N; q++)
232           printf("%7.3g", A[p][q]);
233         printf("\n");
234       }
235     }
236
237 #ifndef ORIGINAL
238     /* 対称性のチェック */
239     for (p = 0; p < N; p++)
240       for (q = 0; q <= p; q++)
241         if (A[p][q] != A[q][p])
242           printf("A[%d][%d]=%g, A[%d][%d]=%g\n", p, q, A[p][q], q, p, A[q][p]);
243 #endif
244
245     printf("備考: 1+2 θ λ=%5.2f, -θ λ x=%5.2f, -θ λ y=%5.2f\n",
246            gamma, alpha, beta);
247
248     printf("Tmax: "); scanf("%lf", &Tmax);
249     printf("Δ t: ");  scanf("%lf", &dt);
250     skip = rint(dt / tau);
```

```
251     if (skip == 0) skip = 1;
252
253     nMax = rint(Tmax / tau);
254
255     /* グラフィックス・ライブラリィ GLSC の呼び出し */
256     g_init("Meta", 250.0, 160.0);
257     g_device(G_BOTH);
258     g_def_scale(0,
259                 0.0, 1.0, 0.0, 1.0,
260                 30.0, 70.0, 100.0, 72.0);
261     g_def_scale(4,
262                 -1.0, 1.0, -1.0, 1.0,
263                 30.0, 30.0, 100.0, 100.0);
264     g_def_line(0, G_BLACK, 0, G_LINE_SOLID);
265     g_sel_scale(0);
266
267     g_cls();
268 #ifdef OLD
269     g_hidden2(1.0, 1.0, 0.4, -1.0, 1.0, 5.0, 25.0, 20.0, 20.0, 20.0,
270               150.0, 100.0, Uk, N_x + 1, N_y + 1,
271               1, G_SIDE_NONE, 2, 1);
272 #else
273     g_hidden(1.0, 1.0, 0.4, -1.0, 1.0, 5.0, 25.0, 20.0, 20.0, 20.0,
274              150.0, 100.0, &Uk[0][0], N_x + 1, N_y + 1,
275              1, G_SIDE_NONE, 2, 1);
276 #endif
277
278     /* 係数行列 LU 分解 */
279     decomp(N, A, &cond, iwork, B);
280     if (cond + 1 == cond) {
281         /* 条件数が大きければ、計算をあきらめる */
282         printf("MATRIX IS SINGULAR TO WORKING PRECISION\n");
283         return 0;
284     }
285
286     /* 時間に関するループ */
287     for (n = 0; n < nMax; n++) {
288         tn = n * tau; tnp1 = (n + 1) * tau;
289
290         /* まず、素朴な連立 1 次方程式の右辺を用意する */
291         /* 内部の格子点 */
292         for (i = 1; i < N_x; i++)
293             for (j = 1; j < N_y; j++) {
294                 L = phi(i,j);
295                 B[L] = gamma_p * Uk[i][j]
296                     + alpha_p * (Uk[i + 1][j] + Uk[i - 1][j])
297                     + beta_p * (Uk[i][j + 1] + Uk[i][j - 1]);
298             }
299         /* 以下、境界にある格子点での方程式を立てる。
300          * 仮想格子点での値は境界条件を中心差分近似した方程式を
301          * 用いて消去する
302          *   ……右辺にも移項する量があるので、右辺について後で処理する */
303         /* 下の辺、上の辺にある格子点（角の点は含めない） */
304         for (i = 1; i < N_x; i++) {
305             x = a + i * h_x;
306             /* (i, 0) */
307             L = phi(i,0); y = c;
308             B[L] = gamma_p * Uk[i][0]
309                  + alpha_p * (Uk[i + 1][0] + Uk[i - 1][0])
310                  + 2 * beta_p * Uk[i][1]
```

```
311                  + 2 * h_y  * (beta_p * b_b(x, y, tn) - beta * b_b(x, y, tnp1));
312  #ifndef ORIGINAL
313              B[L] /= 2;
314  #endif
315              /* (i, N_y) */
316              L = phi(i,N_y); y = d;
317              B[L] = gamma_p * Uk[i][N_y]
318                   + alpha_p * (Uk[i + 1][N_y] + Uk[i - 1][N_y])
319                   + 2 * beta_p * Uk[i][N_y - 1]
320                   + 2 * h_y * (beta_p * b_t(x, y, tn) - beta * b_t(x, y, tnp1));
321  #ifndef ORIGINAL
322              B[L] /= 2;
323  #endif
324          }
325          /* 左の辺、右の辺にある格子点（角の点は含めない）*/
326          for (j = 1; j < N_y; j++) {
327              y = c + j * h_y;
328              /* (0, j) */
329              L = phi(0,j); x = a;
330              B[L] = gamma_p * Uk[0][j]
331                   + 2 * alpha_p * Uk[1][j]
332                   + beta_p * (Uk[0][j + 1] + Uk[0][j - 1])
333                   + 2 * h_x * (alpha_p * b_l(x, y, tn) - alpha * b_l(x, y, tnp1));
334  #ifndef ORIGINAL
335              B[L] /= 2;
336  #endif
337              /* (N_x, j) */
338              L = phi(N_x,j); x = b;
339              B[L] = gamma_p * Uk[N_x][j]
340                   + 2 * alpha_p * Uk[N_x - 1][j]
341                   + beta_p * (Uk[N_x][j + 1] + Uk[N_x][j - 1])
342                   + 2 * h_x * (alpha_p * b_r(x, y, tn) - alpha * b_r(x, y, tnp1));
343  #ifndef ORIGINAL
344              B[L] /= 2;
345  #endif
346          }
347          /* 左下 */
348          L = phi(0,0); x = a; y = c;
349          B[L] = gamma_p * Uk[0][0]
350               + 2 * alpha_p * Uk[1][0] + 2 * beta_p * Uk[0][1]
351               + 2 * h_x * (alpha_p * b_l(x,y,tn) - alpha * b_l(x,y,tnp1))
352               + 2 * h_y * (beta_p * b_b(x,y,tn) - beta * b_b(x,y,tnp1));
353  #ifndef ORIGINAL
354              B[L] /= 4;
355  #endif
356          /* 左上 */
357          L = phi(0,N_y); x = a; y = d;
358          B[L] = gamma_p * Uk[0][N_y]
359               + 2 * alpha_p * Uk[1][N_y] + 2 * beta_p * Uk[0][N_y - 1]
360               + 2 * h_x * (alpha_p * b_l(x,y,tn) - alpha * b_l(x,y,tnp1))
361               + 2 * h_y * (beta_p * b_t(x,y,tn) - beta * b_t(x,y,tnp1));
362  #ifndef ORIGINAL
363              B[L] /= 4;
364  #endif
365          /* 右下 */
366          L = phi(N_x,0); x = b; y = c;
367          B[L] = gamma_p * Uk[N_x][0]
368               + 2 * alpha_p * Uk[N_x - 1][0] + 2 * beta_p * Uk[N_x][1]
369               + 2 * h_x * (alpha_p * b_r(x,y,tn) - alpha * b_r(x,y,tnp1))
370               + 2 * h_y * (beta_p * b_b(x,y,tn) - beta * b_b(x,y,tnp1));
```

```
371 #ifndef ORIGINAL
372             B[L] /= 4;
373 #endif
374         /* 右上 */
375         L = phi(N_x,N_y); x = b; y = d;
376         B[L] = gamma_p * Uk[N_x][N_y]
377              + 2 * alpha_p * Uk[N_x - 1][N_y] + 2 * beta_p * Uk[N_x][N_y - 1]
378              + 2 * h_x * (alpha_p * b_r(x,y,tn) - alpha * b_r(x,y,tnp1))
379              + 2 * h_y * (beta_p * b_t(x,y,tn) - beta * b_t(x,y,tnp1));
380 #ifndef ORIGINAL
381             B[L] /= 4;
382 #endif
383
384         /* A vector_U = B を解く */
385         solve(N, A, B, iwork);
386         /* */
387         for (i = 0; i <= N_x; i++)
388             for (j = 0; j <= N_y; j++)
389                 Uk[i][j] = B[phi(i,j)];
390
391         /* データを数値で表示 */
392         if (n % skip == 0) {
393 #ifdef  PRINT
394             for (i = 0; i <= N_x; i++) {
395                 for (j = 0; j <= N_y; j++)
396                     printf(" %6.2f", Uk[i][j]);
397                 printf("\n");
398             }
399 #endif
400
401             /* 鳥瞰図を描く */
402             g_cls();
403 #ifdef OLD
404             g_hidden2(1.0, 1.0, 0.4, -1.0, 1.0, 5.0, 25.0, 20.0, 20.0, 20.0,
405                       150.0, 100.0, Uk, N_x + 1, N_y + 1,
406                       1, G_SIDE_NONE, 2, 1);
407 #else
408             g_hidden(1.0, 1.0, 0.4, -1.0, 1.0, 5.0, 25.0, 20.0, 20.0, 20.0,
409                      150.0, 100.0, &Uk[0][0], N_x + 1, N_y + 1,
410                      1, G_SIDE_NONE, 2, 1);
411 #endif
412         }
413     }
414     /* マウスでクリックされるのを待つ */
415     g_sleep(-1.0);
416     /* ウィンドウを消す */
417     g_term();
418
419     return 0;
420 }
421
422 /* 初期値 */
423 double f(double x, double y)
424 {
425     /* ピラミッド型の関数 */
426     if (y > 0.5)
427         y = 1 - y;
428     if (x > 0.5)
429         x = 1 - x;
430     if (y < x)
```

```
431          return 5 * y;
432      else
433          return 5 * x;
434  }
435
436  /* Neumann 境界値 */
437  /* 上の辺での Neumann 境界値 ∂ u/∂ y (boundary value on the top side */
438  double b_t(double x, double y, double t)
439  {
440    return 0.0; /* 同次 Neumann 境界条件 */
441  }
442
443  /* 下の辺での Neumann 境界値 -∂ u/∂ y (boundary value on the bottom side */
444  double b_b(double x, double y, double t)
445  {
446    return 0.0; /* 同次 Neumann 境界条件 */
447  }
448
449  /* 左の辺での Neumann 境界値 ∂ u/∂ y (boundary value on the left side */
450  double b_l(double x, double y, double t)
451  {
452    return 0.0; /* 同次 Neumann 境界条件 */
453  }
454
455  /* 右の辺での Neumann 境界値 -∂ u/∂ y (boundary value on the right side */
456  double b_r(double x, double y, double t)
457  {
458    return 0.0; /* 同次 Neumann 境界条件 */
459  }
```

実行結果 (係数行列の確認)

```
% ./heat2n-i-naive2
Nx, Ny: 3 3
θ (0 ≦ θ ≦ 1): 0.5
τ (≦ 0.0555556 ＝最大値ノルムに関する安定性条件を満たすτの上限): 0.0555555555
λ=1
素朴に作った連立 1 次方程式の行列
    0.5 -0.125      0      0 -0.125      0      0      0      0      0      0      0      0      0      0      0      0
 -0.125      1 -0.125      0      0  -0.25      0      0      0      0      0      0      0      0      0      0      0
      0 -0.125      1 -0.125      0      0  -0.25      0      0      0      0      0      0      0      0      0      0
      0      0 -0.125    0.5      0      0      0 -0.125      0      0      0      0      0      0      0      0      0
 -0.125      0      0      0      1  -0.25      0      0 -0.125      0      0      0      0      0      0      0      0
      0  -0.25      0      0  -0.25      2  -0.25      0      0  -0.25      0      0      0      0      0      0      0
      0      0  -0.25      0      0  -0.25      2  -0.25      0      0  -0.25      0      0      0      0      0      0
      0      0      0 -0.125      0      0  -0.25      1      0      0      0 -0.125      0      0      0      0      0
      0      0      0      0 -0.125      0      0      0      1  -0.25      0      0 -0.125      0      0      0      0
      0      0      0      0      0  -0.25      0      0  -0.25      2  -0.25      0      0  -0.25      0      0      0
      0      0      0      0      0      0  -0.25      0      0  -0.25      2  -0.25      0      0  -0.25      0      0
      0      0      0      0      0      0      0 -0.125      0      0  -0.25      1      0      0      0 -0.125      0
      0      0      0      0      0      0      0      0 -0.125      0      0      0    0.5 -0.125      0      0      0
      0      0      0      0      0      0      0      0      0  -0.25      0      0 -0.125      1 -0.125      0      0
      0      0      0      0      0      0      0      0      0      0  -0.25      0      0 -0.125      1 -0.125      0
      0      0      0      0      0      0      0      0      0      0      0 -0.125      0      0 -0.125    0.5      0
備考: 1+2 θ λ= 2.00, -θ λ x=-0.25, -θ λ y=-0.25
Tmax: 1
Δ t: 0.0555555555
%
```

3.4.5 heat2n-i-v2.c

上のプログラム heat2n-i-naive2.c を、対称帯行列向け LU 分解プログラム symbandlu.c を利用するように書き換えたプログラム heat2n-i-v2.c を紹介する。

```
 1  /*
 2   * heat2n-i-v2.c --- 2 次元熱方程式 (Neumann 境界条件) を陰解法で解く
 3   *                   Neumann 境界値の扱いを変更
 4   *
 5   * http://nalab.mind.meiji.ac.jp/~mk/program/fdm/heat2n-i-v2.c
 6   * http://nalab.mind.meiji.ac.jp/~mk/program/fdm/smallmatrix.h
 7   * http://nalab.mind.meiji.ac.jp/~mk/program/linear/symbaldlu.c
 8   * http://nalab.mind.meiji.ac.jp/~mk/program/linear/symbaldlu.h
 9   *
10   * scp -p heat2n-i-v2.c nalab.mind.meiji.ac.jp:Sites/misc/20240821
11   *
12   * To compile
13   *     cglsc heat2n-i-v2.c symbandlu.c
14   * or
15   *     ccmg heat2n-i-v2.c symbandlu.c
16   *
17   *   このプログラムについては 1998 年度卒研の学生だった深石君に感謝します。
18   *
19   * version 2 での変更点
20   *  Neumann 境界値を返す関数をこれまで
21   *    double Phi(double x, double y, double t)
22   *  としていたのを
23   *    double b_t(double x, double y, double t) // -∂ u/∂ y(x,d,t)
24   *    double b_b(double x, double y, double t) // -∂ u/∂ y(x,c,t)
25   *    double b_l(double x, double y, double t) // -∂ u/∂ x(a,y,t)
26   *    double b_r(double x, double y, double t) //  ∂ u/∂ y(b,y,t)
27   *  という 4 つの関数にした。
28   *  引数の数を減らせるが、x,y,t のままにすることを選んだ (1 つダミーになるが)。
29   *    これまで、例えば左下の角点では
30   *
31   *      L = phi(0,0);
32   *      B[L] = gamma_p * Uk[0][0]
33   *           + 2 * alpha_p * Uk[1][0] + 2 * beta_p * Uk[0][1]
34   *           + 2 * h_x * (alpha * Phi(a,c,t) - alpha_p * Phi(a,c,t_p))
35   *           + 2 * h_y * (beta * Phi(a,c,t) - beta_p * Phi(a,c,t_p));
36   *           B[L] /= 4;
37   *
38   *   としていたのを
39   *
40   *      L = phi(0,0);
41   *      B[L] = gamma_p * Uk[0][0]
42   *           + 2 * alpha_p * Uk[1][0] + 2 * beta_p * Uk[0][1]
43   *           + h_x * (alpha * b_l(a,c,t) - alpha_p * b_l(a,c,t_p))
44   *           + h_y * (beta * b_b(a,c,t) - beta_p * b_b(a,c,t_p));
45   *          B[L] /= 4;
46   *
47   *   と改めた。これまでは (温めすぎ|冷やしすぎ) で、こうするのが正しいと
48   *   信じている。
49   *
50   *     下の辺での "修正"
51              // (i, 0)
52              L = phi(i,0);
53              B[L] = gamma_p * Uk[i][0]
54                   + alpha_p * (Uk[i + 1][0] + Uk[i - 1][0])
55                   + 2 * beta_p * Uk[i][1]
56                   + 2 * h_y  * (beta * b_b(x, c, t) - beta_p * b_b(x, c, t_p));
57              B[L] /= 2;
58   *     左の辺での "修正"
59              // (0, j)
60              L = phi(0,j);
```

```
 61              B[L] = gamma_p * Uk[0][j]
 62                   + 2 * alpha_p * Uk[1][j]
 63                   + beta_p * (Uk[0][j + 1] + Uk[0][j - 1])
 64                   + 2 * h_x * (alpha * b_l(a, y, t) - alpha_p * b_l(a, y, t_p));
 65              B[L] /= 2;
 66
 67   */
 68
 69  #include <stdio.h>
 70  #include <stdlib.h>
 71  #include <math.h>
 72  #ifdef OLD
 73  #include <matrix.h>
 74  #else
 75  #include "smallmatrix.h"
 76  #endif
 77  #ifndef G_DOUBLE
 78  #define G_DOUBLE
 79  #endif
 80  #include <glsc.h>
 81  #include "symbandlu.h"
 82
 83  // 格子点に番号をつける。
 84  // いわゆる column major mode である。
 85  // m = N_x + 1
 86  // 行列の最初の m 行, 最後の m 行は j=0,N_y（長方形領域の下辺、上辺）に対応する。
 87  #define phi(i,j) (j)*m+(i)
 88
 89  double sqr(double x) { return x * x; }
 90  double max(double a, double b) { if (a > b) return a; else return b; }
 91  void print_matrix(matrix, int, int);
 92
 93  int p_id = -1; //
 94  double pi;
 95
 96  int main(int argc, char **argv)
 97  {
 98      double a, b, c, d;
 99      int N_x, N_y, m, N, i, j, p, q, L, n, nMax;
100      matrix Uk, A;
101      double *B, *vector_U, cond;
102      int *iwork, skip;
103      double h_x, h_y, lambda_x, lambda_y, lambda, lambda_limit, tau, Tmax, dt;
104      double f(double, double);
105      double b_t(double, double, double), b_b(double, double, double);
106      double b_l(double, double, double), b_r(double, double, double);
107      double theta, gamma, alpha, beta, gamma_p, alpha_p, beta_p;
108      double x, y, tnp1, tn;
109      // 誤差
110      double exact_sol(double, double, double);
111      void print_problem(int);
112
113      /* 円周率 */
114      pi = 4.0 * atan(1.0);
115
116      if (argc == 2) {
117         p_id = atoi(argv[1]);
118         printf("p_id=%d\n", p_id);
119         print_problem(p_id);
120      }
```

```
121
122     /* 問題を考える区間 [a,b] × [c,d] */
123     a = 0.0; b = 1.0; c = 0.0; d = 1.0;
124
125     /* 区間の分割数 */
126     printf("Nx, Ny: "); scanf("%d %d", &N_x, &N_y);
127
128     m = N_x + 1;
129     N = (N_x + 1) * (N_y + 1);
130     /* 空間の刻み幅 */
131     h_x = (b - a) / N_x;
132     h_y = (d - c) / N_y;
133
134     /* 行列、ベクトルを記憶する変数のメモリー割り当て */
135     if ((Uk = new_matrix(N_x + 1, N_y + 1)) == NULL) {
136         fprintf(stderr, "数列 U^k を記憶する領域の確保に失敗\n");
137         exit(1);
138     }
139     if ((A = new_matrix(N, m + 1)) == NULL) {
140         fprintf(stderr, "係数行列 A を記憶する領域の確保に失敗\n");
141         exit(1);
142     }
143     if ((B = malloc(sizeof(double) * N)) == NULL) {
144         fprintf(stderr, "B を記憶する領域の確保に失敗\n");
145         exit(1);
146     }
147     if ((vector_U = malloc(sizeof(double) * N)) == NULL) {
148         fprintf(stderr, "vector_U を記憶する領域の確保に失敗\n");
149         exit(1);
150     }
151     if ((iwork = malloc(sizeof(int) * N)) == NULL) {
152         fprintf(stderr, "iwork を記憶する領域の確保に失敗\n");
153         exit(1);
154     }
155
156     /* θ法の重みの決定 */
157     printf("θ (0≦θ≦1): "); scanf("%lf", &theta);
158
159     if (theta == 1.0) {
160         printf("τ: "); scanf("%lf", &tau);
161     } else {
162         printf("τ (≦%g ≡最大値ノルムに関する安定性条件を満たすτの上限): ",
163                0.5 / (1 - theta) / (1 / (h_x * h_x) + 1 / (h_y * h_y)));
164         scanf("%lf", &tau);
165     }
166
167     lambda_x = tau / (h_x * h_x);
168     lambda_y = tau / (h_y * h_y);
169     lambda = lambda_x + lambda_y;
170
171     /* 最大値ノルムに関する安定性を満たすλの上限 */
172     lambda_limit = 1.0 / (2.0 * (1.0 - theta));
173
174     if (lambda > lambda_limit)
175         printf("注意: λ=%g>1/2(1-θ) となっています。\n", lambda);
176     else
177         printf("λ=%g\n", lambda);
178
179     /* 初期値の設定 */
180     for (i = 0; i <= N_x; i++) {
```

```
181       x = a + i * h_x;
182       for (j = 0; j <= N_y; j++)
183         Uk[i][j] = f(x, c + j * h_y);
184     }
185
186     /* 連立 1 次方程式に現れる係数
187        γ U_{ij}^{n+1}
188          +α (U_{i+1,j}^{n+1}+U_{i-1,j}^{n+1})
189          +β (U_{i,j+1}^{n+1}+U_{i,j-1}^{n+1})
190       =γ'U_{ij}^n
191          +α'(U_{i+1,j}^n+U_{i-1,j}^n)
192          +β'(U_{i,j+1}^n+U_{i,j-1}^n)
193      と書いたときのα, β, γ, α', β', γ' */
194     gamma = 1.0 + 2.0 * theta * lambda;
195     alpha = - theta * lambda_x;
196     beta  = - theta * lambda_y;
197     printf("gamma=%f, alpha=%f, beta=%f\n", gamma, alpha, beta);
198
199     gamma_p = 1.0 - 2.0 * (1.0 - theta) * lambda;
200     alpha_p = (1.0 - theta) * lambda_x;
201     beta_p  = (1.0 - theta) * lambda_y;
202     printf("gamma'=%f, alpha'=%f, beta'=%f\n", gamma_p, alpha_p, beta_p);
203
204     /* 係数行列の作成 */
205     /* まず 0 クリア */
206     for (p = 0; p < N; p++) {
207         for (q = 0; q <= m; q++)
208             A[p][q] = 0.0;
209     }
210     for (i = 0; i <= N_x; i++)
211         for (j = 0; j <= N_y; j++) {
212             L = phi(i, j);
213             /*
214             if (j != 0)   A[L][L - m] = beta;
215             if (i != 0)   A[L][L - 1] = alpha;
216             */
217                           A[L][0] = gamma; /* A[L][L    ] = gamma; */
218             if (i != N_x) A[L][1] = alpha; /* A[L][L + 1] = alpha; */
219             if (j != N_y) A[L][m] = beta;  /* A[L][L + m] = beta;  */
220         }
221
222     if (N < 20) {
223       printf("連立 1 次方程式の行列 (対称化前)\n");
224       print_matrix(A, N, m);
225     }
226
227     /* 左辺または右辺にある格子点 */
228     for (j = 1; j < N_y; j++) {
229         /* (0,j) */
230         L = phi(0,j);
231         A[L][0] /= 2.0; A[L][m] /= 2.0;
232         /* A[L][L - m] /= 2.0; A[L][L] /= 2.0; A[L][L + m] /= 2.0; */
233         /* (N_x,j) */
234         L = phi(N_x,j);
235         A[L][0] /= 2.0; A[L][m] /= 2.0;
236         /* A[L][L - m] /= 2.0; A[L][L] /= 2.0; A[L][L + m] /= 2.0; */
237     }
238
239     /* 下辺または上辺にある格子点 */
240     for (i = 1; i < N_x; i++) {
```

```
241         /* (i,0) */
242         L = phi(i,0);
243         A[L][0] /= 2.0; A[L][1] /= 2.0;
244         /* A[L][L - 1] /= 2.0; A[L][L] /= 2.0; A[L][L + 1] /= 2.0; */
245         /* (i,N_y) */
246         L = phi(i,N_y);
247         A[L][0] /= 2.0; A[L][1] /= 2.0;
248         /* A[L][L - 1] /= 2.0; A[L][L] /= 2.0; A[L][L + 1] /= 2.0; */
249     }
250     /* 角の点（対角成分は４で割る、それ以外は２で割る） */
251     /* 左下 */
252     L = phi(0,0);
253     A[L][0] /= 4.0; A[L][1] /= 2.0; A[L][m] /= 2.0;
254     /* A[L][L] /= 4.0; A[L][L+1] /= 2.0; A[L][L+m] /= 2.0; */
255     /* 左上 */
256     L = phi(0,N_y);
257     A[L][0] /= 4.0; A[L][1] /= 2.0;
258     /* A[L][L] /= 4.0; A[L][L+1] /= 2.0; A[L][L-m] /= 2.0; */
259     /* 右下 */
260     L = phi(N_x,0);
261     A[L][0] /= 4.0; A[L][m] /= 2.0;
262     /* A[L][L] /= 4.0; A[L][L-1] /= 2.0; A[L][L+m] /= 2.0; */
263     /* 右上 */
264     L = phi(N_x,N_y);
265     A[L][0] /= 4.0;
266     /* A[L][L] /= 4.0; A[L][L-1] /= 2.0; A[L][L-m] /= 2.0; */
267
268     /* 連立１次方程式の係数行列を表示する */
269     if (N < 20) {
270       printf("連立１次方程式の行列\n");
271       print_matrix(A, N, m);
272     }
273
274     printf("備考: 1+2 θ λ=%5.2f, -θ λ x=%5.2f, -θ λ y=%5.2f\n",
275            gamma, alpha, beta);
276
277     printf("Tmax: "); scanf("%lf", &Tmax);
278     printf("Δ t: ");  scanf("%lf", &dt);
279     skip = rint(dt / tau);
280     if (skip == 0) skip = 1;
281
282     nMax = rint(Tmax / tau);
283
284     /* グラフィックス・ライブラリィ GLSC の呼び出し */
285     g_init("Meta", 250.0, 160.0);
286     g_device(G_BOTH);
287     g_def_scale(0,
288                 0.0, 1.0, 0.0, 1.0,
289                 30.0, 70.0, 100.0, 72.0);
290     g_def_scale(4,
291                 -1.0, 1.0, -1.0, 1.0,
292                 30.0, 30.0, 100.0, 100.0);
293     g_def_line(0, G_BLACK, 0, G_LINE_SOLID);
294     g_sel_scale(0);
295
296     g_cls();
297 #ifdef OLD
298     g_hidden2(1.0, 1.0, 0.4, -1.0, 1.0, 5.0, 25.0, 20.0, 20.0, 20.0,
299               150.0, 100.0, Uk, N_x + 1, N_y + 1,
300               1, G_SIDE_NONE, 2, 1);
```

```
#else
    g_hidden(1.0, 1.0, 0.4, -1.0, 1.0, 5.0, 25.0, 20.0, 20.0, 20.0,
             150.0, 100.0, &Uk[0][0], N_x + 1, N_y + 1,
             1, G_SIDE_NONE, 2, 1);
#endif
    printf("一時停止します。再開にはウィンドウをクリックして下さい。\n");
    g_sleep(-1.0);

    /* 係数行列 LU 分解 */
    symbandlu(A, N, m + 1);
    if (cond + 1 == cond) {
        /* 条件数が大きければ、計算をあきらめる */
        printf("MATRIX IS SINGULAR TO WORKING PRECISION\n");
        return 0;
    }

    /* 時間に関するループ */
    for (n = 0; n < nMax; n++) {
        tnp1 = (n + 1) * tau; tn = n * tau;

        /* まず、素朴な連立 1 次方程式の右辺を用意する */
        /* 内部の格子点 */
        for (i = 1; i < N_x; i++)
            for (j = 1; j < N_y; j++) {
                L = phi(i,j);
                B[L] = gamma_p * Uk[i][j]
                    + alpha_p * (Uk[i + 1][j] + Uk[i - 1][j])
                    + beta_p * (Uk[i][j + 1] + Uk[i][j - 1]);
            }
        /* 以下、境界にある格子点での方程式を立てる。
         * 仮想格子点での値は境界条件を中心差分近似した方程式を
         * 用いて消去する
         *   ……右辺にも移項する量があるので、右辺について後で処理する */
        /* 下の辺、上の辺にある格子点（角の点は含めない）*/
        for (i = 1; i < N_x; i++) {
            x = a + i * h_x;
            /* (i, 0) */
            L = phi(i,0); y = c;
            B[L] = gamma_p * Uk[i][0]
                 + alpha_p * (Uk[i + 1][0] + Uk[i - 1][0])
                 + 2 * beta_p * Uk[i][1]
                 + 2 * h_y * (beta_p * b_b(x, y, tn) - beta * b_b(x, y, tnp1));
            B[L] /= 2;
            /* (i, N_y) */
            L = phi(i,N_y); y = d;
            B[L] = gamma_p * Uk[i][N_y]
                 + alpha_p * (Uk[i + 1][N_y] + Uk[i - 1][N_y])
                 + 2 * beta_p * Uk[i][N_y - 1]
                 + 2 * h_y * (beta_p * b_t(x, y, tn) - beta * b_t(x, y, tnp1));
            B[L] /= 2;
        }
        /* 左の辺、右の辺にある格子点（角の点は含めない）*/
        for (j = 1; j < N_y; j++) {
            y = c + j * h_y;
            /* (0, j) */
            L = phi(0,j); x = a;
            B[L] = gamma_p * Uk[0][j]
                 + 2 * alpha_p * Uk[1][j]
                 + beta_p * (Uk[0][j + 1] + Uk[0][j - 1])
                 + 2 * h_x * (alpha_p * b_l(x, y, tn) - alpha * b_l(x, y, tnp1));
```

```
361             B[L] /= 2;
362             /* (N_x, j) */
363             L = phi(N_x,j); x = b;
364             B[L] = gamma_p * Uk[N_x][j]
365                  + 2 * alpha_p * Uk[N_x - 1][j]
366                  + beta_p * (Uk[N_x][j + 1] + Uk[N_x][j - 1])
367                  + 2 * h_x * (alpha_p * b_r(x, y, tn) - alpha * b_r(x, y, tnp1));
368             B[L] /= 2;
369         }
370         /* 左下 (left,bottom) の角点 */
371         L = phi(0,0); x = a; y = c;
372         B[L] = gamma_p * Uk[0][0]
373              + 2 * alpha_p * Uk[1][0] + 2 * beta_p * Uk[0][1]
374              + 2 * h_x * (alpha_p * b_l(x,y,tn) - alpha * b_l(x,y,tnp1))
375              + 2 * h_y * (beta_p * b_b(x,y,tn) - beta * b_b(x,y,tnp1));
376         B[L] /= 4;
377         /* 左上 (left,top) の角点 */
378         L = phi(0,N_y); x = a; y = d;
379         B[L] = gamma_p * Uk[0][N_y]
380              + 2 * alpha_p * Uk[1][N_y] + 2 * beta_p * Uk[0][N_y - 1]
381              + 2 * h_x * (alpha_p * b_l(x,y,tn) - alpha * b_l(x,y,tnp1))
382              + 2 * h_y * (beta_p * b_t(x,y,tn) - beta * b_t(x,y,tnp1));
383         B[L] /= 4;
384         /* 右下 (right,bottom) の角点 */
385         L = phi(N_x,0); x = b; y = c;
386         B[L] = gamma_p * Uk[N_x][0]
387              + 2 * alpha_p * Uk[N_x - 1][0] + 2 * beta_p * Uk[N_x][1]
388              + 2 * h_x * (alpha_p * b_r(x,y,tn) - alpha * b_r(x,y,tnp1))
389              + 2 * h_y * (beta * b_b(x,y,tn) - beta_p * b_b(x,y,tnp1));
390         B[L] /= 4;
391         /* 右上 (right,top) の角点 */
392         L = phi(N_x,N_y); x = b; y = d;
393         B[L] = gamma_p * Uk[N_x][N_y]
394              + 2 * alpha_p * Uk[N_x - 1][N_y] + 2 * beta_p * Uk[N_x][N_y - 1]
395              + 2 * h_x * (alpha_p * b_r(x,y,tn) - alpha * b_r(x,y,tnp1))
396              + 2 * h_y * (beta_p * b_t(x,y,tn) - beta * b_t(x,y,tnp1));
397         B[L] /= 4;
398
399         /* A vector_U = B を解く */
400         symbandsolve(A, B, N, m + 1);
401         /* */
402         for (i = 0; i <= N_x; i++)
403             for (j = 0; j <= N_y; j++)
404                 Uk[i][j] = B[phi(i,j)];
405
406         /* データを数値で表示 */
407         if (n % skip == 0) {
408           double errorp, error, sum;
409 #ifdef  PRINT
410             for (i = 0; i <= N_x; i++) {
411                 for (j = 0; j <= N_y; j++)
412                     printf(" %5.1f", Uk[i][j]);
413                 printf("\n");
414             }
415 #endif
416
417             /* 鳥瞰図を描く */
418             g_cls();
419 #ifdef OLD
420             g_hidden2(1.0, 1.0, 0.4, -1.0, 1.0, 5.0, 25.0, 20.0, 20.0, 20.0,
```

```
421                       150.0, 100.0, Uk, N_x + 1, N_y + 1,
422                       1, G_SIDE_NONE, 2, 1);
423 #else
424             g_hidden(1.0, 1.0, 0.4, -1.0, 1.0, 5.0, 25.0, 20.0, 20.0, 20.0,
425                      150.0, 100.0, &Uk[0][0], N_x + 1, N_y + 1,
426                      1, G_SIDE_NONE, 2, 1);
427 #endif
428             error = 0.0; sum = 0.0;
429             for (i = 0; i <= N_x; i++) {
430               x = a + i * h_x;
431               for (j = 0; j <= N_y; j++) {
432                 y = c + j * h_y;
433                 // error += sqr(exact_sol(x, y, t) - Uk[i][j]);
434                 errorp = sqr(exact_sol(x, y, tnp1) - Uk[i][j]);
435                 if (errorp > error)
436                   error = errorp;
437                 sum += Uk[i][j];
438               }
439             }
440             printf("error=%e, sum=%e\n", error / ((N_x+1)*(N_y+1)),
441                                          h_x * h_y *sum);
442         }
443     }
444     /* マウスでクリックされるのを待つ */
445     g_sleep(-1.0);
446     /* ウィンドウを消す */
447     g_term();
448
449     return 0;
450 }
451
452 void error(int p_id)
453 {
454   fprintf(stderr, "unknown p_id=%d\n", p_id);
455   exit(-1);
456 }
457
458 void print_problem(int id)
459 {
460   if (id == 0)
461     printf("初期値: cos(π x)cos(π y), Neumann 境界値: 0\n");
462   else if (id == 1)
463     printf("初期値: (x-1/2)^2-(y-1/2)^2+cos(π x)cos(π y), Neumann 境界値: 1\n");
464   else if (id == 2)
465     printf("初期値: (x-1/2)(y-1/2)+cos(π x)cos(π y), Neumann 境界値: 1\n");
466 }
467
468 /* 初期値 */
469 double f(double x, double y)
470 {
471   if (p_id == 0)
472     return cos(pi * x) * cos(pi * y);
473   else if (p_id == 1)
474     return sqr(x - 0.5) - sqr(y - 0.5) + cos(pi * x) * cos(pi * y);
475   else if (p_id == 2)
476     return (x - 0.5) * (y - 0.5) + cos(pi * x) * cos(pi * y);
477   else
478     error(p_id);
479   return 0.0; // dummy
480 }
```

```
481
482 /* 上の辺での Neumann 境界値 ∂ u/∂ y (boundary value on the top side */
483 double b_t(double x, double y, double t)
484 {
485   if (p_id == 0)
486     return 0.0; /* 同次 Neumann 境界条件 */
487   else if (p_id == 1)
488     return - 1.0;
489   else if (p_id == 2)
490     return x - 0.5;
491   else
492     error(p_id);
493   return 0.0; // dummy
494 }
495
496 /* 下の辺での Neumann 境界値 -∂ u/∂ y (boundary value on the bottom side */
497 double b_b(double x, double y, double t)
498 {
499   if (p_id == 0)
500     return 0.0; /* 同次 Neumann 境界条件 */
501   else if (p_id == 1)
502     return - 1.0;
503   else if (p_id == 2)
504     return 0.5 - x;
505   else
506     error(p_id);
507   return 0.0; // dummy
508 }
509
510 /* 左の辺での Neumann 境界値 ∂ u/∂ y (boundary value on the left side */
511 double b_l(double x, double y, double t)
512 {
513   if (p_id == 0)
514     return 0.0; /* 同次 Neumann 境界条件 */
515   else if (p_id == 1)
516     return 1.0;
517   else if (p_id == 2)
518     return 0.5 - y;
519   else
520     error(p_id);
521   return 0.0; // dummy
522 }
523
524 /* 右の辺での Neumann 境界値 -∂ u/∂ y (boundary value on the right side */
525 double b_r(double x, double y, double t)
526 {
527   if (p_id == 0)
528     return 0.0; /* 同次 Neumann 境界条件 */
529   else if (p_id == 1)
530     return 1.0;
531   else if (p_id == 2)
532     return y - 0.5;
533   else
534     error(p_id);
535   return 0.0; // dummy
536 }
537
538 double exact_sol(double x, double y, double t)
539 {
540   if (p_id == 0)
```

```
541       return exp(- 2 * pi * pi * t) * cos(pi * x) * cos(pi * y);
542     else if (p_id == 1)
543       return sqr(x - 0.5) - sqr(y - 0.5)
544         + exp(- 2.0 * pi * pi * t) * cos(pi * x) * cos(pi * y);
545     else if (p_id == 2)
546       return (x - 0.5) * (y - 0.5)
547         + exp(- 2.0 * pi * pi * t) * cos(pi * x) * cos(pi * y);
548     else
549       error(p_id);
550     return 0.0; // dummy
551   }
552
553   void print_matrix(matrix A, int N, int m)
554   {
555     int i, j;
556     double t;
557     for (i = 0; i < N; i++) {
558       for (j = 0; j < N; j++) {
559         if (i - j < - m || i - j > m)
560           t = 0;
561         else if (i <= j)
562           t = A[i][j-i];
563         else
564           t = A[j][i-j];
565         printf("%8.4g ", t);
566       }
567       printf("\n");
568     }
569   }
```

3.4.6 MATLAB プログラム

MATLAB プログラムを作ってある。そのうち、この文書とマージするつもりであるが、現在は別文書にしてある。

桂田 [16] (2015), [17] (2024), [19] (2024) を見よ。それぞれ同次 Ditichlet 境界条件の場合、非同次 Ditichlet 境界条件の場合、非同次 Neumann 境界条件の場合を説明してある。

この問題は一応解決した (つもり)。ADI 法で解く人が多いような気がしているので、この辺の解説が実際上役に立つかは分からないけれど。

こちらへのマージはどうやろうかな。

総熱量の保存とか、差分法プログラムの書き方だけでなく、どういう連立 1 次方程式になるか公式として示していることとか、それから行列を対称化するテクニックについて、現時点で分かっていることを説明していることとか。なるべく早くこちらに取り込みたい。

3.5 将来の課題等

3.5.1 効率的なプログラムの作成

前節までに色々なプログラムを紹介したが、いくつか不満な点が残っている。卒業レポート課題などで解決してもらえると嬉しい。

- LAPACK, LAPACK++ などの利用も検討したい。

- 反復法 (SOR 法, CG 法) のプログラムがない (卒研でやった人はいるけれど、ここに取り込める状態になっていない)。

3.5.2 多次元領域の問題ではどうすれば効率的か？

ここはかなり以前に書いたもので、書き直したいのだが…

1 次元では、文句なしに陰解法が良い 空間 1 次元の場合、λ を共通にしたとき、陰解法の (時間) 計算量は、陽解法のそれのせいぜい数倍程度であった。したがって、陰解法で λ をほんの少し大きく取るだけで、計算量の観点から有利になった (陽解法では安定性の要請から λ を大きく取れない)。また精度の面でも Crank-Nicolson 法を使えば陽解法と互角以上であった。要するに

空間 1 次元の問題では陰解法が陽解法よりもはっきりと優れている。

陰解法のココロ

簡単な陽解法以外の方法が色々あるのは、安定性を保ったまま、λ をなるべく大きく (言い換えると時間刻み τ をなるべく大きく) 取ることによって、問題を効率的に解くためである。

余談 3.5.1 演習で学生に実験をやらせると、せっかく陰解法で解いているのに、λ を 1/4 程度に取って計算したりする。これでは計算が遅くなってしまって陰解法を用いる理由があまりないと考えられる (もっとも長時間にわたって解く必要がある場合に、Crank-Nicolson 法を用いるというのなら、誤差の τ に関する次数が高く精度が高いのでそれなりに意味はあるかもしれない)。最近のコンピューターは速いので、空間 1 次元の問題では、それでも十分速く解けるため、あまり気にならないのだろうが、側で見ている者としては少し複雑な気持ちになる。■

2 次元は微妙？ 以上の事情は、2 次元領域ではかなり異なってくる。陽解法の場合は、時間に関して 1 ステップ進めるのに $O(N_xN_y)$ の計算量で済む。一方、陰解法については、現われる連立 1 次方程式の係数行列は、帯行列ではあるが、三重対角のような簡単なものではない。未知数の個数は約 N_xN_y で、係数行列の半バンド幅は約 N_x または N_y となるので、Gauss の消去法で LU 分解するには、$O(N_x^3N_y)$ または $O(N_xN_y^3)$ の計算量が必要である。また LU 分解した後に、個々の連立 1 次方程式を解くのに必要な計算量は $O(N_x^2N_y)$ または $O(N_xN_y^2)$ である。つまり、陰解法において普通の直接法を用いると、時間に関して 1 ステップ進めるのに、$O(N_x^2N_y)$ または $O(N_xN_y^2)$ の計算量が必要になる。陽解法と比べると巾が 1 違う。従ってこのやり方で連立 1 次方程式を解く限り、λ を N_x または N_y 程度の大きさにしないと (時間の刻み幅 τ を N_x or N_y 倍程度にしないと)、計算量の点で太刀打ちできない、ということになる。

そこで、次のような問題が出て来る。

(1) $\lambda = CN_a^\alpha$ (C, α は正定数、例えば $\alpha = 1/2, 1$) のようにした場合、差分スキームの適合性 (consistency), 安定性、差分解の精度はどうなるか？
→安定性については、少なくとも理論的には以下のことが分かっている。

(a) 最大値ノルムで安定性を考えると、θ を 1 にするか、1 に非常に近い値にしないと安定性が保証できない。

(b) 2 ノルムで安定性を考えると、$\theta \geq 1/2$ であれば大丈夫。

→精度については以下のように考えられる。

(a) 一般には

$$2\text{ ノルムによる誤差} = O(\tau + h_x^2 + h_y^2)$$

という精度しか保証できないので、τ を大きくすると、精度は陽解法よりもはっきり悪くなると考えられる。

(b) Crank-Nicolson 法など、特別の場合には

$$2\text{ ノルムによる誤差} = O(\tau^2 + h_x^2 + h_y^2)$$

のような精度が保証される。この場合には、精度に関しては τ を h_x や h_y と同程度の大きさにしても、それほど精度は落ちないと考えられる (あまり自信がないので誰か確かめて欲しい)。

(2) 反復法を使うと、陰解法で 1 ステップ進めるのに必要な計算量が軽減できるか？
→ 反復法の場合には、十分な精度の得られる反復回数の見積りをしないと計算量の解析ができない。定常反復法の場合には、少なくとも長方形領域などの簡単な場合には、係数行列のスペクトル半径の値が分るが、反復法の初期値の精度の収束への影響が無視できないため、あまり簡単ではない (熱方程式のような非定常問題では、1 ステップ前の解を初期値に採用するのが普通であるが、それはどの程度の精度があると考えれば良いだろうか？ とか)。非定常反復法の場合についてはまた別の難しさがある。実験的に色々な主張を見たことはあるが、理論的な分析は可能だろうか？

(3) 空間 3 次元ではどうなるか？

3.5.3 2017 年 3 月メモ

この節の前項までを書いたのは随分以前のことである。今回、久しぶりにプログラムの見直しをすることにした。

この文書も加筆・修正する可能性 (必要性?) があるが、現時点で少し補足しておく。

1. 行列の Kronecker 積を用いた連立 1 次方程式の係数行列表現 (3.3.8 参照) は、それなりに便利であるが、Poisson 方程式の場合ほど見通しが良いものではなく、それだけで済ませられるものでもないだろうと考えている。

2. 差分方程式から得られる (連立 1 次方程式の) 係数行列はどうすると対称行列になるか、なぜ対称になるか、きちんと説明をしたいと考えているが、(正直に白状すると) 実は良く理解できていない。

3. ここ数年、MATLAB が便利に使える場合は、C で書くことにこだわらずに MATLAB を使うことにしている。解説をここに取り込むべきかもしれないが、今の所は別文書にしてある (大きな行列を書こうとすると、紙のサイズが A4 では窮屈とか…)。

   - Dirichlet 条件の場合: https://m-katsurada.sakura.ne.jp/labo/text/heat2d.pdf
   - Neumann 条件の場合: https://m-katsurada.sakura.ne.jp/labo/text/heat2n.pdf

4. C++ はほぼ C 言語の上位互換で、色々便利なクラス・ライブラリィが用意されているので、これまで C 言語で記述していたプログラムを、C++ に移行させようと考えている。C++ には線形演算用のクラス・ライブラリィも色々あり、現在 **Eigen**[25] に注目している (Eigen では、"Eigen is a C++ template library for linear algebra" と言っている)。とりあえず 2 つのプログラムを紹介しておく (`heat2d-i.c`, `heat2n-i.c` と同じ差分方程式の計算をしている)。疎行列や連立 1 次方程式の扱いは Eigen に任せたため、(プログラムを書くのはある意味で) 簡単である。実行効率はかなり高い。

   - https://m-katsurada.sakura.ne.jp/program/fdm/heat2d-i-eigen-sparse.cpp
   - https://m-katsurada.sakura.ne.jp/program/fdm/heat2n-i-eigen-sparse.cpp

3.6 ADI 法

ADI 法 (alternating direction implicit method, 交互方向陰解法) は、最初

> Peaceman, D. W., and Rachford, H. H., The numerical solution of parabolic and elliptic differential equations, J. Soc. Indust. Appl. Math. **3** (1955), 28–41.

で提案された。

以下に説明する差分スキームのサンプル・プログラム (`heat2d-adi.c`, `heat2n-adi.c`) は、これまでと同様

https://m-katsurada.sakura.ne.jp/program/index.html#heat2d

においてある。

3.6.1 差分方程式

ADI 法とは、時間に関する 1 ステップを半分に分割し、各半ステップの一方では x 方向についてのみ陰解法を施し、もう一方では y 方向についてのみ陰解法を施す、つまり交互に陰解法を適用する方向を入れ換える、という方法である。

以下このことを

$$u_t(x,y,t) = \triangle u(x,y,t) \quad (\text{in } \Omega\times(0,\infty)), \tag{3.17}$$

$$u(x,y,t) = \alpha(x,y) \quad ((x,y,t)\in\partial\Omega\times(0,\infty)) \tag{3.18}$$

$$u(x,y,0) = f(x,y) \quad ((x,y)\in\overline{\Omega}) \tag{3.19}$$

という初期値境界値問題に即して説明する。

熱伝導方程式を ADI 法により近似して作った差分方程式は

$$\frac{U_{i,j}^{n+1/2}-U_{i,j}^{n}}{\tau/2} = \frac{U_{i+1,j}^{n+1/2}-2U_{i,j}^{n+1/2}+U_{i-1,j}^{n+1/2}}{h_x^2}+\frac{U_{i,j+1}^{n}-2U_{i,j}^{n}+U_{i,j-1}^{n}}{h_y^2},$$

$$\frac{U_{i,j}^{n+1}-U_{i,j}^{n+1/2}}{\tau/2} = \frac{U_{i+1,j}^{n+1/2}-2U_{i,j}^{n+1/2}+U_{i-1,j}^{n+1/2}}{h_x^2}+\frac{U_{i,j+1}^{n+1}-2U_{i,j}^{n+1}+U_{i,j-1}^{n+1}}{h_y^2}$$

[25] `http://http://eigen.tuxfamily.org/`

である。

$\lambda_x := \tau/(2h_x^2)$, $\lambda_y := \tau/(2h_y^2)$ として、まず n から $n+1/2$ 段へは、

$$(1+2\lambda_x)U_{i,j}^{n+1/2} - \lambda_x(U_{i+1,j}^{n+1/2} + U_{i-1,j}^{n+1/2}) = (1-2\lambda_y)U_{i,j}^{n} + \lambda_y(U_{i,j+1}^{n} + U_{i,j-1}^{n}) \quad \left(\begin{array}{l} 1 \le i \le N_x - 1, \\ 1 \le j \le N_y - 1 \end{array} \right).$$

ゆえに行列、ベクトルで連立 1 次方程式の形に表すと、

$$\begin{pmatrix} 1+2\lambda_x & -\lambda_x & & & & \\ -\lambda_x & 1+2\lambda_x & -\lambda_x & & & \\ & -\lambda_x & 1+2\lambda_x & -\lambda_x & & \\ & & \ddots & \ddots & \ddots & \\ & & & -\lambda_x & 1+2\lambda_x & -\lambda_x \\ & & & & -\lambda_x & 1+2\lambda_x \end{pmatrix} \begin{pmatrix} U_{1,j}^{n+1/2} \\ U_{2,j}^{n+1/2} \\ U_{3,j}^{n+1/2} \\ \vdots \\ U_{N_x-2,j}^{n+1/2} \\ U_{N_x-1,j}^{n+1/2} \end{pmatrix}$$

$$= \begin{pmatrix} (1-2\lambda_y)U_{1,j}^{n} + \lambda_y(U_{1,j+1}^{n} + U_{1,j-1}^{n}) \\ (1-2\lambda_y)U_{2,j}^{n} + \lambda_y(U_{2,j+1}^{n} + U_{2,j-1}^{n}) \\ (1-2\lambda_y)U_{3,j}^{n} + \lambda_y(U_{3,j+1}^{n} + U_{3,j-1}^{n}) \\ \vdots \\ (1-2\lambda_y)U_{N_x-2,j}^{n} + \lambda_y(U_{N_x-2,j+1}^{n} + U_{N_x-2,j-1}^{n}) \\ (1-2\lambda_y)U_{N_x-1,j}^{n} + \lambda_y(U_{N_x-1,j+1}^{n} + U_{N_x-1,j-1}^{n}) \end{pmatrix} + \begin{pmatrix} \lambda_x U_{0,j}^{n+1/2} \\ 0 \\ \vdots \\ \vdots \\ 0 \\ \lambda_x U_{N_x,j}^{n+1/2} \end{pmatrix}$$

$(j = 1, 2, \cdots, N_y - 1)$.

次に $n+1/2$ から $n+1$ 段へは

$$(1+2\lambda_y)U_{i,j}^{n+1} - \lambda_y(U_{i,j+1}^{n+1} + U_{i,j-1}^{n+1}) = (1-2\lambda_x)U_{i,j}^{n+1/2} + \lambda_x(U_{i+1,j}^{n+1/2} + U_{i-1,j}^{n+1/2}) \quad \left(\begin{array}{l} 1 \le i \le N_x - 1, \\ 1 \le j \le N_y - 1 \end{array} \right).$$

すなわち

$$\begin{pmatrix} 1+2\lambda_y & -\lambda_y & & & & \\ -\lambda_y & 1+2\lambda_y & -\lambda_y & & & \\ & -\lambda_y & 1+2\lambda_y & -\lambda_y & & \\ & & \ddots & \ddots & \ddots & \\ & & & -\lambda_y & 1+2\lambda_y & -\lambda_y \\ & & & & -\lambda_y & 1+2\lambda_y \end{pmatrix} \begin{pmatrix} U_{i,1}^{n+1} \\ U_{i,2}^{n+1} \\ U_{i,3}^{n+1} \\ \vdots \\ U_{i,N_y-2}^{n+1} \\ U_{i,N_y-1}^{n+1} \end{pmatrix}$$

$$= \begin{pmatrix} (1-2\lambda_x)U_{i,1}^{n+1/2} + \lambda_x(U_{i+1,1}^{n+1/2} + U_{i-1,1}^{n+1/2}) \\ (1-2\lambda_x)U_{i,2}^{n+1/2} + \lambda_x(U_{i+1,2}^{n+1/2} + U_{i-1,2}^{n+1/2}) \\ (1-2\lambda_x)U_{i,3}^{n+1/2} + \lambda_x(U_{i+1,3}^{n+1/2} + U_{i-1,3}^{n+1/2}) \\ \vdots \\ (1-2\lambda_x)U_{i,N_y-2}^{n+1/2} + \lambda_x(U_{i+1,N_y-2}^{n+1/2} + U_{i-1,N_y-2}^{n+1/2}) \\ (1-2\lambda_x)U_{i,N_y-1}^{n+1/2} + \lambda_x(U_{i+1,N_y-1}^{n+1/2} + U_{i-1,N_y-1}^{n+1/2}) \end{pmatrix} + \begin{pmatrix} \lambda_y U_{i,0}^{n+1} \\ 0 \\ \vdots \\ \vdots \\ 0 \\ \lambda_y U_{i,N_y}^{n+1} \end{pmatrix}$$

$(i = 1, 2, \cdots, N_x - 1)$.

3.6.2 安定性と収束性

安定性については、詳しくは、4.6 「ADI 法の安定性解析」を見よ。τ がどんなに大きくても安定 (無条件安定) という結果がある。

収束証明を書いたことはないが、

$$\|(a_{ij})\| = \left(\sum_{i,j} |a_{ij}|^2 \right)^{1/2}$$

というノルムについての収束証明は得られる、と信じている (だれか初等的な証明を書かないかな)。

3.6.3 効率に関する考察

以上から、ADI 法の計算では、t_n から t_{n+1} までの間に、$N_x - 1$ 未知数の連立 1 次方程式を $N_y - 1$ 個、$N_y - 1$ 未知数の連立 1 次方程式を $N_x - 1$ 個、解く必要がある。連立 1 次方程式はいずれも 3 項方程式なので、Gauss の消去法を用いて、$O(N_x N_y)$ 程度の演算回数で解けることが分かる。これは[26]、陽解法と ADI 法では、共通の τ を取る場合に、計算量が定数倍程度しか違わず、θ 法と比べると格段に計算量が少ないということを意味する。

補足: θ 法を消去法で解く場合の計算量

係数行列が、未知数の個数 N, 半バンド幅 m の帯行列である場合、LU 分解や解を求めるのに要する演算量はそれぞれ $O(Nm^2)$, $O(Nm)$ である。
長方形領域における熱方程式を θ 法で解く場合、この文書で解説した方法を採用すると、$N \fallingdotseq N_x N_y$, $m \fallingdotseq N_x$ であるから、演算量は、LU 分解が $O(N_x^3 N_y)$, 連立 1 次方程式の求解が $O(N_x^2 N_y)$ となる。熱方程式の場合、LU 分解は最初に 1 度だけすればよいので、目をつむるにしても、連立 1 次方程式の求解に要する計算量が、ADI 法 $O(N_x N_y)$ vs. θ 法 $O(N_x^2 N_y)$ で、冪が 1 異なることに注目しよう。 ■

というわけで、ただの熱伝導方程式を解くだけならば、θ 法よりは ADI 法がお勧めである。しかし熱伝導方程式をもじって、1 階の空間微分があるような方程式を (物理的には、媒質の流れがあるような現象) 考えると、ADI 法はあまりうまく行かなくなる (と言われているらしい)。そのような応用を考えると、結局は θ 法のような解法の重要性は揺るがない、ということらしい (あまり自信がない)。

3.6.4 実験の手引 — ある年の「応用数理実験」から

(古い話で、今見ると「何だこれは」という印象があるが、新しいのを書く気もしないので、とりあえず放置。)

2 次元領域における熱伝導方程式の初期値境界値問題に対する差分法として、陽解法、θ 法 (陰解法), ADI 法を解説した。簡単な陽解法以外の方法が色々あるのは、安定性を保ったまま、時間刻み τ をなるべく大きく取ることによって、問題を効率的に解くためである。そこで、次のような実験により、高級な解法を用いることの効用を確かめてみよう。

[26] `trilu()` では、引き算、掛け算、割り算をそれぞれ $n-1$ 回、`trisol()` では、引き算と掛け算をそれぞれ $2(n-1)$ 回、割り算を n 回実行するようになっている。

1. 分割数を変えながら、陽解法のプログラムの計算時間を測り、分割数と計算時間の関係を調べる。(もちろん、同じ初期値境界値問題を、λ など、分割数以外のパラメーターはすべて固定して解く。) 計算時間は `time` コマンドを使って測定する。

```
oyabun% time heat2d-e
N: 10
λ: 0.2
Tmax: 0.1
τ=0.002 ですが、Δtをどうしますか: 0.01
0.220u 0.300s 0:06.40 8.1% 0+614k 0+3io 0pf+0w
oyabun%
```

   これは計算時間 0.220 秒、システム時間 0.300 秒、経過時間 6.40 秒と言うことを表している。

2. 上と同様のことを ADI 法のプログラムで行なう。

3. 厳密解が簡単に分かるような初期値をいくつか用意して、それぞれの問題を ($\lambda = 1/2,\ 1/4$ で) 陽解法で解き、計算誤差と計算時間を測定する。

   - 誤差については、($T_{\max} = K\tau$ として)、離散化最大値ノルム

   $$\max_{0\le i\le N_x, 0\le j\le N_y} \left|u(x_i, y_j, T_{\max}) - U^K_{i,j}\right|$$

   や離散化 L^2 ノルム

   $$\left(\frac{1}{N_x N_y} \sum_{0\le i\le N_x, 0\le j\le N_y} \left|u(x_i, y_j, T_{\max}) - U^K_{i,j}\right|^2\right)^{1/2}$$

   などを指標とする。

   - 分割数と計算時間の関係については、簡単のため $N_x = N_y$ として、N_x と計算時間をグラフにプロット (必要ならば対数目盛) するのが良い。
   - (陽解法であるから)、$\lambda \equiv \tau/h_x^2 + \tau/h_y^2 \le 1/2$ でないと不安定になるので、せいぜい $\lambda = 1/2,\ 1/4$ で実験すれば十分。いくつかの初期値、$\lambda = 1/2$ または $1/4$, 適当に固定した一つの $T_{\max}$ について、N_x を変えながら、誤差と計算時間を測定する。

4. 同じ初期値境界値問題を ADI 法を用いて解く。その際、区間の分割は陽解法と同じにするが、時間刻み τ は色々変えて、精度がどのように変化するか調べる。そうして、陽解法と同程度の精度が得られる範囲で、どこまで大きくできる (速く計算できる) か調べる。その場合の計算時間を測って、陽解法と比較する。
   ADI 法について、λ 以外のパラメーターはなるべく陽解法と同じ値を選ぶ (比較するときの常識)。λ については、陽解法と同じ値から始めて、順々に大きくして行く。L^2 ノルム的には、λ をいくら大きくとっても安定であることが証明できる。そこで問題は、精度を落さない範囲でどこまで λ を大きく取れるか？ であるが、理論的には

   $$\text{誤差} = O(\tau^2 + h_x^2 + h_y^2)$$

であることが分かっている。これから τ は h_x, h_y と同程度、つまり λ について言えば、$N_x(=N_y)$ 程度まで大きく取れる可能性があることが分かる。そこで、λ の値としては、少なくとも $1/2$, $\sqrt{N_x}$, N_x の 3 つの場合について調べること ($T_{\max}$ をなるべく共通に選べるよう、少しくらいは変化させても良い)。

3.6.5 付録: Neumann 境界条件の場合

$$u_t = \triangle u \quad (\text{in } \Omega \times (0,\infty)), \tag{3.20}$$

$$\frac{\partial u}{\partial n}(x,y,t) = \beta(x,y) \quad ((x,y,t) \in \partial\Omega \times (0,\infty)) \tag{3.21}$$

$$u(x,y,0) = f(x,y) \quad ((x,y) \in \overline{\Omega}) \tag{3.22}$$

という初期値境界値問題に即して説明する。

第 n 段から第 $n+1/2$ 段まで。まず $1 \le i \le N_x-1$, $1 \le j \le N_y-1$ に対しては

$$(1+2\lambda_x)U_{i,j}^{n+1/2} - \lambda_x(U_{i+1,j}^{n+1/2} + U_{i-1,j}^{n+1/2}) = (1-2\lambda_y)U_{i,j}^{n} + \lambda_y(U_{i,j+1}^{n} + U_{i,j-1}^{n})$$

という差分方程式のままでよい。

- $i=0$ の場合は、$\dfrac{\partial u}{\partial n}(a,y_j) = \beta(a,y_j)$ の近似として

$$-\frac{U_{1,j}^{n+1/2} - U_{-1,j}^{n+1/2}}{2h_x} = \beta(a,y_j)$$

を採用する。これから得られる

$$U_{-1,j}^{n+1/2} = U_{1,j}^{n+1/2} + 2h_x\beta(a,y_j)$$

を用いて、差分方程式から $U_{-1,j}^{n+1/2}$ を消去する。例えば $1 \le j \le N_y-1$ の場合、

$$(1+2\lambda_x)U_{0,j}^{n+1/2} - 2\lambda_x(U_{1,j}^{n+1/2} + h_x\beta(a,y_j)) = (1-2\lambda_y)U_{0,j}^{n} + \lambda_y(U_{0,j+1}^{n} + U_{0,j-1}^{n})$$

を整理して

$$(1+2\lambda_x)U_{0,j}^{n+1/2} - 2\lambda_x U_{1,j}^{n+1/2} = (1-2\lambda_y)U_{0,j}^{n} + \lambda_y(U_{0,j+1}^{n} + U_{0,j-1}^{n}) + 2\lambda_x h_x\beta(a,y_j).$$

- 同様に $i=N_x$ の場合は、$\dfrac{\partial u}{\partial n}(b,y_j) = \beta(b,y_j)$ の近似として

$$\frac{U_{N_x+1,j}^{n+1/2} - U_{N_x-1,j}^{n+1/2}}{2h_x} = \beta(b,y_j)$$

を採用する。これから得られる

$$U_{N_x+1,j}^{n+1/2} = U_{N_x-1,j}^{n+1/2} + 2h_x\beta(b,y_j)$$

を用いて、差分方程式から $U_{N_x+1,j}^{n+1/2}$ を消去する。例えば $1 \le j \le N_y-1$ の場合、

$$(1+2\lambda_x)U_{N_x,j}^{n+1/2} - 2\lambda_x(U_{N_x-1,j}^{n+1/2} + h_x\beta(b,y_j)) = (1-2\lambda_y)U_{N_x,j}^{n} + \lambda_y(U_{N_x,j+1}^{n} + U_{N_x,j-1}^{n})$$

を整理して

$$(1+2\lambda_x)U_{N_x,j}^{n+1/2} - 2\lambda_x U_{N_x-1,j}^{n+1/2} = (1-2\lambda_y)U_{N_x,j}^{n} + \lambda_y(U_{N_x,j+1}^{n} + U_{N_x,j-1}^{n}) + 2\lambda_x h_x\beta(b,y_j).$$

- $j = 0$ の場合は

$$U^n_{i,-1} = U^n_{i,1} + 2h_y\beta(x_i, c)$$

によって $U^n_{i,-1}$ を消去する。例えば $1 \le i \le N_x - 1$ の場合、

$$(1 + 2\lambda_x)U^{n+1/2}_{i,0} - \lambda_x(U^{n+1/2}_{i+1,0} + U^{n+1/2}_{i-1,0}) = (1 - 2\lambda_y)U^n_{i,0} + 2\lambda_y(U^n_{i,1} + h_y\beta(x_i, c)).$$

$i = 0,\ N_x$ の場合はそれぞれ

$$\begin{aligned}(1 + 2\lambda_x)U^{n+1/2}_{0,0} - 2\lambda_x U^{n+1/2}_{1,0} &= (1 - 2\lambda_y)U^n_{0,0} + 2\lambda_y(U^n_{0,1} + h_y\beta(a, c)) + 2\lambda_x h_x\beta(a, c),\\ (1 + 2\lambda_x)U^{n+1/2}_{N_x,0} - 2\lambda_x U^{n+1/2}_{N_x-1,0} &= (1 - 2\lambda_y)U^n_{N_x,0} + 2\lambda_y(U^n_{N_x-1,1} + h_y\beta(b, c)) + 2\lambda_x h_x\beta(b, c).\end{aligned}$$

- $j = N_y$ の場合は

$$U^n_{i,N_y+1} = U^n_{i,N_y-1} + 2h_y\beta(x_i, d)$$

によって U^n_{i,N_y+1} を消去する。例えば $1 \le i \le N_x - 1$ の場合、

$$(1 + 2\lambda_x)U^{n+1/2}_{i,N_y} - \lambda_x(U^{n+1/2}_{i+1,N_y} + U^{n+1/2}_{i-1,N_y}) = (1 - 2\lambda_y)U^n_{i,N_y} + 2\lambda_y(U^n_{i,N_y-1} + h_y\beta(x_i, d)).$$

$i = 0,\ N_x$ の場合はそれぞれ

$$\begin{aligned}(1 + 2\lambda_x)U^{n+1/2}_{0,N_y} - 2\lambda_x U^{n+1/2}_{1,N_y} &= (1 - 2\lambda_y)U^n_{0,N_y} + 2\lambda_y(U^n_{0,N_y-1} + h_y\beta(a, d)) + 2\lambda_x h_x\beta(a, d),\\ (1 + 2\lambda_x)U^{n+1/2}_{N_x,N_y} - 2\lambda_x U^{n+1/2}_{N_x-1,N_y} &= (1 - 2\lambda_y)U^n_{N_x,N_y} + 2\lambda_y(U^n_{N_x-1,N_y-1} + h_y\beta(b, d)) + 2\lambda_x h_x\beta(b, d).\end{aligned}$$

3.7 Kronecker 積の利用

本来、この文書にマージすべきであるが、桂田 [16], [18] が参考になるかもしれない。

第4章　差分法の安定性と収束性

「応用数学」の『発展系の数値解析』では、対象が熱伝導方程式であるという特性を利用して、離散最大値原理をもとにして安定性を論じた。この章では、より一般の場合にも適用可能な形で、安定性について論じる。

(2004 年現在、

「『発展系の数値解析』に加えること」
https://m-katsurada.sakura.ne.jp/labo/text/heat-fdm-0-add.pdf

を書いている途中である。行列法については、まずそちらを読んで欲しい。かなり改良されているところがある。

ADI 法、θ 法の安定性の解析を書き加えた (2007 年 3 月下旬)。おもちゃ箱をぶちまけたような全体を整理したいものだ…)

4.1　概論

4.1.1　二つの解析手法

熱伝導方程式のような線形偏微分方程式の問題を差分法を用いて離散化すると、線形差分方程式が得られる。

『発展系の数値解析』では、熱方程式を差分化して得られた差分方程式が離散最大値原理を満たすという「特殊事情」を活用して、差分解の厳密解への収束や安定性を示した。ここでは差分方程式の安定性解析に適用できる、より一般的な手法を紹介しよう。

一般に線形差分方程式の安定性の解析には、次の二つの方法が用いられる。

- 差分方程式を行列、ベクトルを用いて表し、行列の固有値に関する議論に持ち込む (行列法)
- 有限 Fourier 級数を用いる (von Neumann の方法)

「Fourier 級数を用いる方法は簡単だが、境界条件を無視するために、厳密さを欠く。」と述べている本もある。正確には、きちんと境界条件を考慮して考察するのが難しかったり面倒だったりするので、ずぼらにやっている人が多いということではないか？ (きちんとやろうと思えば出来る場合があるのでは) と疑っている。

(2024 年夏加筆) 区間における熱方程式は、(全周 Dirichlet とか全周 Neumann とか) 固有関数が具体的に求められたりするので、Fourier 級数を用いた解の公式があり、よく紹介されるわけだが、差分方程式についても、固有ベクトルが求められるならば、同様の (差分) 解の公式が出来る。安定性の問題に説明が傾きすぎて、そういうことにあまり意識が行っていない気がする。安定性条件が満たされない場合の差分解がジグザグする理由については、以前から気づいていて、ほのめかして書いたりしていたけれど、差分解の公式を書いてしまえば「この項が成長してくるのだ」と明快に

説明できる。安定とは正の固有値が存在しないこと、と言ってみたり。von Neumann の安定性解析って要するにそういうことだよね。差分解の厳密解への収束証明とかもストレートに見えるかも。離散 Fourier 変換との関係とか、そういう話もあるかもしれない (周期境界条件の場合に J は巡回行列になる。そうでない場合は巡回行列にならなかったりするけれど、何か近いことが言えたりするのか？ さすがに無理かな。でも選点直交性があったりするし…)。公式があると考えると、ずっと先のことを知りたい場合に、差分法で延々と計算したりする必要はないのだな (そういう考えでプログラム書いてみるのかな)。「なんだ結局 Fourier 級数か」と言われると悲しい気もする。そういうことを考えると「熱方程式以外にも使える。差分法はそっちが本分だ。」を強調すべきなのだろう。

4.1.2 スキームの概念

上では「差分方程式の安定性」という書き方をしたのだが、安定性は一つの差分方程式について考えるよりも、ある条件を満す無限個の差分方程式の集合について考えるべき概念である。

> 正直に白状すると、筆者は現時点で、誰もが納得する安定性の概念というものをつかめていない。本により、人により、また同じ本・人であっても場合によって少しずつ違った定義があるような気がする。
>
> ここでは、自分で確かめるために、最初からゆっくり考える。
>
> 1. 熱方程式の解の安定性というと、
>
> $$\| \text{解} \| \leq C \| \text{初期値・境界値などのデータ} \|$$
>
> の形の不等式が成り立つこととされる (C は初期値・境界値などのデータによらない正定数)。この形の式で安定性を定義できるのは、熱方程式の特殊性がある。つまり、方程式によっては、解が時間の経過とともにいくらでも大きく成長することもあり、そのような場合は上の不等式のような評価が得ることは期待できないのである。ところで、我々は (かなり) 一般の差分方程式に対して、その安定性を論じようとしている。
> (Lax の同等性定理というのがあるから、一般的に安定性を考える必要性は高いのだ。)
>
> 2. それから、熱方程式の解の安定性と言った場合は、本質的に一つの方程式しか対象にしていないが、差分方程式の場合には、パラメーターが多い。例えば熱方程式の初期値境界値問題を離散化して得た差分方程式の場合、空間の刻み幅 h と時間の刻み幅 τ があり、既に分っているように、この値の選び方によって状況が全然異なるのであった。この場合、パラメーターのある集合については対応する差分方程式は安定、それ以外のパラメーターについては対応する差分方程式は不安定、という結果を考えたいがそれはしないようである。
>
> 3. どうも Lax の定理にひきずられて安定性の定義が難しくなっているような気がする。

4.2 ベクトルのノルム

詳しくは付録「有限次元ベクトルと行列のノルム」を参照。
以下この節では、ベクトルのノルムとして

$$\|x\|_2 := \sqrt{\sum_{j=1}^{N} |x_j|^2} \quad (x = (x_1, \cdots, x_N) \in \mathbf{C}^N)$$

で定義される $\|\cdot\|_2$ を採用する。

注意 4.2.1 離散最大値原理から導かれる安定性は

$$\|x\|_\infty := \max_{1\le j\le N} |x_j|$$

で定義されるノルム $\|\cdot\|_\infty$ に関するものであった。

行列のノルムとしては、

$$\|A\|_2 := \max_{\|x\|_2=1} \frac{\|Ax\|_2}{\|x\|_2} \quad (A \in M(N;\mathbf{C}))$$

で定められる $\|\cdot\|_2$ を用いる (ベクトルと行列のノルムの記号が同じものになっている — いわゆる記号の濫用 — 大目に見て下さい)。

漸化式

$$x_{j+1} = Ax_j \quad (j = 0, 1, 2, \cdots)$$

で定められるベクトル列 $\{x_j\}_{j=0,1,\cdots}$ は、いわば等比数列のようなものであり、基本的ではあるが、応用上煩雑に現われ、重要である (杉浦 [21] に詳しく書いてあったと記憶している)。

以下では (係数行列が実対称行列であることを利用して) Jordan 標準形を用いない証明を述べる。基礎となるのは次の有名な事実[1]である。

補題 4.2.1 (実対称行列のノルムはスペクトル半径に等しい) 実対称行列 A について、$\|A\cdot\|_2$ は A のスペクトル半径に等しい。すなわち

$$\|A\|_2 = \max\{|\lambda|; \lambda \text{ は } A \text{ の固有値 }\}.$$

証明 (この事実は普通の線形代数の本にも載っていることが多い。) ■

[1] ひょっとすると、数値計算をするものにとっては、Jordan 標準形の知識よりも重要かも知れない。

補題 4.2.2 n 次正方行列 A と、ベクトル x_0 が与えられたとき、漸化式

$$x_{j+1} = Ax_j \quad (j = 0, 1, 2, \cdots)$$

でベクトル列 $\{x_j\}_{j=0,1,\cdots}$ を定義する。

(1) A が実対称で $r(A) \leq 1$ ならば、任意の x_0 に対して

$$\|x_j\|_2 \leq \|x_0\|_2 \quad (j = 0, 1, 2, \cdots).$$

特に $\{x_j\}_{j=0,1,\cdots}$ は有界である。

(2) A が実対称でない場合は、$r(A) = 1$ であっても、適当な x_0 を取ると、$\{x_j\}_{j=0,1,\cdots}$ は非有界となることがある。

(3) $r(A) < 1$ ならば、任意の x_0 に対して

$$\lim_{j\to\infty} x_j = 0.$$

(4) $r(A) > 1$ ならば、ある x_0 に対して

$$\lim_{j\to\infty} \|x_j\|_2 = \infty.$$

証明

(1) $r(A) = \|A\|_2$ であるから、

$$\|x_j\|_2 = \|A^j x_0\|_2 \leq \|A^j\|_2 \|x_0\|_2 \leq (\|A\|_2)^j \|x_0\|_2 = r(A)^j \|x_0\|_2 \leq \|x_0\|_2.$$

(2) 例えば

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

とするとき、$r(A) = 1$ であるが、任意の $m \in \mathbf{N}$ に対して

$$A^m = \begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix}$$

であり、$x = (0, 1)^T$ について $A^m x = (m, 1)^T$ で $\|A^m x\| \to \infty$.

(3) $r(A) = \|A\|_2$ であるから、

$$\|x_j\|_2 = \|A^j x_0\|_2 \leq \|A^j\|_2 \|x_0\|_2 \leq (\|A\|_2)^j \|x_0\|_2 = r(A)^j \|x_0\|_2 \to 0 \quad (j \to \infty).$$

(実は A が実対称でない場合にも成立する。)

(4) A の固有値 λ で $|\lambda| = r(A) > 1$ なるものと、それに属する固有ベクトル u を取り、$x_0 = u$ とおくと、

$$x_j = A^j x_0 = \lambda^j u.$$

これから任意のノルムに対して

$$\|x_j\| = |\lambda|^j \|u\| \to \infty. \blacksquare$$

4.3 $\|\cdot\|_2$ ノルムに関する安定性 — 行列法

4.3.1 準備

I_n で n 次の単位行列を表わし、J_n で

$$J_n := \begin{pmatrix} 0 & 1 & 0 & & \\ 1 & 0 & \ddots & & \\ 0 & \ddots & & \ddots & \\ & & \ddots & 0 & 1 \\ & & 0 & 1 & 0 \end{pmatrix}$$

なる行列を表わす。

補題 4.3.1 J_n の固有値、固有ベクトルは

$$\begin{aligned} \mu_j &= 2\cos\left(\frac{j\pi}{n+1}\right), \\ v_j &= \left(\sin\frac{j\pi}{n+1}, \sin\frac{2j\pi}{n+1}, \cdots, \sin\frac{nj\pi}{n+1}\right) \quad (j = 1, 2, \cdots, n). \end{aligned}$$

証明 代入して確かめてみれば良い。 ■

4.3.2 1 次元熱方程式に対する陽解法の安定性

同次 Dirichlet 境界条件を課した 1 次元熱方程式の初期値境界値問題に対する陽差分法 (前進 Euler 法) を考えると、

$$\vec{U}^j := \begin{pmatrix} U_{1,j} \\ \vdots \\ U_{N-1,j} \end{pmatrix}$$

とおくと、差分方程式は

$$\vec{U}^{j+1} = [(1-2\lambda)I_{N-1} + \lambda J_{N-1}]\vec{U}^j \quad (j = 0, 1, 2, \cdots)$$

となる。

そこで

$$A_{N-1} := (1-2\lambda)I_{N-1} + \lambda J_{N-1}$$

とおくと、

$$\vec{U}^{j+1} = A_{N-1}\vec{U}^j \quad (j = 0, 1, 2, \cdots)$$

で、A_{N-1} の固有値は

$$\begin{aligned} (1-2\lambda) + \lambda\mu_j &= (1-2\lambda) + \lambda \cdot 2\cos\left(\frac{j\pi}{N}\right) = 1 - 2\lambda\left[1 - \cos\left(\frac{j\pi}{N}\right)\right] \\ &= 1 - 2\lambda \cdot 2\sin^2\left(\frac{j\pi}{2N}\right) = 1 - 4\lambda\sin^2\left(\frac{j\pi}{2N}\right) \quad (j = 1, 2, \cdots, N-1). \end{aligned}$$

さらに A_{N-1} は実対称行列だから

$$\|A_{N-1}\|_2 = \max_{j=1,\cdots,N-1} \left|1 - 4\lambda \sin^2\left(\frac{j\pi}{2N}\right)\right|.$$

ゆえに $\|A_{N-1}\|_2 \leq 1$ であるための必用十分条件は

$$\left|1 - 4\lambda \sin^2\left(\frac{j\pi}{2N}\right)\right| \leq 1 \quad (j = 1, 2, \cdots, N-1).$$

これは

$$-1 \leq 1 - 4\lambda \sin^2\left(\frac{j\pi}{2N}\right) \quad (j = 1, 2, \cdots, N-1),$$

すなわち

$$\lambda \leq \min_{1\leq j\leq N-1} \frac{1}{2\sin^2\left(\frac{j\pi}{2N}\right)} = \frac{1}{2\sin^2\left(\frac{(N-1)\pi}{2N}\right)}$$

と同値である。スキームが安定であるためには、任意の N についてこの式が成り立つことが必要十分で、それは

$$\lambda \leq \frac{1}{2}$$

と同値である。

4.3.3 1 次元熱方程式に対する θ 法の安定性

θ 法の場合の差分方程式は

$$[(1+2\theta\lambda)I_{N-1} - \theta\lambda J_{N-1}]\,\vec{U}^{j+1} = [(1-2(1-\theta)\lambda)I_{N-1} + (1-\theta)\lambda J_{N-1}]\,\vec{U}^{j} \quad (j = 0, 1, 2, \cdots).$$

ゆえに

$$B_{N-1} := [(1+2\theta\lambda)I_{N-1} - \theta\lambda J_{N-1}]^{-1}\,[(1-2(1-\theta)\lambda)I_{N-1} + (1-\theta)\lambda J_{N-1}]$$

とおけば、

$$\vec{U}^{j+1} = B_{N-1}\vec{U}^{j} \quad (j = 0, 1, 2, \cdots).$$

B_{N-1} の固有値は

$$\begin{aligned}
\frac{[1-2(1-\theta)\lambda] + (1-\theta)\lambda \cdot 2\cos\frac{j\pi}{N}}{(1+2\theta\lambda) - \theta\lambda \cdot 2\cos\frac{j\pi}{N}} &= \frac{1 - 2(1-\theta)\lambda\left(1 - \cos\dfrac{j\pi}{N}\right)}{1 + 2\theta\lambda\left(1 - \cos\dfrac{j\pi}{N}\right)} \\
&= \frac{1 - 4(1-\theta)\lambda\sin^2\dfrac{j\pi}{2N}}{1 + 4\theta\lambda\sin^2\dfrac{j\pi}{2N}} \quad (j = 1, 2, \cdots, N-1).
\end{aligned}$$

ゆえに $\|B_{N-1}\|_2 \leq 1$ であるための必用十分条件は

$$-1 \leq \frac{1 - 4(1-\theta)\lambda\sin^2\dfrac{j\pi}{2N}}{1 + 4\theta\lambda\sin^2\dfrac{j\pi}{2N}} \leq 1 \quad (j = 1, 2, \cdots, N-1).$$

$$\lambda(1-2\theta)\sin^2\frac{j\pi}{2N} \le \frac{1}{2} \quad (j=1,2,\cdots,N-1).$$

$\theta \ge 1/2$ ならば、これは常に (λ が何であっても) 満たされる。$\theta < 1/2$ の場合は、

$$\lambda \le \min_{1\le j\le N-1} \frac{1}{2(1-2\theta)\sin^2\dfrac{j\pi}{2N}} = \frac{1}{2(1-2\theta)\sin^2\dfrac{(N-1)\pi}{2N}}.$$

これが任意の N について成り立つためには

$$\lambda \le \frac{1}{2(1-2\theta)}$$

であることが必要十分。

以上をまとめると、

(i) $\theta \ge 1/2$ ならば無条件に安定。

(ii) $0 \le \theta < 1/2$ ならば、安定であるための必用十分条件は

$$\lambda \le \frac{1}{2(1-2\theta)}.$$

4.4 行列法

(前の節と内容がだぶっている。適当にまとめる、あるいは書き直すつもり。)

4.4.1 熱方程式に対する Euler 陽解法

熱方程式の陽差分法

$$u_{i,j+1} = (1-2\lambda)u_{i,j} + \lambda(u_{i-1,j} + u_{i+1,j}) \quad (i=1,2,\cdots,N-1,\ j=0,1,\cdots),$$

$$u_{0,j+1} = u_{N,j+1} = 0$$

から

$$\begin{pmatrix} U_1^{j+1} \\ U_2^{j+1} \\ \vdots \\ u_{N-2,j+1} \\ U_{N-1}^{j+1} \end{pmatrix} = \begin{pmatrix} 1-2\lambda & \lambda & & & \\ \lambda & 1-2\lambda & \lambda & & \\ & \lambda & 1-2\lambda & \lambda & \\ & & \ddots & \ddots & \ddots \\ & & & \lambda & 1-2\lambda \end{pmatrix} \begin{pmatrix} u_{1,j} \\ u_{2,j} \\ \vdots \\ u_{N-2,j} \\ u_{N-1,j} \end{pmatrix}.$$

この方程式を

$$\boldsymbol{u}_{j+1} = A\boldsymbol{u}_j$$

と書こう。つまり

$$\boldsymbol{u}_j = \begin{pmatrix} U_1^{j+1} \\ U_2^{j+1} \\ \vdots \\ u_{N-2,j+1} \\ U_{N-1}^{j+1} \end{pmatrix} \quad (j=0,1,\cdots), \quad A = \begin{pmatrix} 1-2\lambda & \lambda & & & \\ \lambda & 1-2\lambda & \lambda & & \\ & \lambda & 1-2\lambda & \lambda & \\ & & \ddots & \ddots & \ddots \\ & & & \lambda & 1-2\lambda \end{pmatrix}$$

とおくわけである。

$N-1$ 次正方行列 T を

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

で定めると

$$A=I+\lambda T$$

である。

T の固有値、固有ベクトルは次のようになる。

$$\mu_i=-4\sin^2\left(\frac{i\pi}{2N}\right),\quad \boldsymbol{v}_i=\begin{pmatrix} \sin\frac{i\pi}{N} \\ \sin\frac{2i\pi}{N} \\ \vdots \\ \sin\frac{(N-1)i\pi}{N} \end{pmatrix}\quad (i=1,2,\cdots,N-1).$$

ところで、$f(x)=1+\lambda x$ とおくと、$A=f(T)$ であるが、

スペクトル写像定理[2]

$f(T)$ の固有値は $f(\lambda)$ (λ は T の固有値).

から、A の固有値は

$$f(\mu_i)=1+\lambda\left[-4\sin^2\left(\frac{i\pi}{2N}\right)\right]\quad (i=1,2,\cdots,N-1).$$

したがって、安定性の条件は

$$\left|1+\lambda\left[-4\sin^2\left(\frac{i\pi}{2N}\right)\right]\right|\le 1\quad (i=1,2,\cdots,N-1).$$

これは

$$-1\le 1+\lambda\left[-4\sin^2\left(\frac{i\pi}{2N}\right)\right]\le 1\quad (i=1,2,\cdots,N-1)$$

ということであり、

$$\lambda\le\frac{1}{2\sin^2\left(\frac{i\pi}{2N}\right)}\quad (i=1,2,\cdots,N-1).$$

これから

$$\lambda\le\frac{1}{2}$$

であれば安定であることが分かる。

4.4.2 熱方程式に対する θ 法

$$\begin{pmatrix} 1+2\theta\lambda & -\theta\lambda & & & \\ -\theta\lambda & 1+2\theta\lambda & -\theta\lambda & & \\ & -\theta\lambda & 1+2\theta\lambda & -\theta\lambda & \\ & & \ddots & \ddots & \ddots \\ & & & -\theta\lambda & 1+2\theta\lambda \end{pmatrix} \begin{pmatrix} U_1^{j+1} \\ U_2^{j+1} \\ \vdots \\ u_{N-2,j+1} \\ U_{N-1}^{j+1} \end{pmatrix}$$

$$= \begin{pmatrix} 1-2(1-\theta)\lambda & (1-\theta)\lambda & & & \\ (1-\theta)\lambda & 1-2(1-\theta)\lambda & (1-\theta)\lambda & & \\ & (1-\theta)\lambda & 1-2(1-\theta)\lambda & (1-\theta)\lambda & \\ & & \ddots & \ddots & \ddots \\ & & & (1-\theta)\lambda & 1-2(1-\theta)\lambda \end{pmatrix} \begin{pmatrix} u_{1,j} \\ u_{2,j} \\ \vdots \\ u_{N-2,j} \\ u_{N-1,j} \end{pmatrix}.$$

$$(I-\theta\lambda T)\boldsymbol{u}_{j+1} = (I+(1-\theta)\lambda T)\boldsymbol{u}_j.$$

ゆえに

$$\boldsymbol{u}_{j+1} = (I-\theta\lambda T)^{-1}(I+(1-\theta)\lambda T)\boldsymbol{u}_j.$$

今度は $g(x) = (1+(1-\theta)\lambda x)/(1-\theta\lambda x)$ とおくと、

$$(I-\theta\lambda T)^{-1}(I+(1-\theta)\lambda T) = g(T).$$

ゆえに安定性の条件は

$$|g(\mu_i)| \leq 1 \quad (i = 1, 2, \cdots, N-1).$$

これから、

- $0 \leq \theta < 1/2$ の場合、

$$\lambda \leq \frac{1}{2(1-2\theta)}$$

であれば安定である。

- $1/2 \leq \theta \leq 1$ の場合、無条件に安定である。

4.4.3 備考

N 次正方行列

$$\begin{pmatrix} a & b & & & \\ c & a & b & & \\ & c & a & b & \\ & & \ddots & \ddots & \ddots \\ & & & c & a \end{pmatrix}$$

の固有値は

$$\mu_i = a + 2\sqrt{bc}\cos\left(\frac{i\pi}{N+1}\right) \quad (i = 1, 2, \cdots, N).$$

4.5 von Neumann の安定性解析

(時間がなくて確かめられないのだが、この節の記述は抜本的に改良できるのでは？ と感じている。

『波動方程式に対する差分法』
https://m-katsurada.sakura.ne.jp/labo/text/wave.pdf

の 1 次元波動方程式の初期値境界値問題に対する差分法の解析で使った考え方が有効だと想像している。)

von Neumann 法、あるいは Fourier 級数法と呼ばれる安定性解析手法について述べる。

前節の行列法は、行列の固有値の解析が困難な場合も多いため、応用家にはこちらの方法の方が人気があるらしい。ただし、いつもきちんと解析できるとは限らず、大ざっぱな目安を得て満足するというものらしい。正直に言って、これを書いている時点で、筆者はあまり良く理解できていない。

Smith [22] や田端 [23] に色々書いてある。

歴史的には、von Neumann によって開発され、O'Brien, Hyman, Kaplan[24] で発表された、とある。この方法は、厳密には、定数係数線形差分方程式の初期値問題にしか適用できない。

これは、初期値に誤差が含まれているときに、その誤差を Fourier 級数で表し、これが時間の経過と共にどうなるか (増幅されるのか、減衰するのか) を Fourier の変数分離法で調べる、というのが基本的なアイディアである。関数の定義域を $[0,\ell]$ として、誤差関数 $E(x,t)$ が

$$E(x,0)=\sum_n A_n e^{\sqrt{-1}n\pi x/\ell}$$

を満たすとしよう。問題の線形性から、もしも各項

$$e^{\sqrt{-1}n\pi x/\ell}$$

を初期値とした場合に時刻 t における状態が分かれば、$E(x,t)$ は完全に決定される。

$$E_{p,q}=e^{\sqrt{-1}\beta x}e^{\sqrt{-1}\alpha t}|_{x=ph,t=q\tau}=e^{\sqrt{-1}\beta ph}e^{\sqrt{-1}\alpha q\tau}=\xi^q e^{\sqrt{-1}\beta ph}.$$

ただし、$\xi=e^{\alpha\tau}$ である。

$$E(x,t)=\sum_n A_n B_n(t)e^{\sqrt{-1}n\pi x/\ell}$$

であることが

$h=\ell/N$ とおき、$[0,\ell]$ の N 等分点 ph $(p=0,1,\cdots,N)$ における誤差 $E(ph,0)$ を E_p で表す。

$$E_p=\sum_{n=0}^{N}A_n e^{\sqrt{-1}\beta_n ph}\quad(p=0,1,\cdots,N)$$

は $\{A_n\}_{n=0,1,\cdots,N}$ を決定するのに十分である。

1 次元熱方程式に対する完全陰公式　熱方程式 $u_t = u_{xx}$ の完全陰差分方程式

$$\frac{u_{p,q+1} - u_{p,q}}{\tau} = \frac{u_{p+1,q+1} - 2u_{p,q+1} + u_{p-1,q+1}}{h^2}$$

の安定性を調べる。誤差関数 $E_{p,q}$ は、差分解 $u_{p,q}$ と同じ差分方程式を満たすので、

$$E_{p,q} = \xi^q e^{\sqrt{-1}\beta ph}$$

を に代入すると

$$e^{\sqrt{-1}\beta ph}\xi^{q+1} = \lambda(e^{\sqrt{-1}\beta(p-1)h}\xi^{q+1} - 2e^{\sqrt{-1}\beta ph}\xi^{q+1} + e^{\sqrt{-1}\beta(p+1)h}\xi^{q+1}).$$

これを $e^{\sqrt{-1}\beta ph}\xi^q$ で割れば

$$\xi - 1 = \lambda\xi(e^{\sqrt{-1}\beta h} - 2 + e^{-\sqrt{-1}\beta h}) = \lambda\xi(2\cos\beta h - 2) = -4\lambda\xi\sin^2(\beta h/2).$$

ゆえに

$$\xi = \frac{1}{1 + 4\lambda\xi\sin^2(\beta h/2)}.$$

これから明らかに $|\xi| \leq 1$ であり、λ の値によらず、安定であることが分かる。 ■

1 次元波動方程式に対する陽公式　波動方程式 $u_{tt} = u_{xx}$ に対する陽公式

$$\frac{u_{p,q+1} - 2u_{p,q} + u_{p,q-1}}{\tau^2} = \frac{u_{p+1,q} - 2u_{p,q} + u_{p-1,q}}{h^2}$$

の安定性を考える。

$u_{p,q} = e^{\sqrt{-1}\beta ph}\xi^q$ を代入すると、

$$\xi - 2 + \xi^{-1} = \lambda^2(e^{\sqrt{-1}\beta h} - 2 + e^{-\sqrt{-1}\beta h}).$$

ゆえに

$$\xi - 2 + \xi^{-1} = \lambda^2(2\cos\beta h - 2) = -4\lambda^2\sin^2(\beta h/2).$$

これから

$$\xi^2 - 2A\xi + 1 = 0, \quad A = 1 - 2\lambda^2\sin^2(\beta h/2).$$

2 次方程式の根の公式から

$$\xi = A \pm \sqrt{A^2 - 1}$$

であるが、

(1) $A^2 > 1$ の場合は、$A > 1$ または $A < -1$ であるが、

(a) $A > 1$ の場合 $A + \sqrt{A^2 - 1} > A > 1$ ゆえ $\xi > 1$.

(b) $A < -1$ の場合 $A - \sqrt{A^2 - 1} < A < -1$ ゆえ $\xi < -1$.

となるので、$|\xi| > 1$ となり、安定性の条件は満たされない。

(2) $A^2 \leq 1$ の場合は、$\xi = A \pm \sqrt{1 - A^2}\sqrt{-1}$ であり、

$$|\xi| = \sqrt{A^2 + (1 - A^2)} = 1.$$

ゆえに、安定性の条件は満たされる。

まとめると、

$$\text{安定} \iff |A| \le 1.$$

ゆえに

$$-1 \le 1 - 2\lambda^2 \sin^2(\beta h/2) \le 1.$$

これから

$$\lambda^2 \le \frac{1}{\sin^2(\beta h/2)}.$$

β の任意性から $\lambda^2 \le 1$ が導かれる。 ■

4.6 ADI 法の安定性の解析

「ADI 法は無条件安定」と良く言われているが、筆者は証明を見かけたことはない。拙い証明を以下に示す。

長方形領域 $\Omega := (0, W) \times (0, H)$ における熱方程式の初期値境界値問題

$$u_{tt}(x, y, t) = \triangle u(x, y, t) \quad ((x, y, t) \in \Omega \times (0, \infty)), \tag{4.1}$$

$$u(x, y, t) = 0 \quad ((x, y) \in \Gamma := \partial\Omega,\ t \in (0, \infty)), \tag{4.2}$$

$$u(x, y, 0) = u_0(x, y) \quad ((x, y) \in \overline{\Omega}) \tag{4.3}$$

に対する ADI 法の差分方程式は、

$$\frac{U_{ij}^{n+1/2} - U_{ij}^{n}}{\tau/2} = \frac{U_{i+1,j}^{n+1/2} - 2U_{ij}^{n+1/2} + U_{i-1,j}^{n+1/2}}{h_x^2} + \frac{U_{i,j+1}^{n} - 2U_{ij}^{n} + U_{i,j-1}^{n}}{h_y^2}, \tag{4.4}$$

$$\frac{U_{ij}^{n+1} - U_{ij}^{n+1/2}}{\tau/2} = \frac{U_{i+1,j}^{n+1/2} - 2U_{ij}^{n+1/2} + U_{i-1,j}^{n+1/2}}{h_x^2} + \frac{U_{i,j+1}^{n+1} - 2U_{ij}^{n+1} + U_{i,j-1}^{n+1}}{h_y^2}, \tag{4.5}$$

$$U_{i,j}^{\ell} = 0 \quad (i = 0 \text{ or } i = N_x \text{ or } j = 0 \text{ or } j = N_y;\ \ell = 1/2, 1, 3/2, 2, 5/2, \dots), \tag{4.6}$$

$$U_{ij}^{0} = u_0(x_i, y_j) \quad (i = 0, 1, \dots, N_x;\ j = 0, 1, \dots, N_y) \tag{4.7}$$

である。

定理 4.6.1 (ADI 法の無条件安定性) $\{U_{ij}^{\ell}\}$ が (4.4), (4.5), (4.6) を満たすならば、任意の $n \in \mathbf{N}_0$ に対して、

$$\sum_{i=1}^{N_x-1} \sum_{j=1}^{N_y-1} \left|U_{ij}^{n+1}\right|^2 \le \sum_{i=1}^{N_x-1} \sum_{j=1}^{N_y-1} \left|U_{ij}^{n}\right|^2 \tag{4.8}$$

が成り立つ。

(4.4), (4.5) は、

$$\lambda_x := \frac{\tau/2}{h_x^2}, \quad \lambda_y := \frac{\tau/2}{h_y^2}$$

とおくと[3]、

$$(1+2\lambda_x)U^{n+1/2}ij-\lambda_x(U^{n+1/2}_{i+1,j}+U^{n+1/2}_{i-1,j})=(1-2\lambda_y)U^nij+\lambda_y(U^n_{i,j+1}+U^n_{i,j-1}), \tag{4.9}$$

$$(1+2\lambda_y)U^{n+1}ij-\lambda_y(U^{n+1}_{i,j+1}+U^{n+1}_{i,j-1})=(1-2\lambda_x)U^{n+1/2}ij+\lambda_x(U^{n+1/2}_{i+1,j}+U^{n+1/2}_{i-1,j}) \tag{4.10}$$

と書き直される。

式を簡単にするため、

$$p:=N_x-1,\quad q:=N_y-1,\quad N:=pq$$

とおき、

$$U^\ell_{ij}=U^\ell_{p(j-1)+i}$$

によって U^ℓ_k $(k=1,2,\dots,N)$ を定め、

$$\boldsymbol{U}^\ell:=(U^\ell_1,U^\ell_2,\cdots,U^\ell_N)^T\quad(\ell=0,\frac{1}{2},1,\frac{3}{2},\dots)$$

というベクトルを定義すると、(4.9), (4.10) は

$$\{I_q\otimes[(1+2\lambda_x)I_p-\lambda_xJ_p]\}\,\boldsymbol{U}^{n+1/2}=\{[(1-2\lambda_y)I_q+\lambda_yJ_q]\otimes I_p\}\,\boldsymbol{U}^n, \tag{4.11}$$

$$\{[(1+2\lambda_y)I_q-\lambda_yJ_q]\otimes I_p\}\,\boldsymbol{U}^{n+1}=\{I_q\otimes[(1-2\lambda_x)I_p+\lambda_xJ_p]\}\,\boldsymbol{U}^{n+1/2} \tag{4.12}$$

と書き直される。ここで $\otimes$ は行列の Kronecker 積を表す:

$$A\otimes B=\begin{pmatrix}a_{11}B & a_{12}B & \cdots & a_{1n}B\\ a_{21}B & a_{22}B & \cdots & a_{2n}B\\ \vdots & & & \vdots\\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B\end{pmatrix}\quad(A=(a_{ij})\in\mathbf{R}^{m\times n},\quad B\in\mathbf{R}^{k\times\ell}).$$

ゆえに

$$\begin{aligned}\boldsymbol{U}^{n+1}=&\{[(1+2\lambda_y)I_q-\lambda_yJ_q]\otimes I_p\}^{-1}\{I_q\otimes[(1-2\lambda_x)I_p+\lambda_xJ_p]\}\\&\times\{I_q\otimes[(1+2\lambda_x)I_p-\lambda_xJ_p]\}^{-1}\{[(1-2\lambda_y)I_q+\lambda_yJ_q]\otimes I_p\}\,\boldsymbol{U}^n.\end{aligned} \tag{4.13}$$

Kronecker 積の性質を二三紹介する。

命題 4.6.2 $(A\otimes B)^T=A^T\otimes B^T$, $(A\otimes B)^*=A^*\otimes B^*$. 特に実対称行列の Kronecker 積は実対称行列で、ユニタリー行列の Kronecker 積はユニタリー行列である。

証明 (略) ■

命題 4.6.3 $A_1,\,A_2\in\mathbf{R}^{q\times q}$ かつ $B_1,\,B_2\in\mathbf{R}^{p\times p}$ とするとき、

$$(A_1\otimes B_1)(A_2\otimes B_2)=(A_1A_2)\otimes(B_1B_2).$$

証明 (略) ■

[3]後で θ 法と比較するためには、$\lambda_x=\tau/h_x^2$, $\lambda_y=\tau/h_y^2$ とおく方がよいかもしれないが、次の (4.9), (4.10) で分数が出るのを嫌った。

これから、もちろん任意個数の積について

$$(A_1 \otimes B_1)(A_2 \otimes B_2)\cdots(A_r \otimes B_r) = (A_1 A_2 \cdots A_r) \otimes (B_1 B_2 \cdots B_r) \tag{4.14}$$

が成り立つことが分かる。また $A \in GL(q;\mathbf{R})$, $B \in GL(p;\mathbf{R})$ とするとき、

$$A \otimes B \in GL(qp;\mathbf{R}), \quad (A \otimes B)^{-1} = A^{-1} \otimes B^{-1}. \tag{4.15}$$

$r = 4$ についての (4.14) と、(4.15) を用いて (4.13) を変形すると

$$\begin{aligned}\boldsymbol{U}^{n+1} = &\left\{[(1+2\lambda_y)I_q - \lambda_y J_q]^{-1}\,[(1-2\lambda_y)I_q + \lambda_y J_q]\right\} \\ &\otimes \left\{[(1-2\lambda_x)I_p + \lambda_x J_p]\,[(1+2\lambda_x)I_p - \lambda_x J_p]^{-1}\right\}\boldsymbol{U}^n.\end{aligned}$$

記述を簡潔にするため、

$$\begin{aligned}A_{q,\lambda_y} &:= [(1+2\lambda_y)I_q - \lambda_y J_q]^{-1}\,[(1-2\lambda_y)I_q + \lambda_y J_q], \\ B_{p,\lambda_x} &:= [(1-2\lambda_x)I_p + \lambda_x J_p]\,[(1+2\lambda_x)I_p - \lambda_x J_p]^{-1}\end{aligned}$$

とおくと、

$$\boldsymbol{U}^{n+1} = \left[A_{q,\lambda_y} \otimes B_{p,\lambda_x}\right]\boldsymbol{U}^n. \tag{4.16}$$

以下 A_{q,λ_y}, B_{p,λ_x} の固有値について調べる。

補題 4.6.4 $N \in \mathbf{N}$, $N \geq 3$ とするとき、

$$J := \begin{pmatrix} 0 & 1 & & & \\ 1 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 0 & 1 \\ & & & 1 & 0 \end{pmatrix} \in \mathbf{R}^{(N-1)\times(N-1)},$$

$$h := 1/N, \quad x_j := jh \quad (j = 1, 2, \ldots, N-1),$$

$$\lambda_n := 2\cos n\pi h, \quad \boldsymbol{v}_n := \begin{pmatrix} \sin n\pi x_1 \\ \sin n\pi x_2 \\ \vdots \\ \sin n\pi x_{N-1} \end{pmatrix} \quad (n = 1, 2, \ldots, N-1)$$

とおくと、

$$-2 < \lambda_1 < \lambda_2 < \cdots < \lambda_{N-1} < 2,$$

$$J\boldsymbol{v}_n = \lambda_n \boldsymbol{v}_n \quad (n = 1, 2, \ldots, N-1)$$

が成り立つ。すなわち $\{\lambda_n; n = 1, 2, \ldots, N-1\}$ は J の固有値全体で、$\boldsymbol{v}_n$ は J の固有値 λ_n に属する固有ベクトルである。

証明　これは 1 次元熱方程式に対する差分法を調べるときに登場する。『発展系の数値解析の続き』(https://m-katsurada.sakura.ne.jp/labo/text/heat-fdm-0-add.pdf) に書いておいた。

■

命題 4.6.5 $\lambda > 0$, $r \in \mathbf{N}$, $r \geq 2$ とするとき、

$$A := [(1+2\lambda)I_r - \lambda J_r]^{-1}[(1-2\lambda)I_r + \lambda J_r] = [(1-2\lambda)I_r + \lambda J_r][(1+2\lambda)I_r - \lambda J_r]^{-1}$$

の固有値は $(-1, 1)$ に含まれる。

証明　上の補題から J_r の固有値は $(-2, 2)$ に含まれることが分かる。

$$f(x) := \frac{(1-2\lambda) + \lambda x}{(1+2\lambda) - \lambda x} \quad (x \in [-2, 2])$$

とおくと (分母が 0 となるのは $x = (1+2\lambda)/\lambda = 2 + \dfrac{1}{\lambda}$ のときであるから、$\lambda > 0$, $x \in [-2, 2]$ のとき、分母は 0 にならない)、$A = f(J_r)$ である。

$$f'(x) = \frac{2\lambda}{[(1+2\lambda) - \lambda x]^2} > 0$$

であるから、f は狭義単調増加で、

$$f(-2) = \frac{1-4\lambda}{1+4\lambda} = -1 + \frac{2}{1+4\lambda} > -1, \quad f(2) = 1$$

であるから、$f((-2, 2)) \subset (-1, 1)$. Frobenius の定理 (それは普通行列の多項式の話で、ここでは行列の有理式だから、「スペクトル写像定理[4]」というべき？) から、A の固有値は $(-1, 1)$ に含まれる。 ■

…手元の線形代数のテキストを何冊か見てみたが、「フロベニウスの定理」が載っていないものが多いのには少々驚いた (先日放送大学の授業を視ていても出て来たし、そんなにマイナーな定理ではないと思うのだが…)。Dunford 積分とか、自己共役作用素のスペクトル分解とやると「やりすぎ」気味なので、ここでは次の形のささやかな命題を証明つきで与えておく。

命題 4.6.6 (有理式版フロベニウスの定理) A は n 次正方行列で、$\lambda_1, \ldots, \lambda_n$ が固有値 (i.e., $\det(xI - A) = \prod_{i=1}^{n}(x - \lambda_i)$) で、

$$f(x) = \frac{q(x)}{p(x)} \quad (p(x),\, q(x) \text{ は多項式})$$

であり、$p(\lambda_i) \neq 0$ $(i = 1, \ldots, n)$ とする。このとき $p(A)$ は正則で、$p(A)^{-1}q(A)$ $(= q(A)p(A)^{-1})$ の固有値は $f(\lambda_1), \ldots, f(\lambda_n)$ である。

証明　適当な unitary 行列 U を取ると、

$$U^*AU = \begin{pmatrix} \lambda_1 & & * \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix}$$

[4] A の固有値全体を $\sigma(A)$ と書くとき、スペクトル写像定理とは、集合についての等式

$$f(\sigma(A)) = \sigma(f(A))$$

のことを言う。

となる (A の Schur 分解)。これから

$$U^*p(A)U = \begin{pmatrix} p(\lambda_1) & & * \\ & \ddots & \\ 0 & & p(\lambda_n) \end{pmatrix}, \quad U^*q(A)U = \begin{pmatrix} q(\lambda_1) & & *' \\ & \ddots & \\ 0 & & q(\lambda_n) \end{pmatrix}.$$

第 1 式の逆行列を取ると

$$U^*p(A)^{-1}U = \begin{pmatrix} p(\lambda_1)^{-1} & & *'' \\ & \ddots & \\ 0 & & p(\lambda_n)^{-1} \end{pmatrix}.$$

(左辺については、$(ABC)^{-1} = C^{-1}B^{-1}A^{-1}$ と、$U^* = U^{-1}$ を用いた。右辺については、“上三角行列の逆行列は上三角行列” という定理と、仮定 $p(\lambda_i) \neq 0$ を用いた。)

ゆえに

$$U^*p(A)^{-1}q(A)U = U^*p(A)^{-1}UU^*q(A)U = \begin{pmatrix} p(\lambda_1)^{-1}q(\lambda_1) & & ** \\ & \ddots & \\ 0 & & p(\lambda_n)^{-1}q(\lambda_n) \end{pmatrix}$$

$$= \begin{pmatrix} f(\lambda_1) & & ** \\ & \ddots & \\ 0 & & f(\lambda_n) \end{pmatrix}.$$

これから $p(A)^{-1}q(A)$ の固有値は $f(\lambda_1), \ldots, f(\lambda_n)$ である。 ■

注意 4.6.7 1 次元熱方程式の初期値境界値問題

$$\begin{aligned} &u_t(x,t) = u_{xx}(x,t) \quad (x \in (0,1),\, t \in (0,\infty)), \\ &u(0,t) = u(1,t) = 0 \quad (t \in (0,\infty)), \\ &u(x,0) = f(x) \quad (x \in [0,1]) \end{aligned}$$

に対する θ 法の差分方程式は

$$[(1+2\theta\lambda)I_{N-1} - \theta\lambda J_{N-1}]\boldsymbol{U}^{n+1} = \{[1-2(1-\theta)\lambda]I_{N-1} + (1-\theta)\lambda J_{N-1}\}\,\boldsymbol{U}^n$$

である。ゆえに

$$\boldsymbol{U}^{n+1} = [(1+2\theta\lambda)I_{N-1} - \theta\lambda J_{N-1}]^{-1}\,\{[1-2(1-\theta)\lambda]I_{N-1} + (1-\theta)\lambda J_{N-1}\}\,\boldsymbol{U}^n.$$

いわゆる Crank-Nicolson 法、すなわち $\theta = 1/2$ の場合は

$$\boldsymbol{U}^{n+1} = \left[(1+\lambda)I_{N-1} - \frac{\lambda}{2}J_{N-1}\right]^{-1}\left\{[(1-\lambda)I_{N-1} + \frac{\lambda}{2}J_{N-1}\right\}\boldsymbol{U}^n.$$

$\lambda = 2\lambda'$ とおくと、

$$\boldsymbol{U}^{n+1} = [(1+2\lambda')I_{N-1} - \lambda' J_{N-1}]^{-1}\,\{[(1-2\lambda')I_{N-1} + \lambda' J_{N-1}\}\,\boldsymbol{U}^n.$$

つまり、命題 4.6.5 の行列は、1 次元熱方程式を Crank-Nicolson 法で解く場合に現れる行列と同じである！ ■

さて、ゴールの仕方は色々あるが、まずは、次の命題を使ってやっつける。

命題 4.6.8 $A \in \mathbf{C}^{m\times m}$ の固有値を λ_i $(i = 1, 2, \ldots, m)$, $B \in \mathbf{C}^{n\times n}$ の固有値を μ_j $(j = 1, 2, \ldots, n)$ とするとき、$A \otimes B$ の固有値は $\lambda_i\mu_j$ $(i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n)$.

証明　(常識的な結果なので、「証明略」としてもよいかもしれないが、証明を示すと、実対称行列の場合に固有ベクトルがどうなるかが見えて、後々役に立ちそうなので、さぼらずに書いておく。)

適当なユニタリー行列 Q_1, Q_2 によって、A, B は上三角行列に変換できる (Schur 分解):

$$Q_1^*AQ_1 = \begin{pmatrix} \lambda_1 & & & * \\ & \lambda_2 & & \\ & & \ddots & \\ 0 & & & \lambda_m \end{pmatrix}, \quad Q_2^*BQ_2 = \begin{pmatrix} \mu_1 & & & * \\ & \mu_2 & & \\ & & \ddots & \\ 0 & & & \mu_n \end{pmatrix}.$$

この右辺をそれぞれ R_1, R_2 とおくと、(4.14) を用いて

$$R_1 \otimes R_2 = (Q_1^*AQ_1) \otimes (Q_2^*BQ_2) = (Q_1^* \otimes Q_2^*)(A \otimes B)(Q_1 \otimes Q_2).$$

一般に $(A \otimes B)^* = A^* \otimes B^*$ が成り立つので、

$$R_1 \otimes R_2 = (Q_1 \otimes Q_2)^*(A \otimes B)(Q_1 \otimes Q_2).$$

$Q := Q_1 \otimes Q_2$ とおくと、$R_1 \otimes R_2 = Q^*(A \otimes B)Q$ であるが、実は Q はユニタリー行列である。実際

$$Q^*Q = (Q_1 \otimes Q_2)^*(Q_1 \otimes Q_2) = (Q_1^* \otimes Q_2^*)(Q_1 \otimes Q_2) = (Q_1^*Q_1) \otimes (Q_2^*Q_2) = I_m \otimes I_n = I_{mn}.$$

$R_1 \otimes R_2$ は上三角行列であるから、$A \otimes B$ の固有値はその対角成分に等しく、$\lambda_i\mu_j$ である。 ■

従って、(4.16) における A_{q,λ_y}, B_{p,λ_x} の固有値はいずれも $(-1, 1)$ に属するので、$A_{q,\lambda_y} \otimes B_{p,\lambda_x}$ の固有値も (それらの積であるから) $(-1, 1)$ に属する。

次は常識であるが、念のため。

命題 4.6.9 実対称行列 A に対して、ベクトルのノルムとして普通の Euclid ノルム $\|\cdot\|_2$ を採ったときの作用素ノルム $\|A\|$ は、その行列のスペクトル半径 $r(A)$ に等しい:

$$\|A\| = r(A).$$

念のため、出て来た言葉の定義を式で示しておくと、

$$\|A\| := \sup_{x\neq 0} \frac{\|Ax\|_2}{\|x\|_2} \quad (A \in \mathbf{R}^{r\times r}), \quad \|y\|_2 := \sqrt{\sum_{j=1}^{r} y_j^2} \quad (y = (y_j) \in \mathbf{R}^r),$$

$$r(A) := \max\{|\lambda|; \lambda \in \sigma(A)\}, \quad \sigma(A) := A \text{ の固有値全体の集合.}$$

ゆえに

$$\|A_{q,\lambda_y} \otimes B_{p,\lambda_x}\| = r(A_{q,\lambda_y} \otimes B_{p,\lambda_x}) < 1.$$

従って、

$$\|\boldsymbol{U}^{n+1}\|_2 \leq \|\boldsymbol{U}^n\|_2.$$

これは (4.8) に他ならない。

余談 4.6.10 (MATLAB で試しながら…) この問題はもう片付いてしまったが、MATLAB や Octave で次のようにして、実際に行列のスペクトル半径が計算できて、心強い。

```
p=4
q=5
Ip=eye(p,p)
Iq=eye(q,q)
Jp=diag(ones(p-1,1),1)+diag(ones(p-1,1),-1)
Jq=diag(ones(q-1,1),1)+diag(ones(q-1,1),-1)
P=inv((1+2*lambda)*Ip-lambda*Jp)*((1-2*lambda)*Ip+lambda*Jp)
max(abs(eig(P)))
```

ここでは $[(1+2\lambda)I_p - \lambda J_p]^{-1}\,[(1-2\lambda)I_p + \lambda J_p]$ のスペクトル半径を計算している。 ■

ところで (4.13) の行列は 4 つの因子の積であるが、これらは互いに可換なので[5]、好きなように順番を入れ替えることができる。その結果を解釈することはちょっと面白い…

4.7 θ 法の安定性解析

この節を読む前に 4.6「ADI 法の安定性解析」を読むことをお勧めする。
長方形領域 $\Omega := (0,W)\times(0,H)$ における熱方程式の初期値境界値問題

$$u_{tt}(x,y,t) = \triangle u(x,y,t) \quad ((x,y,t)\in\Omega\times(0,\infty)), \tag{4.17}$$

$$u(x,y,t) = 0 \quad ((x,y)\in\Gamma := \partial\Omega,\ t\in(0,\infty)), \tag{4.18}$$

$$u(x,y,0) = u_0(x,y) \quad ((x,y)\in\overline{\Omega}) \tag{4.19}$$

に対する θ 法の差分方程式は ($\psi(i,j) = (j-1)(N_x-1)+i$ という番号づけを用いる場合)、

$$\begin{aligned} A &:= [1+2\theta(\lambda_x+\lambda_y)]\,I_q\otimes J_p - \theta\lambda_x I_q\otimes J_p - \theta\lambda_y J_q\otimes I_p,\\ B &:= [1-2(1-\theta)(\lambda_x+\lambda_y)]\,I_q\otimes J_p + (1-\theta)\lambda_x I_q\otimes J_p + (1-\theta)\lambda_y J_q\otimes I_p \end{aligned}$$

とおくとき、

$$A\boldsymbol{U}^{n+1} = B\boldsymbol{U}^n$$

と表される。ただし記述を簡潔にするため

$$p := N_x - 1, \quad q := N_y - 1, \quad N := pq$$

とおいた。

4.7.1 $\|\cdot\|_2$ ノルムに関する安定性

安定性を調べるには、

$$\begin{aligned} A^{-1}B = &\{[1+2\theta(\lambda_x+\lambda_y)]\,I_q\otimes I_p - \theta\lambda_x I_q\otimes J_p - \theta\lambda_y J_q\otimes I_p\}^{-1}\\ &\times\{[1-2(1-\theta)(\lambda_x+\lambda_y)]\,I_q\otimes I_p + (1-\theta)\lambda_x I_q\otimes J_p + (1-\theta)\lambda_y J_q\otimes I_p\} \end{aligned}$$

[5] $A = I_q\otimes J_p$, $B = J_q\otimes I_p$ とするとき、$AB = (I_q\otimes J_p)(J_q\otimes I_p) = (I_qJ_q)\otimes(J_pI_p) = J_q\otimes J_p$, $BA = (J_q\otimes I_p)(I_q\otimes J_p) = (J_qI_q)\otimes(I_pJ_p) = J_q\otimes J_p$.

の固有値を調べれば良い。

J_p, J_q の固有値、固有ベクトルは分かっている。

$$\boldsymbol{v}_i := \begin{pmatrix} \sin(i\pi/N_y) \\ \sin(2i\pi/N_y) \\ \vdots \\ \sin(qi\pi/N_y) \end{pmatrix} \in \mathbf{R}^q, \quad \lambda_i := 2\cos(i\pi/N_y) \quad (i = 1, 2, \ldots, q),$$

$$\boldsymbol{w}_j := \begin{pmatrix} \sin(j\pi/N_x) \\ \sin(2j\pi/N_x) \\ \vdots \\ \sin(pj\pi/N_x) \end{pmatrix} \in \mathbf{R}^p, \quad \mu_j := 2\cos(j\pi/N_x) \quad (j = 1, 2, \ldots, p)$$

とおくと、

$$J_q\boldsymbol{v}_i = \lambda_i\boldsymbol{v}_i \quad (i = 1, 2, \ldots, q), \quad J_p\boldsymbol{w}_j = \lambda_j\boldsymbol{w}_j \quad (j = 1, 2, \ldots, p)$$

が成り立つ。

このとき、

$$\boldsymbol{e}_{ij} := \boldsymbol{v}_i \otimes \boldsymbol{w}_j$$

とおくと、

$$(I_q \otimes J_p)\boldsymbol{e}_{ij} = \mu_j\boldsymbol{e}_{ij}, \quad (J_q \otimes I_p)\boldsymbol{e}_{ij} = \lambda_i\boldsymbol{e}_{ij} \tag{4.20}$$

となる。

念のため、用いたことを命題としてまとめておく。

命題 4.7.1 $A \in \mathbf{C}^{q\times q}$, $B \in \mathbf{C}^{p\times p}$, $\boldsymbol{v} \in \mathbf{C}^q$, $\boldsymbol{w} \in \mathbf{C}^p$, $\lambda \in \mathbf{C}$, $\mu \in \mathbf{C}$ が

$$A\boldsymbol{v} = \lambda\boldsymbol{v}, \quad B\boldsymbol{w} = \mu\boldsymbol{w}$$

を満たすとき、

$$(A \otimes B)(\boldsymbol{v} \otimes \boldsymbol{w}) = \lambda\mu(\boldsymbol{v} \otimes \boldsymbol{w}).$$

証明

$$\begin{aligned}(A \otimes B)(\boldsymbol{v} \otimes \boldsymbol{w}) &= \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1q}B \\ a_{21}B & a_{22}B & \cdots & a_{2q}B \\ \vdots & & & \cdots \\ a_{q1}B & a_{q2}B & \cdots & a_{qq}B \end{pmatrix} \begin{pmatrix} v_1\boldsymbol{w} \\ v_2\boldsymbol{w} \\ \vdots \\ v_q\boldsymbol{w} \end{pmatrix} \\ &= \begin{pmatrix} (a_{11}v_1 + \cdots + a_{1q}v_q)B\boldsymbol{w} \\ (a_{21}v_1 + \cdots + a_{2q}v_q)B\boldsymbol{w} \\ \vdots \\ (a_{q1}v_1 + \cdots + a_{qq}v_q)B\boldsymbol{w} \end{pmatrix} = \begin{pmatrix} \lambda v_1\mu\boldsymbol{w} \\ \lambda v_2\mu\boldsymbol{w} \\ \vdots \\ \lambda v_q\mu\boldsymbol{w} \end{pmatrix} = \lambda\mu(\boldsymbol{v} \otimes \boldsymbol{w}).\blacksquare\end{aligned}$$

(4.20) より

$$\begin{aligned}B\boldsymbol{e}_{ij} &= \{[1 - 2(1-\theta)(\lambda_x + \lambda_y)] + (1-\theta)\lambda_x\mu_j + (1-\theta)\lambda_y\lambda_i\}\,\boldsymbol{e}_{ij}, \\ A\boldsymbol{e}_{ij} &= \{[1 + 2\theta(\lambda_x + \lambda_y)] - \theta\lambda_x\mu_j - \theta\lambda_y\lambda_i\}\,\boldsymbol{e}_{ij}\end{aligned}$$

となるので、

$$A^{-1}B\boldsymbol{e}_{ij} = \frac{[1-2(1-\theta)(\lambda_x+\lambda_y)]+(1-\theta)\lambda_x\mu_j+(1-\theta)\lambda_y\lambda_i}{[1+2\theta(\lambda_x+\lambda_y)]-\theta\lambda_x\mu_j-\theta\lambda_y\lambda_i}\boldsymbol{e}_{ij}.$$

ゆえに $\boldsymbol{e}_{ij}$ は $A^{-1}B$ の固有ベクトルで、

$$\nu_{ij} := \frac{[1-2(1-\theta)(\lambda_x+\lambda_y)]+(1-\theta)\lambda_x\mu_j+(1-\theta)\lambda_y\lambda_i}{[1+2\theta(\lambda_x+\lambda_y)]-\theta\lambda_x\mu_j-\theta\lambda_y\lambda_i}$$

が対応する固有値である。この ν_{ij} の定義式の分母はつねに 1 以上である。実際、$\mu_j < 2$, $\lambda_i < 2$ であるから

$$[1+2\theta(\lambda_x+\lambda_y)]-\theta\lambda_x\mu_j-\theta\lambda_y\lambda_i \geq 1+2\theta(\lambda_x+\lambda_y)-\theta\lambda_x\cdot 2-\theta\lambda_y\cdot 2 = 1.$$

(なお、$\theta > 0$ の場合は、分母は 1 より大きいことが分かる。)

また、つねに $\nu_{ij} \leq 1$ である。実際、$1-\theta \geq 0$, $\mu_j < 2$, $\lambda_i < 2$ に注意すると

$$\nu_{ij} \text{ の定義式の分子} \leq 1-2(1-\theta)(\lambda_x+\lambda_y)+(1-\theta)\lambda_x\cdot 2+(1-\theta)\lambda_y\cdot 2 = 1$$

であるから、既に見た「ν_{ij} の分母 ≥ 1」と合わせて $\nu_{ij} \leq 1$.

(実は、場合分けしてていねいに議論すると、真不等式 $\nu_{ij} < 1$ が得られる。)

ゆえに

$$r(A^{-1}B) \leq 1 \quad\Longleftrightarrow\quad \forall i \forall j \quad -1 \leq \nu_{ij}.$$

$$\begin{aligned}
-1 \leq \nu_{ij} \Longleftrightarrow\quad & -\{[1+2\theta(\lambda_x+\lambda_y)]-\theta\lambda_x\mu_j-\theta\lambda_y\lambda_i\} \\
& \leq [1-2(1-\theta)(\lambda_x+\lambda_y)]+(1-\theta)\lambda_x\mu_j+(1-\theta)\lambda_y\lambda_i \\
\Longleftrightarrow\quad & 0 \leq 2-2(1-2\theta)(\lambda_x+\lambda_y)+(1-2\theta)\lambda_x\mu_j+(1-2\theta)\lambda_y\lambda_i \\
\Longleftrightarrow\quad & 0 \leq 2-(1-2\theta)\left[\lambda_x(2-\mu_j)+\lambda_y(2-\lambda_i)\right] \\
\Longleftrightarrow\quad & (1-2\theta)\left[\lambda_x(2-\mu_j)+\lambda_y(2-\lambda_i)\right] \leq 2.
\end{aligned}$$

もしも $\dfrac{1}{2} \leq \theta \leq 1$ ならば、最後の不等式の左辺は 0 以下であるから、この不等式は無条件で成立することが分かる。

$0 \leq \theta < \dfrac{1}{2}$ の場合は割り算して、

$$-1 \leq \nu_{ij} \quad\Longleftrightarrow\quad \lambda_x(2-\mu_j)+\lambda_y(2-\lambda_i) \leq \frac{2}{1-2\theta}.$$

ゆえに

$$r(A^{-1}B) \leq 1 \quad\Longleftrightarrow\quad \lambda_x(2-\mu_p)+\lambda_y(2-\lambda_q) \leq \frac{2}{1-2\theta}.$$

つねに $2-\mu_p < 4$, $2-\lambda_q < 4$ であるが、N_x, N_y を十分大きくすると $2-\mu_p$, $2-\lambda_q$ はいくらでも 4 に近くなることに注意すると、

(a) もしも $\lambda_x+\lambda_y \leq \dfrac{1}{2(1-2\theta)}$ ならば、$r(A^{-1}B) \leq 1$ が成り立つ (実は $r(A^{-1}B) < 1$ が成立)。

(b) もしも $\lambda_x + \lambda_y > \dfrac{1}{2(1-2\theta)}$ ならば、N_x, N_y を十分大きくすることによって $r(A^{-1}B) > 1$ となる。

定理 4.7.2 (θ 法の $\|\cdot\|_2$ 安定性解析) θ 法の差分方程式 $A\boldsymbol{U}^{n+1} = B\boldsymbol{U}^n$ の係数行列 A, B について、次の (i), (ii), (iii) が成り立つ。

(i) $\dfrac{1}{2} \le \theta \le 1$ ならば、つねに $r(A^{-1}B) < 1$.

(ii) $0 \le \theta < \dfrac{1}{2}$, $\lambda_x + \lambda_y \le \dfrac{1}{2(1-2\theta)}$ ならば、$r(A^{-1}B) < 1$.

(iii) $0 \le \theta < \dfrac{1}{2}$, $\lambda_x + \lambda_y > \dfrac{1}{2(1-2\theta)}$ ならば、N_x と N_y が十分大きいとき $r(A^{-1}B) > 1$.

従って、θ 法の差分スキームが $\|\cdot\|_2$ ノルムの意味で安定であるための必要十分条件は

$$\frac{1}{2} \le \theta \le 1 \quad \text{または} \quad \left(0 \le \theta < \frac{1}{2} \quad \text{かつ} \quad \lambda_x + \lambda_y \le \frac{1}{2(1-2\theta)}\right).$$

系 4.7.3 (Crank-Nicolson 法の無条件安定性) Crank-Nicolson 法 ($\theta = 1/2$) では、任意の時間刻み τ, 任意の分割数 N_x, N_y に対して、

$$\left\|\boldsymbol{U}^{n+1}\right\|_2 \le \|\boldsymbol{U}^n\|_2 \quad (n = 0, 1, 2, \dots).$$

すなわち

$$\sum_{i=1}^{N_x-1}\sum_{j=1}^{N_y-1} \left|U_{ij}^{n+1}\right|^2 \le \sum_{i=1}^{N_x-1}\sum_{j=1}^{N_y-1} \left|U_{ij}^{n}\right|^2 \quad (n = 0, 1, \dots).$$

(議論が「緩い」ような気がする。後で整理すること。)

4.7.2 $\|\cdot\|_\infty$ ノルムに関する安定性

(とりあえず書いてしまったのだが、本来は離散最大値原理の話として書くべきであったと反省している)

どうやら、かなり一般の場合に、$A\boldsymbol{U}^{n+1} = B\boldsymbol{U}^n$ という差分方程式の解が有界であるためには、$B \ge 0$ が必要十分条件であるらしい。今の場合 (長方形領域における θ 法) も、$B \ge 0$ であれば、差分解が有界であることを証明できる。

$$B := [1 - 2(1-\theta)(\lambda_x + \lambda_y)]\, I_q \otimes I_p + (1-\theta)\lambda_x I_q \otimes J_p + (1-\theta)\lambda_y J_q \otimes I_p$$

であるから、$B \ge 0$ であるためには、$1 - 2(1-\theta)(\lambda_x + \lambda_y) \ge 0$ となることが必要十分条件で、これは

$$\theta = 1 \quad \text{または} \quad \left(0 \le \theta < 1 \quad \text{かつ} \quad \lambda_x + \lambda_y \le \frac{1}{2(1-\theta)}\right)$$

と同値である。以下、このことを仮定して、$\|B\|_\infty \leq 1$ を証明する。B の成分を b_{ij} と書くとき、

$$\forall i \quad \sum_{j=1}^{N} b_{ij} \leq 1$$

であることに注意する (多くの i について $\sum_{j=1}^{N} b_{ij} = 1$ が成り立つ)。

$\forall \boldsymbol{v} = (v_j) \in \mathbf{R}^N$ に対して、

$$\left| \sum_{j=1}^{N} b_{ij} v_j \right| \leq \sum_{j=1}^{N} |b_{ij}|\,|v_j| \leq \max_{1\leq j\leq N} |v_j| \sum_{j=1}^{N} |b_{ij}| = \|\boldsymbol{v}\|_\infty \cdot \sum_{j=1}^{N} b_{ij} \leq \|\boldsymbol{v}\|_\infty.$$

ゆえに

$$\|B\boldsymbol{v}\|_\infty = \max_{1\leq i\leq N} \left| \sum_{j=1}^{N} b_{ij} v_j \right| \leq \|\boldsymbol{v}\|_\infty. \tag{4.21}$$

一方

$$A := [1 + 2\theta(\lambda_x + \lambda_y)]\, I_q \otimes I_p - \theta\lambda_x I_q \otimes J_p - \theta\lambda_y J_q \otimes I_p$$

について調べる。$\theta = 0$ のときは $A = I_{qp}$ なので、$A^{-1} = I_{qp}$ であるから、

$$\left\|A^{-1}\boldsymbol{v}\right\|_\infty = \|\boldsymbol{v}\|_\infty .$$

以下 $0 < \theta \leq 1$ のときを調べる。$\forall \boldsymbol{v} \in \mathbf{R}^N$ に対して、$\boldsymbol{w} := A^{-1}\boldsymbol{v}$ とおくと、$\boldsymbol{v} = A\boldsymbol{w}$ である。$\boldsymbol{v}$ の典型的な (対応する格子点の上下左右の格子点が領域内部にある場合のことを指す) 成分は

$$v_i = [1 + 2\theta(\lambda_x + \lambda_y)]\, w_i - \theta\lambda_x(w_{i+1} + w_{i-1}) - \theta\lambda_y(w_{i+p} + w_{i-p})$$

となる。このとき

$$\begin{aligned}
|v_i| &= |[1 + 2\theta(\lambda_x + \lambda_y)]\, w_i - \theta\lambda_x\,(w_{i+1} + w_{i-1}) - \theta\lambda_y\,(w_{i+p} + w_{i-p})| \\
&\geq |[1 + 2\theta(\lambda_x + \lambda_y)]\, w_i| - |\theta\lambda_x w_{i+1}| - |\theta\lambda_x w_{i-1}| - |\theta\lambda_y w_{i+p}| - |\theta\lambda_y w_{i-p}| \\
&= [1 + 2\theta(\lambda_x + \lambda_y)]\, |w_i| - \theta\lambda_x\, |w_{i+1}| - \theta\lambda_x\, |w_{i-1}| - \theta\lambda_y\, |w_{i+p}| - \theta\lambda_y\, |w_{i-p}| \\
&\geq [1 + 2\theta(\lambda_x + \lambda_y)]\, |w_i| - 2\theta(\lambda_x + \lambda_y)\, \|\boldsymbol{w}\|_\infty .
\end{aligned}$$

すなわち

$$|v_i| \geq [1 + 2\theta(\lambda_x + \lambda_y)]\, |w_i| - 2\theta(\lambda_x + \lambda_y)\, \|\boldsymbol{w}\|_\infty .$$

この不等式自体は、任意の i について成立することが分かる。i について、両辺の最大値を取って

$$\max_{1\leq i\leq N} |v_i| \geq \max_{1\leq i\leq N} \{[1 + 2\theta(\lambda_x + \lambda_y)]\, |w_i| - 2\theta(\lambda_x + \lambda_y)\, \|\boldsymbol{w}\|_\infty\} .$$

ゆえに

$$\|\boldsymbol{v}\|_\infty \geq [1 + 2\theta(\lambda_x + \lambda_y)]\, \|\boldsymbol{w}\|_\infty - 2\theta(\lambda_x + \lambda_y)\, \|\boldsymbol{w}\|_\infty = \|\boldsymbol{w}\|_\infty .$$

すなわち

$$\left\|A^{-1}\boldsymbol{v}\right\|_\infty = \|\boldsymbol{v}\|_\infty \leq \|\boldsymbol{v}\|_\infty . \tag{4.22}$$

(4.21), (4.22) から、

$$\left\|U^{n+1}\right\|_\infty = \left\|A^{-1}B\boldsymbol{U}^n\right\|_\infty \le \|B\boldsymbol{U}^n\|_\infty \le \|\boldsymbol{U}^n\|_\infty .$$

以上より、次の定理が得られた。

定理 4.7.4 (θ 法の $\|\cdot\|_\infty$ 安定性解析) θ, λ_x, λ_y が

$$\theta = 1 \quad \text{または} \quad \left(0 \le \theta < 1 \quad \text{かつ} \quad \lambda_x + \lambda_y \le \frac{1}{2(1-\theta)}\right)$$

を満たすとき、任意の n について

$$\left\|\boldsymbol{U}^{n+1}\right\|_\infty \le \|\boldsymbol{U}^n\|_\infty .$$

θ, λ_x, λ_y が定理の条件を満たさない場合に、$\|\boldsymbol{U}^n\|_\infty$ が発散しうることを示したいが、これについては後日。

付 録A　サンプル・プログラムについて

A.1　プログラムの公開

この文書に載っているプログラムは、自由に利用してもらって構わない。
ほとんどのプログラムは、『公開プログラムのページ』[1] からファイルとして入手できる。

A.2　GLSC ライブラリィ

多くのプログラムで、グラフィックスの実現に **GLSC** (を少し機能拡張したもの) を用いている。
GLSC については、『明大数学科計算機室ユーザーのための GLSC の紹介』[2] を見よ。

A.3　matrix ライブラリィ

多くのプログラムで、matrix library という小さな自作ライブラリィを使用している。詳しい説明は https://m-katsurada.sakura.ne.jp/program/matrix/matrix-lib3.tar.gz の中の文書を読んでもらいたいが (文書を直接 WWW に載せて、その URL を書くようにしよう)、C 言語のプログラムで簡単に行列を扱えるようにするためのもので、使い方の要点は、例えば

```
#include "matrix.h"
  ...
  matrix a;
```

のように変数 a を宣言しておき、

```
  a = new_matrix(3, 2);
  if (a == NULL) {
    fprintf(stderr, "メモリーが確保できませんでした。\n");
    exit(1);
  }
```

のようにすると、double a[3][2]; と宣言したのと同様に a[0][0], a[0][1], a[1][0], a[1][1], a[2][0], a[2][1] という double 型の変数が使えるようになる、ということである。

[1] https://m-katsurada.sakura.ne.jp/program/
[2] https://m-katsurada.sakura.ne.jp/labo/howto/intro-glsc/

参考文献

[1] 菊地文雄, 山本昌宏：微分方程式と計算機演習, 山海堂 (1991).

[2] 桂田祐史：微分方程式 2 講義ノート (旧「応用解析 II」), https://m-katsurada.sakura.ne.jp/lecture/pde/pde2013.pdf (1997 年〜).

[3] Farlow, S. J.: *Partial Differential Equations for Scientists and Engineers*, John Wiley & Sons, Inc (1982), 邦訳: スタンリー・ファーロウ 著, 入理 正夫・入理 由美 訳, 偏微分方程式, 朝倉書店 (1996).

[4] 藤田宏, 黒田成俊(しげとし), 伊藤清三：関数解析, 岩波書店 (1991), 岩波講座 基礎数学 (1978 年) の書籍化.

[5] 伊藤清三(せいぞう)：偏微分方程式, 培風館 (1966, 1983).

[6] 伊藤清三：拡散方程式, 紀伊国屋書店 (1979).

[7] 増田久弥(きゅうや)：発展方程式, 紀伊国屋書店 (1975).

[8] 丹羽功, 山田英二：△ (ラプラス作用素) の固有値問題の差分法による解法, https://m-katsurada.sakura.ne.jp/labo/report/pdf/1997-niwa-yamada.pdf (1998).

[9] 高藤康孝：偏微分方程式の固有値問題, https://m-katsurada.sakura.ne.jp/labo/report/pdf/1997-takatou.pdf (1998).

[10] 鈴木康大：偏微分方程式の固有値問題の有限要素法による解法, https://m-katsurada.sakura.ne.jp/labo/report/pdf/1998-suzuki.pdf (1999).

[11] 俣野博：微分方程式 II, 岩波講座 応用数学, 岩波書店 (1994).

[12] Protter, M. H. and Weinberger, H. F.: *Maximum Principles in Differential Equations*, Springer-Verlag (1984).

[13] 桂田祐史：熱方程式に対する差分法 II — 円盤における熱方程式 —, https://m-katsurada.sakura.ne.jp/labo/text/heat-fdm-2.pdf (1998〜).

[14] 桂田祐史：発展系の数値解析, https://m-katsurada.sakura.ne.jp/labo/text/heat-fdm-0.pdf (1997 年〜).

[15] 伊理正夫(いりまさお)：一般線形代数, 岩波書店 (2003), 伊理正夫, 線形代数 I, II, 岩波講座応用数学, 岩波書店 (1993, 1994) の単行本化.

[16] 桂田祐史：長方形領域における熱方程式に対する差分法 — MATLAB を使って数値計算 —, https://m-katsurada.sakura.ne.jp/labo/text/heat2d.pdf (2015 年〜).

[17] 桂田祐史：2 次元熱方程式の非同次 Dirichlet 境界値問題を解く差分法プログラムを作る, https://m-katsurada.sakura.ne.jp/labo/text/heat2d-nonhomo.pdf (2024 年〜).

[18] 桂田祐史：同次 Neumann 境界条件下の熱方程式に対する差分法 — MATLAB を使って数値計算 —, https://m-katsurada.sakura.ne.jp/labo/text/heat2n.pdf (2015 年〜).

[19] 桂田祐史：非同次 Neumann 境界条件下の熱方程式に対する差分法 — MATLAB を使って数値計算 —, https://m-katsurada.sakura.ne.jp/labo/text/heat2n-nonhomo.pdf (2024/8/25, 2024/8/24).

[20] 桂田祐史：連立 1 次方程式 I, https://m-katsurada.sakura.ne.jp/labo/text/linear-eq-1.pdf (2002 年〜).

[21] 杉浦光夫, 横沼健雄：Jordan 標準形・テンソル代数, 岩波書店 (1990), この前半は、杉浦 光夫, Jordan 標準形と単因子論 I, II, 岩波講座 基礎数学 (1976,1977) が元になっている。

[22] Smith, G. D.: *Numerical solution of partial differential equations third edition*, Clarendon Press Oxford (1986), 第一版の邦訳が G. D. スミス著, 藤川洋一郎訳, コンピュータによる偏微分方程式の解法 新訂版, サイエンス社 (1996) である.

[23] 田端正久：偏微分方程式の数値解法, 岩波書店 (2010), もともとは岩波講座応用数学の「微分方程式の数値解法 II」(1994) であった。

[24] OBrien, G. G., Hyman, M. A. and Kaplan, S.: A study of the numerical solution of partial differential equations, *J. Math. Phys.*, Vol. **29**, pp. 223–251 (1951).