新 MATLAB 入門

桂田 祐史

2018 年 8 月 24 日, 2024 年 9 月 10 日

思い出すたびに MATLAB を採り上げて、にわか勉強で動くプログラムを書く、というのを続けてきて、ガラクタの山が残った (https://m-katsurada.sakura.ne.jp/labo/text/matlab/)。いいかげんに整理しよう (目的別にこうするのが良いだろう、というものをまとめる。結局は今回もすでに作ってあるものの寄せ集めであるが、一つの目的に複数のプログラムがあった場合は、なるべく一つにまとめる。)。

使用上の注意: 「とにかく一つの文書にまとめておく」が主目的なので、全体の出来は非常に粗い。間違っているところも少なからずあると思われる。ちらっと見て間違っているように感じられても、確認する時間が取れないので、あえてそのまま放置してあるところが多い (ちら見で考えたことが正しいという保証はないので)。自分で再利用するときは、書いてあることを読んで、プログラムを動かしながら、内容を思い出し、確認してから先に進むつもりでいる。

目 次

1 MATLAB とは何か

(あとまわし)

1.1 MATLAB とは、その起源

MATLAB (MATrix LABoratory) は、著名な線形演算ライブラリィ **LINPACK**, **EISPACK** の開発でも中心的な役割を果たした Cleve Moler が、1980 年頃に製作したものが発展した数値実験環境 (「実験室」) である (当時の開発言語は FORTRAN)。彼は 1985 年に C 言語で MATLAB を書き直し、MathWorks 社を設立して販売を開始した (Moler は会長兼技師長であるとか — 現在もそうであるかは知らない)。

MATLAB の特徴

- インタープリター型言語である。そのため[a]、
  - 対話的で使いやすいシステムになっている。
  - 注意深く利用しないと実行効率が低くなる[b](個々の命令の実行時に命令解釈のコストが必要なため、繰り返し処理を多用すると計算時間が長くなりがちである)。
- **LAPACK** などの各種数値計算ライブラリィを内蔵している (これらのライブラリィ群へのインターフェイスであると理解すべきかもしれない)。
- ベクトル、行列などのデータの型が始めから定義されているので、命令が簡潔になっていて、プログラミングも楽になった[c]。

[a] ここで指摘することは、例えば **Mathematica**, **Maple** のような多くの**数式処理系**にも当てはまる。

[b] ここで述べたような注意は、かつてはパソコン上で BASIC 言語を使ってプログラムを開発する際の常識であったのだが、今ではあまり知られていないことなのだろう。

[c] オブジェクト指向であり、データ構造が隠蔽されていると言って良いかもしれない。LAPACK などの利用で面倒な点の一つに、プログラマーにライブラリィ中で定義されたデータ構造を正しくなぞったプログラムを書く努力が要求されるというものがあるが、MATLAB ではこれがなくなっている。この点は C++ で書かれたライブラリィでも期待できることであるが。

MATLAB をいかに評価するか。筆者自身は最初は「ちょっと便利」くらいにしか感じなかったが、使い続けるうちに

案外大したものではないか

さらに

ひょっとするとコロンブスの卵で大発明？

と考えるようになった。このような**システムを作るのは実は簡単**で (実際、「真似」がたくさん出て来た)、しかし使ってみると分るが、**非常に便利**である。

日本の数学界ではあまり人気がない (というか知られていない) ようであるが、欧米や、日本でも工学の世界では浸透している。

MATLAB は改良が続けられていて、行列計算関係では、**疎行列向きの処理法**や**反復法**なども採り入れられている。偏微分方程式のシミュレーションへの応用にも「もってこい」レベルに成長した。

MATLAB を後を追ったシステムがたくさん開発されたが、MATLAB の言語仕様は「デファクト・スタンダード」となっている。以下 MATLAB と似たシステムをいくつか紹介しよう[1]。いずれもソース・プログラム公開のフリーソフトウェアである。

(GNU) Octave MATLAB との互換性が高い。入門には十分であるし、用途を選べば実用性も高い。むしろ、中身が完全に公開されているということを積極的に評価すべきである

[1] http://www.dspguru.com/sw/opendsp/mathclo2.htm などが参考になる。

かもしれない。近年、GUI を採用したせいで、make が難しくなり、動作が不安定な時期があった (個人的には GUI に力を入れたことには、不満であった)。

Scilab MATLAB との互換性の程度は Octave よりも低いが、ソフトウェアとしての完成度はやや高いかもしれない (と感じた時期があった、現在でもそういう評価になるかは知らない)。

2 1 次元 Poisson 方程式

桂田 [1] というかなり詳しい説明を書いた。ここでは、[1] で作成したプログラムを紹介する。

2.1 Dirichlet 境界値問題

$$-u''(x)=f(x)\quad (x\in(0,L)),\quad u(0)=u(L)=0$$
$$h=\frac{L}{N},\quad x_i=ih\quad (i=0,1,\cdots,N).$$
$$\frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{pmatrix}=\begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{N-1}) \end{pmatrix}.$$

laplacian1d.m

```
% laplacian1d.m
% curl -O https://m-katsurada.sakura.ne.jp/program/fdm/laplacian1d.m
% -(d/dx)^2 の差分近似
% [0,L] を N 等分した時の行列は h=L/N; a=laplacian1d(N-1)/(h*h);
function a=laplacian1d(m)
  e=ones(m,1);
  a=spdiags([-e 2*e -e],-1:1,m,m); % 疎行列なので圧縮した形式
end
```

```
>> a=laplacian1d(3));
>> full(a)

ans =
     2    -1     0
    -1     2    -1
     0    -1     2
```

$I=(0,L)$ を N 等分する差分近似の場合は

```
>> L=1;
>> n=5;
>> h=L/n;
>> a=laplacian1d(n-1)/(h*h);
>> full(a)

ans =

   50.0000  -25.0000         0         0
  -25.0000   50.0000  -25.0000         0
         0  -25.0000   50.0000  -25.0000
         0         0  -25.0000   50.0000
```

$I = (0, L)$ における

$$-u''(x) = 1, \quad u(0) = u(L) = 0$$

を解くには ($f \equiv 1$ とするのは安直だけど)

poisson1d_test1.m

```
L=1
n=10
h=L/n
x=linspace(0,L,n+1);
a=laplacian1d(n-1)/(h*h);
f=ones(n-1,1);
u=a\f;
u=[0; u; 0];
plot(x,u)
figure(gcf)
```

とする。

$f(x) = \sin x$ の場合は

poisson1d_test2.m

```
L=1
n=10
h=L/n
x=linspace(0,L,n+1);
a=laplacian1d(n-1)/(h*h);
f=sin(x(2:n))';
u=a\f;
u=[0; u; 0];
plot(x,u)
figure(gcf)
```

とする。

$f(x) = x(1 - x)$ の場合は

poisson1d_test3.m

```
L=1
n=10
h=L/n
x=linspace(0,L,n+1);
a=laplacian1d(n-1)/(h*h);
f=x.*(1-x);
f=f(2:n)';
u=a\f;
u=[0; u; 0];
plot(x,u)
figure(gcf)
```

とする。

(2024/9 改訂版) 非同次 Dirichlet 境界値問題のプログラム。

poisson1d_nh.m

```
% poisson1d_nh.m
% 1 次元領域における Poisson -△ u=f （非同次 Dirichlet 境界条件）を差分法で解く
% 入手 curl -O https://m-katsurada.sakura.ne.jp/program/fdm/poisson1d_nh.m
% 解説 https://m-katsurada.sakura.ne.jp/labo/text/poisson1d.pdf
% (2024/9/4)。
function poisson1d_nh(N)
  arguments
    N=4 % デバッグ用
  end
  function K = poisson1d_mat(W,N)
    h=W/N;
    I=speye(N-1,N-1);
    v=ones(N-2,1);
    J=sparse(diag(v,1)+diag(v,-1));
    K=2*I-J;
    K=1/h^2*K;
  end
  % 想定している厳密解
  function val = exact_solution(x)% =x^4
    val = x .^ 4;
    %val = (x - 1.0/2).^2;
  end
  % f=-△ u=-d^2u/dx^2
  function val = f(x)% = -u''(x) = -12x^2
    val = - 12 * x.^2;
    %val = 0 .* x - 2.0;
  end
  % 領域
  a=0; b=1;
  % 境界値
  alpha = exact_solution(a);
  beta = exact_solution(b);
  fprintf('a=%f, b=%f, u(a)=%f, u(b)=%f\n', a, b, alpha, beta);
  % 差分近似の行列
  h=(b-a)/N;
  fprintf('N=%d, h=%g\n', N, h);
  A=poisson1d_mat(b-a,N);
  if N <= 10
    disp('A='); disp(full(A));
  end
```

```
  % 連立方程式の右辺
  X=linspace(a,b,N+1);
  F=f(X(2:N)');
  F(1)=F(1)+alpha/h^2;
  F(N-1)=F(N-1)+beta/h^2;
  if N <= 10
    disp('X='); disp(X);
    disp('F=f(X)='); disp(F');
  end
  % 差分解を求める
  u=zeros(N+1,1);
  u(2:N)=A\F;
  % 境界値
  u(1)=alpha; u(N+1)=beta;
  % グラフを描く
  plot(X,u); title('差分解のグラフ'); drawnow; shg;
  figure
  plot(X,exact_solution(X')); title('厳密解のグラフ'); drawnow; shg;
  exact=exact_solution(X');
  error=norm(u-exact,inf);
  fprintf('max norm of error=%e\n', error);
end
```

入手するにはターミナルで

```
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/poisson1d_nh.m
```

これを実行するには、例えば

```
cp -p poisson1d_nh.m ~/Documents/MATLAB
```

とコピーして (私はシンボリック・リンクをはることにしている)、MATLAB の中で

```
>> poisson1d_nh

(分割数はデフォールトの N = 4 が採用される。)
```

あるいは、分割数 N を指定するため

```
>> poisson1d_nh(10)
```

のようにする。

上のサンプル・プログラムでは、$u(x) = x^4$ と選び、$f(x) = -u''(x) = -12x^2$, $\alpha = u(0) = 0$, $\beta = u(1) = 1$ としてある。N を 10 から倍々にしていくと以下のようになった。

```
N=10, h=0.1
max norm of error=2.500000e-03

N=20, h=0.05
max norm of error=6.250000e-04

N=40, h=0.025
max norm of error=1.562500e-04

N=80, h=0.0125
max norm of error=3.906250e-05

N=160, h=0.00625
max norm of error=9.765625e-06

N=320, h=0.003125
max norm of error=2.441406e-06
```

N が 2 倍になると誤差は $\frac{1}{4}$ 倍になる。誤差は $O(1/N^2)$ になっているようで、ひとまずは納得できる。

2.2 Neumann 境界値問題

poisson1n_nh.m

```
% poisson1n_nh.m
% 1 次元領域における Poisson -△ u=f （非同次 Neumann 境界条件）を差分法で解く
% 入手 curl -O https://m-katsurada.sakura.ne.jp/program/fdm/poisson1n_nh.m
% 解説 https://m-katsurada.sakura.ne.jp/labo/text/poisson2n-nonhomo.pdf
% (2024/8/27)。
function poisson1n_nm(N)
  arguments
    N=4 % デバッグ用
  end
  function K = poisson1n_mat(W,N)
    h=W/N;
    I=speye(N+1,N+1);
    v=ones(N,1);
    J=sparse(diag(v,1)+diag(v,-1));
    K=2*I-J;
    K(1,1)=1; K(1,2)=0; K(2,1)=0;
    K(N+1,N+1)=1; K(N+1,N+1)=1;
    K=1/h^2*K;
  end
  % 想定している厳密解
  function val = exact_solution(x)% =x*(1-x)=x-x^2
    val = x .* (1 - x);
  end
  % f=-△ u=-d^2u/dx^2
  function val = f(x)% = -u''(x) = 2
    val = 0 .* x + 2.0;
  end
  % ∂ u/∂ n(a)=-∂ u/∂ x(a)
  function val = b_l(x)% =-u'(0)=-(1-2x)|x=0
    val = -1;
  end
  % ∂ u/∂ n(b)=∂ u/∂ x(b)
  function val = b_r(x)% 1-2x|x=1
    val = -1;
  end
  % 領域
  a=0; b=1;
  beta=-1;
  h=(b-a)/N;
  fprintf('N=%d, h=%g\n', N, h);
  A=poisson1n_mat(b-a,N);
  if N <= 10
    disp('A='); disp(full(A));
  end
  fprintf('det(A)=%e\n', det(A))
  fprintf('cond(A)=%e\n', condest(A));
```

入手するにはターミナルで

```
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/poisson1n_nh.m
```

これを実行するには、例えば

```
cp -p poisson1n_nh.m ~/Documents/MATLAB
```

とコピーして (私はシンボリック・リンクをはることにしている)、MATLAB の中で

```
>> poisson1n_nh

(分割数はデフォールトの N = 4 が採用される。)
```

あるいは、分割数 N を指定するため

```
>> poisson1n_nh(10)
```

のようにする。

3　2次元Poisson方程式

3.1　Dirichlet境界値問題

まずは同次 Dirichlet 境界値問題のプログラムを示す。
長方形領域 $\Omega = (a,b) \times (c,d)$ における Poisson 方程式の境界値問題

$$-\triangle u = f \quad \text{in } \Omega, \quad u = 0 \quad (\text{on } \partial\Omega)$$

を考える。

$$\begin{aligned}
&h_x = \frac{b-a}{N_x}, \quad h_y = \frac{d-c}{N_y}, \\
&x_i = a + ih_x \quad (i = 0, 1, \ldots, N_x), \quad y_j = c + jh_y \quad (j = 0, 1, \ldots, N_y), \\
&u_{ij} = u(x_i, y_j) \quad (i = 0, 1, \ldots, N_x;\ j = 0, 1, \ldots, N_y)
\end{aligned}$$

とおき、u_{ij} の近似値を U_{ij} を求めるための差分方程式は

$$-\left(\frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{h_x^2} + \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{h_y^2}\right) = f(x_i, y_j) \tag{1}$$
$$(1 \le i \le N_x - 1,\ 1 \le j \le N_y - 1),$$

$$U_{0,j} = U_{N_x,j} = 0 \quad (0 \le j \le N_y), \tag{2}$$

$$U_{i,0} = U_{0,N_y} = 0 \quad (0 \le i \le N_x) \tag{3}$$

Dirichlet 境界値問題については、U_{ij} $(1 \le i \le N_x,\ 1 \le j \le N_y)$ が未知数になる。桂田 [2] というノートでは

$$U_\ell = U_{i,j}, \quad \ell = i + j(N_x - 1) \quad (1 \le i \le N_x - 1,\ 1 \le j \le N_y - 1) \tag{4}$$

とおき、$\boldsymbol{U} = \begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_N \end{pmatrix}$ を未知ベクトルとする連立 1 次方程式を考えた。連立 1 次方程式の係数行列は

$$A = \frac{1}{h_y^2}(2I_{N_y-1} - J_{N_y-1}) \otimes I_{N_x-1} + I_{N_y-1} \otimes \frac{1}{h_x^2}(2I_{N_x-1} - J_{N_x-1})$$

であった。

以下のプログラムでは (4) の代わりに

$$U_\ell = U_{i,j}, \quad \ell = j + i(N_y - 1) \quad (1 \le i \le N_x - 1,\ 1 \le j \le N_y - 1) \tag{5}$$

とおく。この場合の係数行列は

$$A = I_{N_x-1} \otimes \frac{1}{h_y^2}(2I_{N_y-1} - J_{N_y-1}) + \frac{1}{h_x^2}(2I_{N_x-1} - J_{N_x-1}) \otimes I_{N_y-1}.$$

poisson_coef.m

```
function A=poisson_coef(W, H, nx, ny)
% 長方形領域 (0,W) × (0,H) における Poisson 方程式の Dirichlet 境界値問題
% Laplacian を差分近似した行列を求める。
% 長方形を nx × ny 個の格子に分割して差分近似する。
% MATLAB では
%  (1) 行列は Fotran と同様の column first であり、
%  (2) mesh(), contour() による「行列描画」は Z(j,i) と添字の順が普通と逆なので、
% l=i+(j-1)*(nx-1) と row first となるように 1 次元的番号付けする
  hx=W/nx;
  hy=H/ny;
  m=nx-1;
  n=ny-1;
  ex=ones(nx,1);
  ey=ones(ny,1);
  Lx=spdiags([-ex,2*ex,-ex],-1:1,m,m)/(hx*hx);
  Ly=spdiags([-ey,2*ey,-ey],-1:1,n,n)/(hy*hy);
  A=kron(speye(m,m),Ly)+kron(Lx,speye(n,n));
```

poisson2d.m

```
% 長方形領域 (0,W) × (0,H) で Poisson 方程式の同次 Dirichlet 境界値問題を解く
W=3.0;
H=2.0;
nx=30;
ny=20;
m=nx-1;
n=ny-1;
% 連立方程式を作成して解く
% MATLAB の行列は Fotran と同様の column first であり、
%「行列描画」は Z(j,i) と添字の順が普通と逆なので、
% l=i+(j-1)*(nx-1) と row first となるように 1 次元的番号付けする
A=poisson_coef(W, H, nx, ny);
%
x=linspace(0,W,nx+1); % x=[x_0,x_1,...,x_nx]
y=linspace(0,H,ny+1); % y=[y_0,y_1,...,y_ny]
[X,Y]=meshgrid(x,y);
% f ≡ 1 の場合
%F=ones(m*n,1);
```

```
f=-2*(X.^2-3*X+Y.^2-2*Y);
f=f(2:ny,2:nx);
F=f(:);
%
U=zeros(n,m);
U(:)=A\F;
% 境界値 0 をつける
u=zeros(ny+1,nx+1);
u(2:ny,2:nx)=U;
%
% グラフの鳥瞰図
clf
colormap hsv
subplot(1,2,1);
mesh(X,Y,u);
colorbar
% 等高線
subplot(1,2,2);
contour(X,Y,u);
%
disp('図を保存する');
print -dpdf poisson2d.pdf % 利用できるフォーマットは doc print で分かる
print -dpng poisson2d.png % 利用できるフォーマットは doc print で分かる
print -deps poisson2d.eps % 利用できるフォーマットは doc print で分かる
```

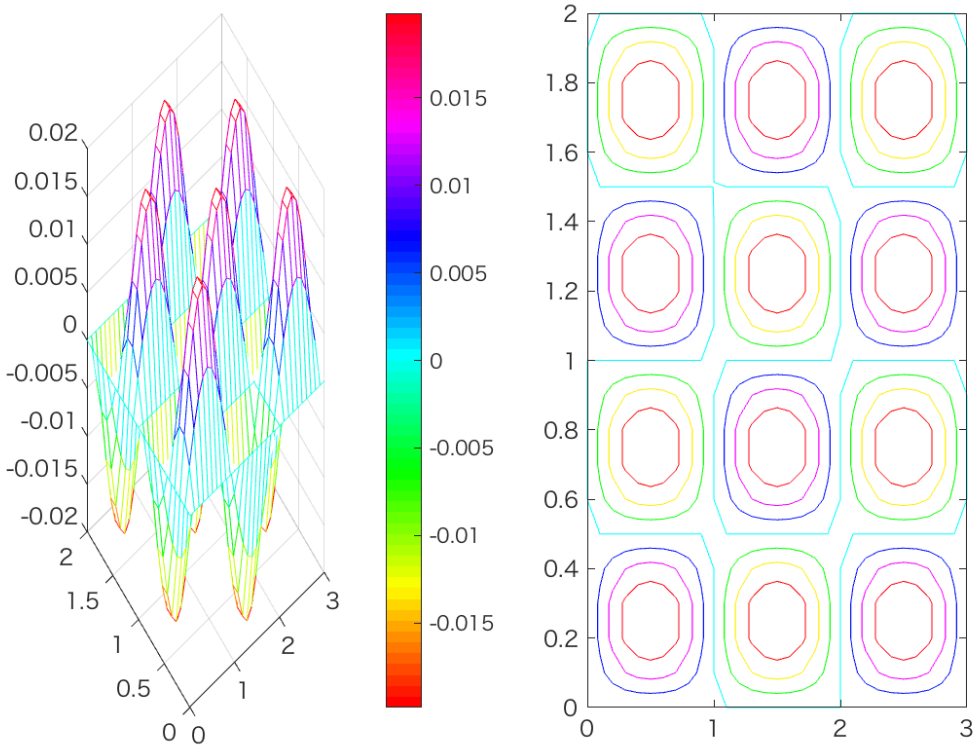

図 1: poisson2d.m の結果

まったく別の時期に作ったプログラム。ほとんど同じで我ながら唖然とする。最初のプログラムが `mesh()` と `contour()` で、鳥瞰図と等高線を別々に描いたが、こちらは `meshc()` で同時に描いている。

poisson2d_v2.m

```
% poisson2d.m --- Poisson equation -△u=sin(π x)sin(2 π y) (0<x<3, 0<y<2), u=0
% [0,3] × [0,2] を 30 × 20 に分割する
a=0; b=3; c=0; d=2;
nx=30; ny=20;
%nx=6; ny=4;
X=linspace(a,b,nx+1);
Y=linspace(c,d,ny+1);
[x,y]=meshgrid(X,Y);
% f(x,y)=sin(x)sin(2y)
F=sin(pi*x).*sin(2*pi*y);
% 係数行列
hx=(b-a)/nx; hy=(d-c)/ny;
e=ones(nx-1,1);
ax=spdiags([-e 2*e -e],-1:1,nx-1,nx-1)/(hx*hx);
e=ones(ny-1,1);
ay=spdiags([-e 2*e -e],-1:1,ny-1,ny-1)/(hy*hy);
a=kron(speye(nx-1),ay)+kron(ax,speye(ny-1));
% F の境界部分の値を削除して、1 次元化
f=F(2:end-1,2:end-1);
f=f(:);
%
u=zeros(ny+1,nx+1);
u(2:end-1,2:end-1)=reshape(a\f,ny-1,nx-1);
%
figure('Name','Poisson equation -△u=sin(π x)sin(2 π y) (0<x<3, 0<y<2)')
meshc(x,y,u)
figure(gcf)
```

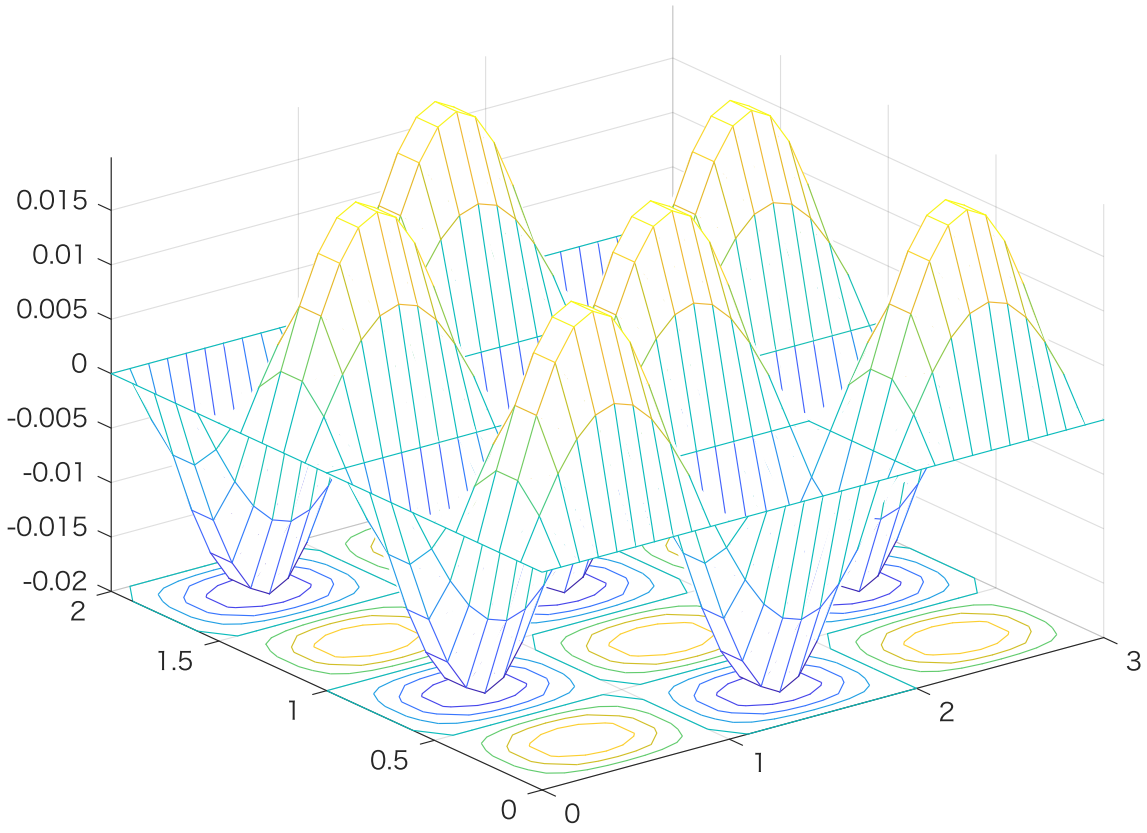

図 2: poisson2d_v2.m の結果

3.1.1 非同次 Neumann 境界値問題

非同次の問題の解説 [3] も書いた。
ここでは、[3] で作成したプログラムを紹介する、

poisson2d_nh.m

```
% poisson2d_nh.m
% 長方形領域における Poisson -△ u=f （非同次 Dirichlet 境界条件）を差分法で解く
% 入手 curl -O https://m-katsurada.sakura.ne.jp/program/fdm/poisson2d_nh.m
% 解説 https://m-katsurada.sakura.ne.jp/labo/text/poisson2d-nonhomo.pdf
% 一応の完成をして、poisson2d_nh.m として公開した（2024/8/25)。

function poisson2d_nh(Nx,Ny)
  arguments
    Nx=50
    Ny=50
  end
  % Dirichlet 境界条件の場合の係数行列
  function A = poisson2d_mat(W,H,Nx,Ny)
    Ix=speye(Nx-1,Nx-1);
    Iy=speye(Ny-1,Ny-1);
    vx=ones(Nx-2,1);
    Jx=sparse(diag(vx,1)+diag(vx,-1));
    vy=ones(Ny-2,1);
    Jy=sparse(diag(vy,1)+diag(vy,-1));
    hx = W / Nx;
    hy = H / Ny;
    betax = 1.0 / hx^2;
    betay = 1.0 / hy^2;
    Kx=betax * (2*Ix - Jx);
    Ky=betay * (2*Iy - Jy);
    % row major (y changes first, l=j+(Ny-1)*(i-1))
    A=kron(Kx,Iy) + kron(Ix,Ky);
    % column major (x chandes first, l=i+(Nx-1)*(j-1))
    %  A=kron(Iy,Kx) + kron(Ky,Ix);
  end
  function A=poisson_coef(W, H, nx, ny)
    hx=W/nx;
    hy=H/ny;
    m=nx-1;
    n=ny-1;
    ex=ones(nx,1);
    ey=ones(ny,1);
    Lx=spdiags([-ex,2*ex,-ex],-1:1,m,m)/(hx*hx);
    Ly=spdiags([-ey,2*ey,-ey],-1:1,n,n)/(hy*hy);
    A=kron(speye(m,m),Ly)+kron(Lx,speye(n,n));
  end
  % 長方形領域 (a,b) × (c,d)
  a=0; b=1; c=0; d=1;
  %
  function u = exact_solution(x, y)
    u = cos(pi * x) .* cos(pi * y);
  end
  % Poisson 方程式の右辺 (exact_solution の -△)
  function val = f(x, y)
```

入手するには、ターミナルで

```
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/poisson2d_nh.m
```

これを実行するには、例えば

```
cp -p poisson2d_nh.m ~/Documents/MATLAB
```

とコピーして (私はシンボリック・リンクをはることにしている)、MATLAB の中で

```
>> poisson2d_nh
```

(分割数はデフォールトの $N_x = 50$, $N_y = 50$ が採用される。)

あるいは、分割数 N_x, N_y を指定して

```
>> poisson2d_nh(100,100)
```

とする。

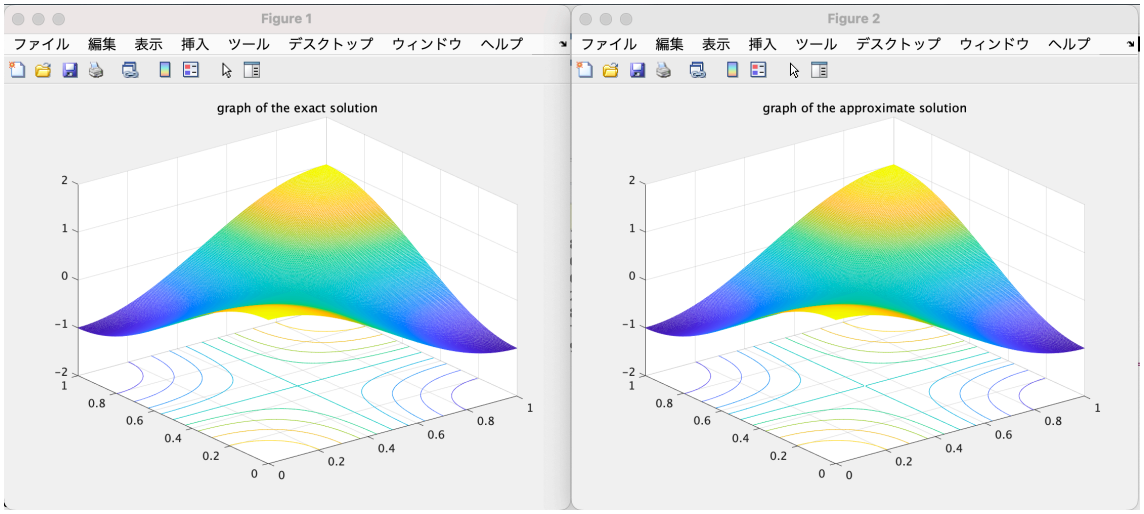

図 3: $Nx = Ny = 200$ 厳密解と差分解 (違いが見て分かるはずはないが…)

誤差は次のようになった。$O(h_x^2 + h_y^2)$ となっていると考えている。

```
>> poisson2d_nh
Nx=50, Ny=50, hx=0.020000, hy=0.020000, betax=2500.000000, betay=2500.000000
Nx=50, Ny=50, ||error||=5.924640e-05, ||u||=1.000000e+00

>> poisson2d_nh(100,100)
Nx=100, Ny=100, hx=0.010000, hy=0.010000, betax=10000.000000, betay=10000.000000

Nx=100, Ny=100, ||error||=1.482114e-05, ||u||=1.000000e+00

>> poisson2d_nh(200,200)
Nx=200, Ny=200, hx=0.005000, hy=0.005000, betax=40000.000000, betay=40000.000000

Nx=200, Ny=200, ||error||=3.705884e-06, ||u||=1.000000e+00

>> poisson2d_nh(400,400)
Nx=400, Ny=400, hx=0.002500, hy=0.002500, betax=160000.000000, betay=160000.000000
Nx=400, Ny=400, ||error||=9.265347e-07, ||u||=1.000000e+00
```

3.2 Neumann 境界値問題

桂田 [4] という解説を書いた。

poisson2n_nh.m

```
% poisson2n_nh.m
% 長方形領域における Poisson -△ u=f（非同次 Neumann 境界条件）を差分法で解く
% 入手 curl -O https://m-katsurada.sakura.ne.jp/program/fdm/poisson2n_nh.m
% 一応の完成をして、poisson2n_nh.m として公開した（2024/8/27)。

function poisson2n_nh(Nx,Ny)
  arguments
    Nx=50
    Ny=50
  end
  % Neumann 境界条件の場合の係数行列
  % poisson2n_mat.m
  function A = poisson2n_mat(W,H,Nx,Ny)
    Ix=speye(Nx+1,Nx+1);
    Iy=speye(Ny+1,Ny+1);
    vx=ones(Nx,1);
    Jx=sparse(diag(vx,1)+diag(vx,-1));
    Jx(1,2)=2; Jx(Nx+1,Nx)=2;
    vy=ones(Ny,1);
    Jy=sparse(diag(vy,1)+diag(vy,-1));
    Jy(1,2)=2; Jy(Ny+1,Ny)=2;
    hx = W / Nx;
    hy = H / Ny;
    betax = 1.0 / hx^2;
    betay = 1.0 / hy^2;
    Kx=betax * (2*Ix - Jx);
    Ky=betay * (2*Iy - Jy);
    % row major (y changes first, l=j+(Ny+1)*i)
    A=kron(Kx,Iy) + kron(Ix,Ky);
    % column major (x chandes first, l=i+(Nx+1)*j)
    %  A=kron(Iy,Kx) + kron(Ky,Ix);
  end
  % 長方形領域 (a,b) × (c,d)
  a=0; b=1; c=0; d=1;
  %
  function u = exact_solution(x, y)
    u = sin(pi * x) .* sin(pi * y);
  end
  % Poisson 方程式の右辺 (exact_solution の -△)
  function val = f(x, y)
    val = 2 * pi^2 * sin(pi * x) .* sin(pi * y);
  end
  % 長方形の下の辺での b() = -∂u/∂y=-π sin πx cos π0=-π sin πx
  function val = b_b(x)
    val = - pi * sin(pi * x);
  end
  % 長方形の上の辺での b() = ∂u/∂y=π sin πx cos π・1=-π sin πx
  function val = b_t(x)
```

入手するには、ターミナルで

```
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/poisson2n_nh.m
```

これを実行するには、例えば

```
cp -p poisson2n_nh.m ~/Documents/MATLAB
```

とコピーして (私はシンボリック・リンクをはることにしている)、MATLAB の中で

```
>> poisson2n_nh
```

(分割数はデフォールトの $N_x = 50$, $N_y = 50$ が採用される。)

あるいは、分割数 N_x, N_y を指定して

```
>> poisson2n_nh(100,100)
```

とする。

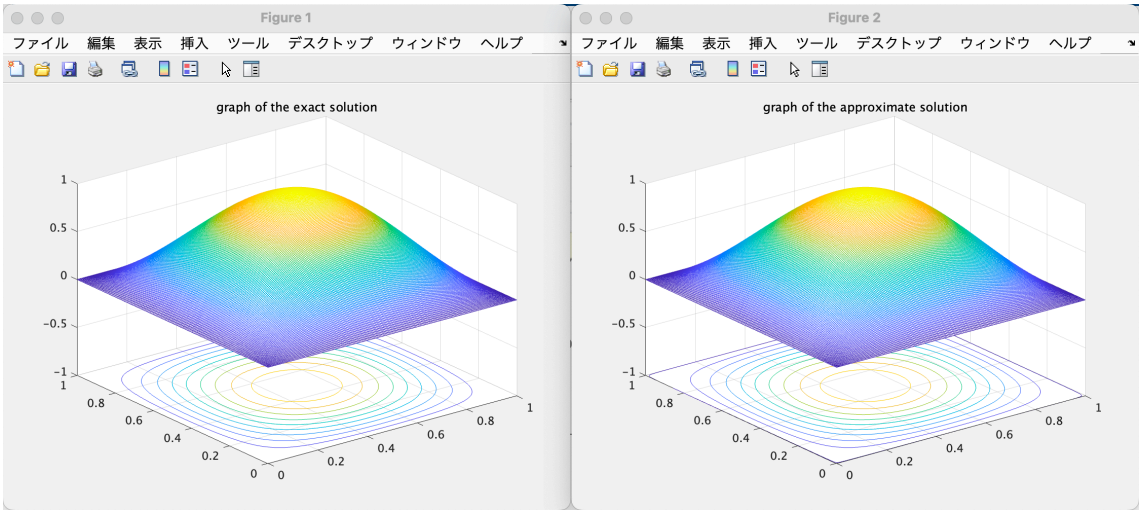

図 4: $Nx = Ny = 200$ 厳密解と差分解 (違いが見て分かるはずはないが…)

4　1 次元熱方程式

4.1　Dirichlet 境界条件

4.1.1　陽解法

```
% heat1d_e.m -- 空間 1 次元熱方程式，同次 Dirichlet 境界条件
% curl -O https://m-katsurada.sakura.ne.jp/program/fdm/heat2d.m
% https://m-katsurada.sakura.ne.jp/labo/text/new-intro-matlab.pdf

% 区間
a=0
b=1
% N 等分
N=100
% N 等分点
x=linspace(a,b,N+1);
% 初期値 (実は u=min(x,1-x); ですむ)
for i=0:N
    if x(i+1)<0.5
        u(i+1)=x(i+1);
    else
        u(i+1)=1-x(i+1);
    end
end
% 初期値のグラフを描いて 1 秒待つ
%fig=figure % 新しく図のウィンドウを出すかどうか
plot(x,u)
fig=gcf; figure(fig) % こうすると visible になる
axis([0 1 -0.1 1.1])
title('heat equation')
pause(1)

% 差分法の準備
h=(b-a)/N
lambda=0.5
tau=lambda*h*h
tMax=1
nMax=tMax/tau
newu=zeros(1,N+1);
% t が dt 増えるごとに描画
dt = 0.01;
nskip = round(dt / tau);
for n=1:nMax
    newu(2:N)=(1-2*lambda)*u(2:N)+lambda*(u(1:N-1)+u(3:N+1));
    newu(1)=0;
    newu(N+1)=0;
    if mod(n,nskip)==0
      plot(x,newu)
      axis([0 1 -0.1 1.1])
      title(['heat equation, t=', num2str(n*tau, '%4.2f')])
      pause(0.1);
    end
    u=newu;
end
```

4.1.2 陰解法

```
% heat1d_i.m -- 空間 1 次元熱方程式，同次 Dirichlet 境界条件

% 区間
a=0;
b=1;
% N 等分
N=100;
% N 等分点
x=linspace(a,b,N+1);
% 初期値
u=min(x,1-x);
% 初期値のグラフを描いて 1 秒待つ
%fig=figure
plot(x,u);
fig=gcf; figure(fig) % こうすると visible になる
axis([0 1 -0.1 1.1]);
title('heat equation');
pause(1);

tMax=1;
h=(b-a)/N;
lambda=0.5;
tau=lambda*h*h;
theta=0.5; % Crank-Nicolson
a=sparse((1+2*theta*lambda)*eye(N-1,N-1)-theta*lambda*(diag(ones(N-2,1),1)+diag(ones(N-2,1),-1)));

nMax=tMax/tau;
% t が dt 増えるごとに描画
dt = 0.01;
nskip = round(dt / tau);
for n=1:nMax
    f=(1-2*(1-theta)*lambda)*u(2:N)+(1-theta)*lambda*(u(1:N-1)+u(3:N+1));
    u(2:N)=a\f';
    u(1)=0;
    u(N+1)=0;
    if mod(n,nskip)==0
      plot(x,u);
      axis([0 1 -0.1 1.1]);
      title(['heat equation, t=', num2str(tau*n, '%4.2f')])
      pause(0.1);
    end
end
```

4.2 Neumann 境界条件

5 2 次元熱方程式

5.1 Dirichlet 境界条件

「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」[2]

「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」[3] という解説を書いてる。

[2] https://m-katsurada.sakura.ne.jp/labo/text/heat2d.pdf

[3] https://m-katsurada.sakura.ne.jp/labo/text/heat2d-nonhomo.pdf

以下は、そこで説明した非同次 Dirichlet 境界値問題のプログラムである。

heat2d_nh.m

```
% heat2d_nh.m
% 長方形領域における熱方程式 u_t=△ u（非同次 Dirichlet 境界条件）を差分法で解く
% 入手 curl -O https://m-katsurada.sakura.ne.jp/program/fdm/heat2d_nh.m
% 解説 https://m-katsurada.sakura.ne.jp/labo/text/heat-nonhomo.pdf
% 一応の完成をして、heat2d_nh.m として公開した（2024/8/17)。少し推敲（2024/8/24)。

function heat2d_nh
  % Dirichlet 境界条件の場合の係数行列
  function [A,B]=heat2d_mat(Nx,Ny,lambdax,lambday,theta)
    Ix=speye(Nx-1,Nx-1);
    Iy=speye(Ny-1,Ny-1);
    vx=ones(Nx-2,1);
    Jx=sparse(diag(vx,1)+diag(vx,-1));
    vy=ones(Ny-2,1);
    Jy=sparse(diag(vy,1)+diag(vy,-1));
    Kx=2*Ix-Jx;
    Ky=2*Iy-Jy;
    % row major (y changes first, l=j+(Ny-1)*i)
    A=kron(Ix,Iy)+theta*lambday*kron(Ix,Ky)+theta*lambdax*kron(Kx,Iy);
    B=kron(Ix,Iy)-(1-theta)*lambday*kron(Ix,Ky)-(1-theta)*lambdax*kron(Kx,Iy);
    % column major (x chandes first, l=i+(Nx-1)*j)
    % A=kron(Iy,Ix)+theta*lambdax*kron(Iy,Kx)+theta*lambday*kron(Ky,Ix);
    % B=kron(Iy,Ix)-(1-theta)*lambdax*kron(Iy,Kx)-(1-theta)*lambday*kron(Ky,Ix);
  end

  % 長方形領域 (a,b) × (c,d)
  a=0; b=1; c=0; d=1;
  % Dirichlet 境界値 b() 調和関数を選ぶとこれが平衡解（定常解）
  function val = dbv(x,y)
    val = (x-0.5).*(x-0.5)-(y-0.5).*(y-0.5);
  end
  % 初期値（平衡解を境界値が 0 のもので摂動する）
  function val = iv(x,y)
    val = dbv(x,y) + sin(pi*x).*sin(2*pi*y);
  end
  % 厳密解が分かる
  function val = exact(x,y,t)
    val = dbv(x,y) + exp(- 5 * pi * pi * t) * sin(pi*x).*sin(2*pi*y);
  end
  % 長方形の下の辺での b()
  function val = bbottom(x)
    val = dbv(x,c);
  end
  % 長方形の上の辺での b()
  function val = btop(x)
    val = dbv(x,d);
  end
  % 長方形の左の辺での b()
  function val = bleft(y)
    val = dbv(a, y);
  end
  % 長方形の右の辺での b()
  function val = bright(y)
    val = dbv(b, y);
  end
  % 誤差を測るためのノルム（最大値ノルム）
  function val = mynorm(a)
    [m,n]=size(a);
    val = norm(reshape(a,m*n,1),Inf);
```

```
end
% 格子への分割
Nx=100;
Ny=100;
hx=(b-a)/Nx;
hy=(d-c)/Ny;
theta=0.5;
tau=0.5/(1/hx^2+1/hy^2); % 陽解法の場合のギリギリの刻み (陰解法なので大きくても OK)
lambdax=tau/hx^2;
lambday=tau/hy^2;
disp(sprintf('Nx=%d, Ny=%d, hx=%f, hy=%f, tau=%e, λ x=%f, λ y=%f', ...
      Nx, Ny, hx, hy, tau, lambdax, lambday));

% 差分方程式 A U^{n+1}=B U^n の行列
[A,B]=heat2d_mat(Nx,Ny,lambdax,lambday,theta);
% LU 分解
[AL,AU,ap]=lu(A,'vector');

% 格子点の座標ベクトル x=(x_1,x_2,...,x_{Nx+1}), y=(y_1,y_2,...,y_{Ny+1})
X=linspace(a,b,Nx+1)';
Y=linspace(c,d,Ny+1)';
% 格子点の x,y 座標の配列 X={X_{ij}}, Y={Y_{ij}}
[x,y]=meshgrid(X,Y);
% 初期値
u= iv(x,y);
% 極限 (境界値 b の式が調和ならば)
ulimit = dbv(x,y);

% 初期値のグラフを描く
disp(' 初期値のグラフ')
meshc(x,y,u); axis([a b c d -2 2]); drawnow;

U=reshape(u(2:Ny,2:Nx),(Nx-1)*(Ny-1),1);

% 境界値 左の辺、右の辺
blvector=bleft(Y(2:Ny));
brvector=bright(Y(2:Ny));
lindex=1:Ny-1;
rindex=(Nx-2)*(Ny-1)+1:(Nx-1)*(Ny-1);
% 境界値 下の辺、上の辺
bbvector=bbottom(X(2:Nx));
btvector=btop(X(2:Nx));
bindex=1:Ny-1:(Nx-1)*(Ny-1);
tindex=Ny-1:Ny-1:(Nx-1)*(Ny-1);

% どこまで計算するか
Tmax=0.5;
Nmax = ceil(Tmax/tau);
% グラフを表示する時間の間隔
dt=0.005;
skip=dt/tau;
%
disp(' ループに入ります。何かキーを押してください。')
pause
t=0;
maxerror=0;
for n=0:Nmax
  % 連立 1 次方程式の右辺の計算
  U=B*U;
  % 移項
  U(lindex)=U(lindex)+lambdax*blvector;
  U(rindex)=U(rindex)+lambdax*brvector;
  U(bindex)=U(bindex)+lambday*bbvector;
```

```
    U(tindex)=U(tindex)+lambday*btvector;
    % 連立 1 次方程式を解いて差分解を求める
    U=AU\(AL\U(ap,:));
    t=(n+1)*tau;
    if mod(n,skip)==0
      u(2:Ny,2:Nx)=reshape(U,Ny-1,Nx-1);
      meshc(x,y,u); axis([a b c d -2 2]); drawnow;
      error = mynorm(u-exact(x,y,t));
      if error > maxerror maxerror = error; end
      disp(sprintf('t=%f, ||error||=%e, ||u||=%e, ||u∞||=%e',...
     t, error, mynorm(u), mynorm(ulimit)));
    end
  end
  display(sprintf('||u(・,t)-u(・,∞)||=%e', mynorm(u-ulimit)));
  display(sprintf('Nx=%d,Ny=%d,max error=%e', Nx, Ny, maxerror));
end
```

```
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/heat2d_nh.m
cp -p heat2d_nh.m ~/Documents/MATLAB
```

```
>> heat2d_nh

>> heat2d_nh(100,100)
```

5.2 Neumann 境界条件

「Neumann 境界条件下の熱方程式に対する差分法」[4]
「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」[5]

heat2n_nh.m

```
% heat2n_nh.m
% 長方形領域における熱方程式 u_t=△u（非同次 Neumann 境界条件）を差分法で解く
% 2024/8/24, written by mk.
% 入手 curl -O https://m-katsurada.sakura.ne.jp/program/fdm/heat2n_nh.m
% 解説 https://m-katsurada.sakura.ne.jp/labo/text/heat2n-nonhomo.pdf

function heat2n_nh
  function [A,B]=heat2n_mat0(Nx,Ny,lambdax,lambday,theta)
    Ix=speye(Nx+1,Nx+1);
    Iy=speye(Ny+1,Ny+1);
    vx=ones(Nx,1);
    Jx=sparse(diag(vx,1)+diag(vx,-1));
    Jx(1,2)=2; Jx(Nx+1,Nx)=2;
    vy=ones(Ny,1);
    Jy=sparse(diag(vy,1)+diag(vy,-1));
    Jy(1,2)=2; Jy(Ny+1,Ny)=2;
    Kx=2*Ix-Jx;
    Ky=2*Iy-Jy;
  % column first
    A=kron(Ix,Iy)+theta*lambday*kron(Ix,Ky)+theta*lambdax*kron(Kx,Iy);
    B=kron(Ix,Iy)-(1-theta)*lambday*kron(Ix,Ky)-(1-theta)*lambdax*kron(Kx,Iy);
  % row first
  %  A=kron(Iy,Ix)+theta*lambdax*kron(Iy,Kx)+theta*lambday*kron(Ky,Ix);
```

[4] https://m-katsurada.sakura.ne.jp/labo/text/heat2n.pdf
[5] https://m-katsurada.sakura.ne.jp/labo/text/heat2n-nonhomo.pdf

```
%  B=kron(Iy,Ix)-(1-theta)*lambdax*kron(Iy,Kx)-(1-theta)*lambday*kron(Ky,Ix);
end

% 長方形領域 (a,b) × (c,d)
a=0; b=1; c=0; d=1;
% Neumann 境界値 b() 調和関数を選ぶとこれが平衡解 (定常解)
function val = harmonic(x,y)
  val = (x-0.5).*(y-0.5);
end
% 初期値 (平衡解を境界値が 0 のもので摂動する)
function val = iv(x,y)
  val = harmonic(x,y) + cos(pi*x).*cos(pi*y);
end
% 厳密解が分かる
function val = exact(x,y,t)
  val = harmonic(x,y) + exp(- 2 * pi * pi * t) * cos(pi*x).*cos(pi*y);
end
% 長方形の下の辺での b_b() … -∂ u/∂ y(x,0,t)
function val = b_b(x)
  val = 0.5 - x;
end
% 長方形の上の辺での b_b() … ∂ u/∂ y(x,1,t)
function val = b_t(x)
  val = x - 0.5;
end
% 長方形の左の辺での b_l() … -∂ u/∂ x(0,y,t)
function val = b_l(y)
  val = 0.5 - y;
end
% 長方形の右の辺での b_r() … ∂ u/∂ x(1,y,t)
function val = b_r(y)
  val = y - 0.5;
end
% 誤差を測るためのノルム (最大値ノルム)
function val = mynorm(a)
  [m,n]=size(a);
  val = norm(reshape(a,m*n,1),Inf);
end
% 格子への分割
Nx=50;
Ny=50;
hx=(b-a)/Nx;
hy=(d-c)/Ny;
theta=0.5;
tau=0.5/(1/hx^2+1/hy^2); % 陽解法の場合のギリギリの刻み (もっと大きくても OK)
lambdax=tau/hx^2;
lambday=tau/hy^2;
display(sprintf("Nx=%d, Ny=%d, hx=%f, hy=%f, tau=%e, λ x=%f, λ y=%f", ...
  Nx, Ny, hx, hy, tau, lambdax, lambday));

% 差分方程式 A U^{n+1}=B U^n の行列
[A,B]=heat2n_mat0(Nx,Ny,lambdax,lambday,theta);
% 対称にするためのスケーリング用の行列
Sx=sparse(diag([1/2;ones(Nx-1,1);1/2]));
Sy=sparse(diag([1/2;ones(Ny-1,1);1/2]));
S=kron(Sy,Sx);
% A を対称行列に直す
A=S*A;
% 対称かどうかチェックする
er=A-A';
if norm(full(er),inf)>=1
  disp('A が対称でない。')
  pause
```

```
end
% LU 分解
[AL,AU,ap]=lu(A,'vector');

% 格子点の座標ベクトル x=(x_1,x_2,...,x_{Nx+1}), y=(y_1,y_2,...,y_{Ny+1})
X=linspace(a,b,Nx+1)';
Y=linspace(c,d,Ny+1)';
% 格子点の x,y 座標の配列 X={X_{ij}}, Y={Y_{ij}}
[x,y]=meshgrid(X,Y);
% 初期値
u= iv(x,y);
% 極限
ulimit = harmonic(x,y);

% 初期値のグラフを描く
disp(' 初期値のグラフ')
meshc(x,y,u);
axis([a b c d -4 4]);
drawnow;
disp(' 何かキーを押して下さい')
pause

% 境界値 左の辺、右の辺
blvector=b_l(Y(1:Ny+1));
brvector=b_r(Y(1:Ny+1));
% 境界値 下の辺、上の辺
bindex=1:Ny+1:(Nx+1)*(Ny+1);
tindex=Ny+1:Ny+1:(Nx+1)*(Ny+1);
bbvector=b_b(X(1:Nx+1));
btvector=b_t(X(1:Nx+1));

% どこまで計算するか
Tmax=0.5;
Nmax = ceil(Tmax/tau);
% グラフを表示する時間の間隔
dt=0.005;
skip=dt/tau;

U=reshape(u,(Nx+1)*(Ny+1),1);

disp(' ループに入ります。何かキーを押してください。')
t=0;
maxerror=0;
for n=0:Nmax
  % 注: これから計算するのが時刻 t での値 (最初が t=tau)
  U=B*U;
  % 移項
  U(1:Ny+1)=U(1:Ny+1)+2*hx*lambdax*blvector;
  U(end-(Ny+1)+1:end)=U(end-(Ny+1)+1:end)+2*hx*lambdax*brvector;
  U(bindex)=U(bindex)+2*hy*lambday*bbvector;
  U(tindex)=U(tindex)+2*hy*lambday*btvector;
  U=S*U;
  % 連立一次方程式を解いて差分解を求める
  U=AU\(AL\U(ap,:));
  t=(n+1)*tau;
  % 計算結果の表示
  if mod(n,skip)==0
    u=reshape(U,Ny+1,Nx+1);
    meshc(x,y,u); axis([a b c d -1 1]); drawnow;
    % 誤差の表示
    error = mynorm(u-exact(x,y,t));
    if error > maxerror maxerror = error; end
    display(sprintf("t=%f, ||error||=%e, ||u||=%e, ||u ∞||=%e",...
```

```
        t, error, mynorm(u), mynorm(ulimit)));
    end
  end
  display(sprintf("max ||u(・,t)-u(・,∞)||=%e", mynorm(u-ulimit)));
  display(sprintf("Nx=%d,Ny=%d,max error=%e", Nx, Ny, maxerror));
end
```

```
curl -O https://m-katsurada.sakura.ne.jp/program/fdm/heat2n_nh.m
cp -p heat2n_nh.m ~/Documents/MATLAB
```

```
>> heat2n_nh

>> heat2n_nh(100,100)
```

6 固有値問題

6.1 はじめに状況説明

最初に重要なことを書く。

長方形領域 $\Omega = (0, W) \times (0, H)$ で、Dirichlet 条件、Neumann 境界条件下の Laplacian の固有値問題を数値計算で解く意味はほとんどない。

厳密解はもちろん (有名で、多くのテキストに載っている)、差分法で近似した場合の固有関数・固有値も簡単な式で求まるためである。

特に固有関数は、微分作用素のそれと一致する。

- Dirichlet 境界条件の場合は $\sin \frac{m\pi x}{W} \sin \frac{n\pi y}{H}$ $(m \in \mathbb{N},\, n \in \mathbb{N})$.
- Neumann 境界条件の場合は $\cos \frac{m\pi x}{W} \cos \frac{n\pi y}{H}$ $(m \in \mathbb{N} \cup \{0\},\, n \in \mathbb{N} \cup \{0\})$.

実を言うと、私はそのことに気づかず、学部卒研の課題として与えてしまったことがある。「先生、誤差がほとんど 0 です。」という (本当は当たり前だけれど、そのときの私には) 驚愕の報告があって大騒ぎだった。20 年以上前の笑い話。

まとめたものを書いたっけ？？

- 1 次元の場合については、桂田 [1] にある程度詳しく書いてある。
- 2 次元の場合は 1 次元の場合のテンソル積の議論になる。いつか気が向いたら書きます (約束はしかねます)。

というわけで Laplacian (Laplace 作用素) の固有値問題を差分法で解くのは今ひとつ面白みがない (もちろん複雑な形状の領域を持ってくれば難しくなるが、今度は逆に難しすぎる)。

少し唐突かもしれないが、重調和作用素 $\triangle^2 = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)^2$ の場合が面白い。実は私のゼミで、卒研や修士論文のテーマで取り組んだ人が何人かいる。これはいわゆる Chladni 図形 (クラドニ図形) のシミュレーションとみなすことが出来る。長方形領域でもそんなに簡単ではない。テキストによっては “単純支持” という境界条件を与えて、変数分離で厳密解を求めてしまっているものがあるが、より自然な設定である (だから重要と考えられる) free edge の場合は、変数分離がうまくいかず、境界条件の方程式がかなり複雑になり、差分法のプログラ

ムを書くのも一苦労する。でもなんとか出来た、と言うのが以下の平野さんの修論である (係数行列の導出に半年近くかかったという力作)。

一方、この Chladni 図形の問題は、Ritz の方法で有名な Walter Ritz の記念碑的な論文 [5] で、数値計算により解かれたことで有名である (コンピューターのない時代の、ほとんど神話のような出来事)。その方法は有限要素法の基礎にもなっている。固有値問題の数値計算には有限要素法がおすすめである。正多角形板の Chladni 図形は古くから色々実験的に調べられてきたが、それについて有限要素法でシミュレーションしたのが以下の遠藤さんの修論である。

1. 平野裕輝 『正方形領域における重調和作用素の固有値問題 — 差分法によるクラドニ図形の解析』[6]　MATLAB プログラムが含まれている。
2. 遠藤小欽「正多角形板の Chladni 図形」[7]　FreeFem++プログラムが含まれている。(MATLAB とはあまり関係がない。)

(間にもう一人いるのだけれど、公開承諾をもらい損ねたような記憶が…失敗したなあ。)

6.2　正方形板の Chladni 図形

(準備中)

A　MATLAB メモ

A.1　基本

- コマンドウィンドウのプロンプト >>　に対してコマンドを入力する。

```
>> 1+1

ans = 2
```

- 行単位の編集機能がある。コマンド履歴 (ヒストリー) があり、呼び出せる。タブ入力により補完が出来る。
- doc 名前 で MATLAB 組み込みの関数や変数についてのヘルプを呼び出せる。
- 変数名=式 で変数に値を代入し、表示する。変数名=式; とコロンをつけると変数に値を代入するが、値は表示しない。

```
>> a=1+1

a =
     2

>> a=1+1;
>> a
a =
     2
```

[6] http://nalab.mind.meiji.ac.jp/~mk/labo/report/open/2011-hirano.pdf
[7] http://nalab.mind.meiji.ac.jp/~mk/gensyou2019/2016-endo.pdf

- who で使っている変数名一覧を表示する。whos で使っている変数の情報 (型、サイズ) 一覧を表示する。
- clear ですべての変数を消去する。clear 変数名 で指定した変数を消去する。

```
>> a=1+1

a =
     2

>> who

変数:

a

>> whos
  Name      Size             Bytes  Class     Attributes

  a         1x1                  8  double

>> clear
>> whos
>>
```

- プログラム・ファイルの編集は (テキスト・エディターを使うことも出来るが) edit 関数名 あるいはedit スクリプト名
- パーセント記号 % から行末までは注釈になる。
- 行を継続したいときは行末に継続用の記号 ... (3 つ以上の連続ピリオド) をおく。

A.2 zeros(), ones(), eye(), rand()

- zeros(m,n), ones(m,n), eye(m,n), rand(m,n) はそれぞれ、m 行 n 列の零行列、全ての成分が 1 の行列、対角成分が 0 でそれ以外が 0 の行列、乱数行列を返す。zeros(n) は zeros(n,n) と同じ (ones(), eye(), rand() も同様)。
- ' は行列・ベクトルのエルミート転置、.' は転置を表す。x と y が同じ次元のたてベクトルであるとき、y'*x は内積に等しい (dot(x,y) と同じ)。

A.3 行列とベクトルの基本

カンマ , と空白は、ともに次の列に移ることを意味する。
セミコロン ; と改行は、ともに次の行に移ることを意味する。

```
>> [1 2 3]

ans =
     1     2     3

>> [1,2,3]

ans =
     1     2     3

>> [1;2;3]

ans =
     1
     2
     3

>> [1
2
3]

ans =
     1
     2
     3

>> [1,2;3,4]

ans =
     1     2
     3     4

>> >> [1,2
3,4
5,6]

ans =
     1     2
     3     4
     5     6

>>
```

- 配列のサイズは size() で求まる。

```
>> x=1:2

x =
     1     2     3

>> size(x)

ans =
     1     3

>> [m,n]=size(x)

m =
     1

n =
     3

>> a=rand(3,2)

a =
    0.9649    0.9572
    0.1576    0.4854
    0.9706    0.8003

>> [m,n]=size(a)

m =
     3

n =
     2
```

- 配列の添字は 1 から始まる。
- 1 次元配列 a の第 i 成分は a(i) で表す。
- 2 次元配列 A の第 (i,j) 成分は A(i,j) で表す。

行列とベクトル よく使う機能早見表

`[m,n]=size(a)`	行列 a のサイズ (行の数、列の数)
`eye(m,n)`	m 行 n 列の単位行列 ((i,j) 成分が δ_{ij})
`zeros(m,n)`	m 行 n 列の零行列
`ones(m,n)`	(m 行 n 列の) 成分がすべて 1 の行列
`rand(m,n)`	(m 行 n 列の) 成分が乱数の行列
`diag()`	対角行列 (を少しずらした行列)
`a'`	a の Hermite 共役 (実行列、実ベクトルの場合転置)
`a.'`	a の転置
`det(a)`	a の行列式
`inv(a)`	a の逆行列 (大きいときは使用を控えるべき)
`y'*x`	縦ベクトル x, y の内積
`norm(x)`	x のノルム (成分の絶対値の二乗の和の平方根)
`tril(a)`	a の下三角部分
`triu(a)`	a の上三角部分
`a(i,:)`	a の第 i 行ベクトル
`a(:,j)`	a の第 j 列ベクトル
`a(i1:i2,j1:j2)`	a の第 i1〜i2 行、第 j1〜j2 列の部分のブロック

A.4 : に慣れる

```
>> 1:10

ans =
     1     2     3     4     5     6     7     8     9    10

>> 0:0.2:1

ans =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
```

$x_0 \le x_1$ とするとき、x_0:x_1 は、n を $x_0+n \le x_1,\ x_0+n+1 > x_1$ となる整数として、[x_0, x_0+1, $\cdots$, x_0+n] という横ベクトルを返す。

$x_0 \le x_1,\ \Delta x > 0$ とするとき、x_0:Δx:x_1 は、n を $x_0+n\Delta x \le x_1,\ x_0+(n+1)\Delta x > x_1$ となる整数として、[x_0, $x_0+\Delta x$, $x_0+2\Delta x$,$\cdots$, $x_0+n\Delta x$] という横ベクトルを返す。

$x_0 \ge x_1,\ \Delta x < 0$ とするとき、x_0:Δx:x_1 は、n を $x_0+n\Delta x \ge x_1,\ x_0+(n+1)\Delta x < x_1$ となる整数として、[x_0,$x_0+\Delta x$,$x_0+2\Delta x$,...,$x_0+n\Delta x$] という横ベクトルを返す。

行列 a に対して、a(:) は 1 次元化したベクトルを返す。

```
>> a=[1,2;3,4]

a =
     1     2
     3     4

>> v=a(:)

v =
     1
     3
     2
     4

>> b=reshape(v,2,2)

b =
     1     2
     3     4
```

A.5 添字ベクトル (?) の利用

ベクトル、行列に対して、添字の代わりに “添字ベクトル” を指定することで、色々なことが出来る。

- ブロックの指定。例えば行列 A の $3 \sim 5$ 行、$4 \sim 8$ 列からなるブロックは 、A(3:5,4:8) で表せる。

```
>> a=(1:9)'*(1:9)

a =
     1     2     3     4     5     6     7     8     9
     2     4     6     8    10    12    14    16    18
     3     6     9    12    15    18    21    24    27
     4     8    12    16    20    24    28    32    36
     5    10    15    20    25    30    35    40    45
     6    12    18    24    30    36    42    48    54
     7    14    21    28    35    42    49    56    63
     8    16    24    32    40    48    56    64    72
     9    18    27    36    45    54    63    72    81

>> a(3:5,4:8)

ans =
    12    15    18    21    24
    16    20    24    28    32
    20    25    30    35    40

>>
```

- ブロックの特別な場合として、第 i 行ベクトルは A(i,:), 第 j 列ベクトルは A(:,j)

```
>> a=rand(4,3)

a =
    0.1419    0.9595    0.9340
    0.4218    0.6557    0.6787
    0.9157    0.0357    0.7577
    0.7922    0.8491    0.7431

>> a(2,:)

ans =
    0.4218    0.6557    0.6787

>> a(:,3)

ans =
    0.9340
    0.6787
    0.7577
    0.7431

>>
```

- 行の置換、列の置換なども行える。

```
>> a(9:-1:1,:)

ans =
     9    18    27    36    45    54    63    72    81
     8    16    24    32    40    48    56    64    72
     7    14    21    28    35    42    49    56    63
     6    12    18    24    30    36    42    48    54
     5    10    15    20    25    30    35    40    45
     4     8    12    16    20    24    28    32    36
     3     6     9    12    15    18    21    24    27
     2     4     6     8    10    12    14    16    18
     1     2     3     4     5     6     7     8     9

>> a([1 3 2 6 5 4 9 8 7],:)

ans =
     1     2     3     4     5     6     7     8     9
     3     6     9    12    15    18    21    24    27
     2     4     6     8    10    12    14    16    18
     6    12    18    24    30    36    42    48    54
     5    10    15    20    25    30    35    40    45
     4     8    12    16    20    24    28    32    36
     9    18    27    36    45    54    63    72    81
     8    16    24    32    40    48    56    64    72
     7    14    21    28    35    42    49    56    63
```

- 配列の最後の添字は end で表せる。例えばベクトル x の最初と最後の成分を除くには x=x(2:end-1) とすれば良い。

```
>> x=1:10

x =

     1     2     3     4     5     6     7     8     9    10

>> x(2:end-1)

ans =

     2     3     4     5     6     7     8     9

>> x=(1:10)'

x =

     1
     2
     3
     4
     5
     6
     7
     8
     9
    10

>> x=x(2:end-1)

x =

     2
     3
     4
     5
     6
     7
     8
     9

>>
```

A.6 lu()

A が与えられたとき、$LU = PA$ を満たす置換行列 P, 下三角行列 L, 上三角行列 U を求めることを A を LU 分解すると言う。

```
a=hilb(4)
```

とすると

```
a =
    1.0000    0.5000    0.3333    0.2500
    0.5000    0.3333    0.2500    0.2000
    0.3333    0.2500    0.2000    0.1667
    0.2500    0.2000    0.1667    0.1429
```

```
[L U P]=lu(a)
```

とすると

```
L =
    1.0000         0         0         0
    0.3333    1.0000         0         0
    0.5000    1.0000    1.0000         0
    0.2500    0.9000   -0.6000    1.0000

U =
    1.0000    0.5000    0.3333    0.2500
         0    0.0833    0.0889    0.0833
         0         0   -0.0056   -0.0083
         0         0         0    0.0004

P =
     1     0     0     0
     0     0     1     0
     0     1     0     0
     0     0     0     1
```

念のため

```
norm(L*U-P*a)
```

とすると

```
ans =
   3.9252e-17
```

[l u]=lu(a) とすると　置換行列を省いて [l u]=lu(a) とすると、l には PL が、u には U が代入される。

```
[L U P]=lu(a);
[L0 U0]=lu(a)
```

とすると

```
L0 =
    1.0000         0         0         0
    0.5000    1.0000    1.0000         0
    0.3333    1.0000         0         0
    0.2500    0.9000   -0.6000    1.0000

U0 =
    1.0000    0.5000    0.3333    0.2500
         0    0.0833    0.0889    0.0833
         0         0   -0.0056   -0.0083
         0         0         0    0.0004
```

本当に L0 $= PL$, U0 $= U$ かチェックする。

```
norm(U0-U)
norm(L0-P*L)
```

どちらも 0 になる。

```
[L U p]=lu(a,'vector')
```

とすると

```
L =
    1.0000         0         0         0
    0.3333    1.0000         0         0
    0.5000    1.0000    1.0000         0
    0.2500    0.9000   -0.6000    1.0000

U =
    1.0000    0.5000    0.3333    0.2500
         0    0.0833    0.0889    0.0833
         0         0   -0.0056   -0.0083
         0         0         0    0.0004

p =
     1     3     2     4
```

p は置換行列 P の “情報を持った” ベクトルである。

- ベクトル c に対して、Pc を求めるには P*c とする代わりに、c(p) とすれば良い。
- 行列 C に対して、PC を求めるには P*C とする代わりに、C(p,:) とすれば良い。

$PA = LU$ となっているとき、

$$Ax = b \quad \Leftrightarrow \quad PAx = Pb \quad \Leftrightarrow \quad LUx = Pb$$

であるから、x を求めるには、U\(L\(P*b))、あるいは U\(L\b(p)) とする。

解が分かりやすい連立１次方程式を用意しよう。

```
x=(1:4)';
b=a*x;
```

解は、もちろん

```
a\b
```

とすれば求まるが、[L U P]=lu(a); で求めた L, U, P を使うには

```
U\(L\(P*b))
```

とすれば良く、[L U p]=lu(a,'vector'); で求めた L, U, p を使うには

```
U\(L\b(p))
```

とすれば良い。

```
>> x=(1:4)'

x =
     1
     2
     3
     4

>> b=a*x;
>> a\b

ans =
    1.0000
    2.0000
    3.0000
    4.0000

>> U\(L\(P*b))

ans =
    1.0000
    2.0000
    3.0000
    4.0000

>> U\(L\b(p))

ans =
    1.0000
    2.0000
    3.0000
    4.0000
```

A.7 diag()

- diag(ベクトル) とすると、ベクトルを対角成分に埋め込んだ正方行列を返す。

```
>> diag(1:5)

ans =
     1     0     0     0     0
     0     2     0     0     0
     0     0     3     0     0
     0     0     0     4     0
     0     0     0     0     5
```

- diag(行列) とすると、行列の対角成分からなるベクトルを返す。

```
>> a=(1:9)'*(1:9);
>> diag(a)

ans =
     1
     4
     9
    16
    25
    36
    49
    64
    81
```

ゆえに diag(diag(行列)) とすると、行列の対角成分以外を 0 クリアした対角行列が得られる。

```
>> a=(1:9)'*(1:9);
>> diag(diag(a))

ans =
     1     0     0     0     0     0     0     0     0
     0     4     0     0     0     0     0     0     0
     0     0     9     0     0     0     0     0     0
     0     0     0    16     0     0     0     0     0
     0     0     0     0    25     0     0     0     0
     0     0     0     0     0    36     0     0     0
     0     0     0     0     0     0    49     0     0
     0     0     0     0     0     0     0    64     0
     0     0     0     0     0     0     0     0    81
```

A.8 kron()

行列 $A \in \mathbb{C}^{k\times\ell}$, $B \in \mathbb{C}^{m\times n}$ の Kronecker 積 $A \otimes B$ は

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1\ell}B \\ \vdots & \ddots & \vdots \\ a_{k1}B & \cdots & a_{k\ell}B \end{pmatrix} \quad \text{(ブロック行列の記法)}$$

で定義される。

kron(a,b) で $A \otimes B$ が計算できる。

A.9 linspace()

linspace(a,b,m) は、$[a, b]$ を $m-1$ 等分した点 (全部で m 個) の座標を並べた横ベクトルを返す。

n 等分点が欲しければ linspace(a,b,n+1) とする。

A.10 meshgrid()

```
>> nx=5; ny=3;
>> X=linspace(0,1,nx+1)

X =
         0    0.2000    0.4000    0.6000    0.8000    1.0000

>> Y=linspace(2,3,ny+1)

Y =
    2.0000    2.3333    2.6667    3.0000

>> [x,y]=meshgrid(X,Y)

x =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
         0    0.2000    0.4000    0.6000    0.8000    1.0000
         0    0.2000    0.4000    0.6000    0.8000    1.0000
         0    0.2000    0.4000    0.6000    0.8000    1.0000

y =
    2.0000    2.0000    2.0000    2.0000    2.0000    2.0000
    2.3333    2.3333    2.3333    2.3333    2.3333    2.3333
    2.6667    2.6667    2.6667    2.6667    2.6667    2.6667
    3.0000    3.0000    3.0000    3.0000    3.0000    3.0000

>> whos
  Name      Size             Bytes  Class     Attributes

  X         1x6                 48  double
  Y         1x4                 32  double
  nx        1x1                  8  double
  ny        1x1                  8  double
  x         4x6                192  double
  y         4x6                192  double
```

X$= [x_1, x_2, \ldots, x_{N_x+1}] \in \mathbb{R}^{N_x+1}$, Y$= [y_1, y_2, \ldots, y_{N_y+1}] \in \mathbb{R}^{N_y+1}$ とするとき、`[x,y]=meshgrid(X,Y)` とすると、`x` と `y` は $\mathbb{R}^{(N_y+1)\times(N_x+1)}$ の要素となる。`x(j,i)` は x_i, `y(j,i)` は y_j である。

MATLAB の 2 次元配列は column first で並んでいるので、例えば `x` の成分は、メモリー内で、

`x(1,1)`, `x(2,1)`, `x(3,1)`, `x(4,1)`,
`x(1,2)`, `x(2,2)`, `x(3,2)`, `x(4,2)`,
...
`x(1,5)`, `x(2,5)`, `x(3,5)`, `x(4,5)`,
`x(1,6)`, `x(2,6)`, `x(3,6)`, `x(4,6)`

と並んでいる。

```
>> x(:)
ans =
         0
         0
         0
         0
    0.2000
    0.2000
    0.2000
    0.2000
    0.4000
    0.4000
    0.4000
    0.4000
    0.6000
    0.6000
    0.6000
    0.6000
    0.8000
    0.8000
    0.8000
    0.8000
    1.0000
    1.0000
    1.0000
    1.0000

>> y(:)'

ans =
  1 列から 9 列
    2.0000    2.3333    2.6667    3.0000    2.0000    2.3333    2.6667    3.0000    2.0000

  10 列から 18 列
    2.3333    2.6667    3.0000    2.0000    2.3333    2.6667    3.0000    2.0000    2.3333

  19 列から 24 列
    2.6667    3.0000    2.0000    2.3333    2.6667    3.0000

>>
```

A.11 meshgrid(m,n) と meshx(x,y,f)

testofmeshgrid.m

```
L=3; H=2;
a=-L; b=L; c=-H; d=H;
nx=30; ny=20;
X=linspace(a,b,nx+1);
Y=linspace(c,d,ny+1);
[x,y]=meshgrid(X,Y);
% f(x,y)=sin(x)sin(2y)
F=sin(x).*sin(2*y);
figure('Name','graph of F')
meshc(x,y,F)
fig1=gcf; figure(fig1)
% g(x,y)=x^2-y^2
G=x.^2-y.^2;
figure('Name','graph of G')
meshc(x,y,G)
fig2=gcf; figure(fig2)
```

A.12 misc: pause(), fprintf()

A.12.1 pause()

指定した秒数だけ待ってくれる C 言語の sleep() が欲しい、と考えたが、MATLAB では pause() という関数がそれをしてくれる。

A.12.2 fprintf()

disp() で表示するのは大雑把すぎる。C 言語の printf() はないけれど、fprintf() というのがある。

```
fprintf('n=%d, t=%5.2f\n', n, t);
```

のようなことが出来る。C プログラマーには説明が不要だろう (printf("n=%d, t=%5.2f\n", n, t); と同じことをする)。

disp(sprintf()) とするのとどう違うのかな。

B 行列を作る

B.1 はじめに

どんな行列 $A = (a_{ij}) \in \mathbb{R}^{m\times n}$ も、成分 a_{ij} を表す式が分かれば

```
a=zeros(m,n);
for i=1:m
  for j=1:n
    a(i,j)=(i,j 成分を表す式);
  end
end
```

で作成出来る (最初に zeros() を使うのが一つの定跡的なテクニック)。

しかし、疎行列の場合など、二重ループは出来れば避けたい。

B.2 Laplacian の近似

本文中に書いたことで再掲ということになる (本文の方は問題に重点があり、こちらは文法や MATLAB の命令の方に重点がある)。

以下、Dirichlet 境界条件の場合のコードを中心に示す。

B.2.1 1 次元の場合

$$A = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix} \in \mathbb{R}^{N-1,N-1}, \quad h = \frac{L}{N}$$

は頻出する。

```
h=L/N;
e=ones(N-1,1);
a=spdiags([-e 2*e -e],-1:1,N-1,N-1)/(h*h);
```

ずっと以前は、疎行列関係の命令を知らずに次のように作っていた。

```
h=L/N;
e=ones(N-1,1);
a=(2*eye(N-1,N-1)-diag(e,1)-diag(e,-1))/(h*h);
```

(この後で、遅ればせながら a=sparse(a); としてメモリを節約するという手はあるかも。)

B.2.2 2 次元の場合

長方形領域 $\Omega = (0, W) \times (0, H)$ を横方向に N_x 等分、縦方向に N_y 等分する。

$$h_x = \frac{W}{N_x}, \quad h_y = \frac{H}{N_y},$$

$$x_i = ih_x \quad (i = 0, 1, \cdots, N_x),$$

$$y_j = jh_y \quad (j = 0, 1, \cdots, N_y),$$
$$u_{ij} = u(x_i, y_j) \quad (i = 0, 1, \cdots, N_x;\ j = 0, 1, \cdots, N_y).$$

u_{ij} の近似値 U_{ij} を求めることが目標となることが多い。

例えば Poisson 方程式の Dirichlet 境界値問題

$$-\triangle u = f \quad \text{in } \Omega, \quad u = \phi \quad \text{on } \partial\Omega$$

を解く場合、U_{ij} $(1 \le i \le N_x - 1,\ 1 \le j \le N_y - 1)$ が未知数となる。

$$m := N_x - 1, \quad n := N_y - 1$$

とおく。

連立 1 次方程式を行列とベクトルで表示するためには、U_{ij} を 1 次元的に並べたベクトルを作る必要がある。例えば

$$\boldsymbol{U} = (U_{1,1}, U_{2,1}, \cdots, U_{m,1}, U_{1,2}, U_{2,2}, \cdots, U_{m,2}, U_{1,3}, U_{2,3}, \cdots, U_{m,3}, \cdots, U_{1,n}, U_{2,n}, \cdots, U_{m,n})^T.$$

$\boldsymbol{U}$ の成分を U_ℓ と書くとき

$$U_{i,j} = U_{i+n(j-1)}$$

という式が成り立つことが分かる。

(6)
$$\ell = i + n(j-1)$$

とおくと

$$\ell - 1 = (i-1) + n(j-1), \quad 0 \le i - 1 \le n - 1$$

が成り立つので、$i-1$ は $\ell - 1$ を n で割った余り、$j-1$ は $\ell - 1$ を n で割った商である。ゆえに

(7)
$$U_\ell = U_{i,j}, \quad j = \lfloor (\ell - 1)/n \rfloor + 1, \quad i = \mathrm{mod}(\ell - 1, n) + 1.$$

余談 B.1 例えば C 言語のプログラムならば、ℓ から i, j を求めるために

```
j=(ell-1)/n; i=(ell-1)%n+1;
```

とすれば良い。もっとも、割り算をする必要はあまりない。ℓ をループの制御変数にするのではなく、i と j をループの制御変数にすればよいから。つまり

```
for (ell=1; ell<=(m*n); ell++) {
  i=(ell-1)%n+1;
  j=(ell-1)/n+1;
  U[ell] = i と j の式;
}
```

と書く代わりに

```
for (i=1; i<Nx; i++) { // i<Nx は i<=m と書く方が分かりやすい？
  for (j=1; j<Ny; j++) { // j<Ny は j<=n と書く方が分かりやすい？
    ell=i+n*(j-1);
    U[ell]= i と j の式;
  }
}
```

とすれば良い。 ■

Laplacian $\triangle = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ には、2階微分が現れるが、それを2階中心差分近似すると、$-\triangle u = f$ は

$$-\left[\frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{h_x^2} + \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{h_y^2}\right] = f(x_i, y_j)$$

という差分方程式で置き換えられる。

$$A\boldsymbol{U} = \boldsymbol{f}, \tag{8}$$

$$A = I_n \otimes \left[\frac{1}{h_x^2}(2I_m - J_m)\right] + \left[\frac{1}{h_y^2}(2I_n - J_n)\right] \otimes I_m, \tag{9}$$

$$\boldsymbol{f} = (f_\ell), \quad f_\ell = f(x_i, y_j). \tag{10}$$

```
% W, H, Nx, Ny に値が適当に記憶されているとして
% 例: W=2; H=1; Nx=8; Ny=4;
hx=W/Nx;
hy=H/Ny;
m=Nx-1;
n=Ny-1;
ex=ones(m,1);
ax=spdiags([-ex 2*ex -ex],-1:1,m,m)/(hx*hx);
ey=ones(n,1);
ay=spdiags([-ey 2*ey -ey],-1:1,n,n)/(hy*hy);
Im=speye(m,m);
In=speye(n,n);
a=kron(ay,Im)+kron(In,ax);
```

なお、U_{ij} を並べる際に、(6) でなく

$$U_\ell = U_{i,j}, \quad \ell = m(i-1) + j \tag{11}$$

とする場合もある。むしろそちらの方が多いかもしれない。MATLAB で `meshgrid()` などを使うには、こちらの方が都合が良い (本文の例を参照せよ)。この場合は

$$j = \mathrm{mod}(\ell - 1, m) + 1, \quad i = \lfloor (\ell - 1)/m \rfloor + 1. \tag{12}$$

また連立1次方程式の係数行列は

$$A = I_m \otimes \left[\frac{1}{h_y^2}(2I_n - J_n)\right] + \left[\frac{1}{h_x^2}(2I_m - J_m)\right] \otimes I_n. \tag{13}$$

コードの方は最後の `a=kron(ay,Im)+kron(In,ax);` を

```
a=kron(Im,ay)+kron(ax,In);
```

に置き換えれば良い。

B.3 コピペして試す

(i, j) 成分を指定して行列を作る、という例を一つくらい。

Hilbert 行列 myhilbert.m

```
% うっかり作ってしまったけれど、hilb(n) という関数が標準で用意されている。
function a=myhilbert(n)
  a=zeros(n,n);
  for i=1:n
      for j=1:n
          a(i,j)=1/(i+j-1);
      end
  end
end
```

5 次の Hilbert 行列の逆行列と条件数を求めてみる。

```
edit myhilbert
```

(上のコードを入力して保存した後)

```
h=myhilbert(10)
inv(h)
cond(h)
```

(逆行列の成分はほぼ 10^{12} 程度の大きさになり、条件数もほぼ 1.6×10^{13} 程度になる。— 非常に大きい、つまり悪条件。)

1 次元ラプラシアン、逆行列や LU 分解を求めてみる。

1 次元ラプラシアン (Dirichlet 境界条件)

```
L=1;
N=5;
h=L/N;
e=ones(N-1,1);
a=spdiags([-e 2*e -e],-1:1,N-1,N-1)/(h*h);
full(a)

b=inv(a);
full(b)

[l u]=lu(a);
full(l)
full(u)
```

2 次元ラプラシアン (Dirichlet 境界条件)

```
W=2;
H=1;
Nx=8;
Ny=4;
hx=W/Nx;
hy=H/Ny;
m=Nx-1;
n=Ny-1;
ex=ones(m,1);
ax=spdiags([-ex 2*ex -ex],-1:1,m,m)/(hx*hx);
ey=ones(n,1);
ay=spdiags([-ey 2*ey -ey],-1:1,n,n)/(hy*hy);
Im=speye(m,m);
In=speye(n,n);
a=kron(ay,Im)+kron(In,ax);
full(a)
```

```
ans =

    64   -16     0     0     0     0     0   -16     0     0     0     0     0     0     0     0     0     0     0     0     0
   -16    64   -16     0     0     0     0     0   -16     0     0     0     0     0     0     0     0     0     0     0     0
     0   -16    64   -16     0     0     0     0     0   -16     0     0     0     0     0     0     0     0     0     0     0
     0     0   -16    64   -16     0     0     0     0     0   -16     0     0     0     0     0     0     0     0     0     0
     0     0     0   -16    64   -16     0     0     0     0     0   -16     0     0     0     0     0     0     0     0     0
     0     0     0     0   -16    64   -16     0     0     0     0     0   -16     0     0     0     0     0     0     0     0
     0     0     0     0     0   -16    64     0     0     0     0     0     0   -16     0     0     0     0     0     0     0
   -16     0     0     0     0     0     0    64   -16     0     0     0     0     0   -16     0     0     0     0     0     0
     0   -16     0     0     0     0     0   -16    64   -16     0     0     0     0     0   -16     0     0     0     0     0
     0     0   -16     0     0     0     0     0   -16    64   -16     0     0     0     0     0   -16     0     0     0     0
     0     0     0   -16     0     0     0     0     0   -16    64   -16     0     0     0     0     0   -16     0     0     0
     0     0     0     0   -16     0     0     0     0     0   -16    64   -16     0     0     0     0     0   -16     0     0
     0     0     0     0     0   -16     0     0     0     0     0   -16    64   -16     0     0     0     0     0   -16     0
     0     0     0     0     0     0   -16     0     0     0     0     0   -16    64     0     0     0     0     0     0   -16
     0     0     0     0     0     0     0   -16     0     0     0     0     0     0    64   -16     0     0     0     0     0
     0     0     0     0     0     0     0     0   -16     0     0     0     0     0   -16    64   -16     0     0     0     0
     0     0     0     0     0     0     0     0     0   -16     0     0     0     0     0   -16    64   -16     0     0     0
     0     0     0     0     0     0     0     0     0     0   -16     0     0     0     0     0   -16    64   -16     0     0
     0     0     0     0     0     0     0     0     0     0     0   -16     0     0     0     0     0   -16    64   -16     0
     0     0     0     0     0     0     0     0     0     0     0     0   -16     0     0     0     0     0   -16    64   -16
     0     0     0     0     0     0     0     0     0     0     0     0     0   -16     0     0     0     0     0   -16    64

>>
```

C INTLAB

C.1 いんとろ

C.2 入手とインストール

Paypal で支払うのだけど、事務を通すのが面倒だ。

zip ファイルを展開して、出て来たフォルダーを ~/ Documents/MATLAB の下に移動する。それから

最初に一度だけこれを実行

```
>> cd Intlab_V9
>> startintlab

(何回か Enter する。)
```

~/Documents/MATLAB/startup.m

```
addpath('~/Documents/MATLAB/Intlab_V9');
```

C.3 簡単な使い方

そういえば、過去に説明を書いたことがない。
Rump 氏によると、Rump [6] は

> 「INTLAB の原論文です．INTLAB を用いた結果を論文にするときには必ずこの文献を参照しましょう.」

とのこと。
Rump の学生 Hargreaves 氏の修士論文 [7] に、INTLAB のチュートリアルが含まれているとか (いかにもドイツですね)。
Moore-Kearfott-Cloud [8] も参考になるかも。
私の周囲の人が作ったプログラム例としては、福澤誠人君の修士論文 [9] がある。

D row major vs. column major

配列はメモリーの中で連続した領域を使って記憶されている。
1 次元配列 a の場合は、アドレスの低い (数値として小さい) 方から、a(1), a(2), ··· が並んでいる。これは大抵のプログラミング言語に共通である。
2 次元配列の場合にメモリーの中でどう並ぶかは、2 つの流儀がある。

D.1 MATLAB, Fortran は column major order

MATLAB の場合、例えば a=zeros(m,n); で用意した a について、アドレスの低い方から

$$\begin{gathered}\texttt{a(1,1)}, \texttt{a(2,1)}, \cdots, \texttt{a(m,1)},\\ \texttt{a(1,2)}, \texttt{a(2,2)}, \cdots, \texttt{a(m,2)},\\ \cdots,\\ \texttt{a(1,n)}, \texttt{a(2,n)}, \cdots, \texttt{a(m,n)}\end{gathered}$$

と、行 (row) 番号が先に動くように並んでいる。これを “column major order” という。
MATLAB 互換の Octave, Scilab 以外に、Fortran や R, S-Plus, Julia なども column major order である。

D.2 C は row major order

一方、C 言語で `double a[m][n];` で定義した `a` について、アドレスの低い方から

$$\begin{gathered}\texttt{a[0][0]}, \texttt{a[0][1]}, \cdots, \texttt{a[0][n-1]},\\ \texttt{a[1][0]}, \texttt{a[1][1]}, \cdots, \texttt{a[1]][n-1]},\\ \cdots,\\ \texttt{a[m-1][0]}, \texttt{a[m-1][1]}, \cdots, \texttt{a[m-1][n-1]}\end{gathered}$$

と、列 (column) 番号が先に動くように並んでいる。これを “row major order” という。

C 以外に、C++, Mathematica, Pascal なども row major order である。

FORTRAN は column major order, C は row major order

- 2 次元の差分法での説明で、筆者は $U_{i,j}$ を $u(1,1), u(2,1), \cdots$ と並べたが、これは column major order ということになる。
- MATLAB で

```
% a=0; b=3; c=0; d=1; nx=30; ny=10;
  x=linspace(a,b,nx+1);
  y=linspace(c,d,ny+1);
  [X,Y]=meshgrid(x,y);
  size(X)
  size(Y)
  X(:)
  Y(:)
```

とすると、`X` と `Y` のサイズはともに、$(\texttt{ny}+1, \texttt{nx}+1)$ である。`X(`j`,`i`)` と `Y(`j`,`i`)` $(1 \le i \le \texttt{nx}+1,\ 1 \le j \le \texttt{ny}+1)$ は、それぞれ格子点 $P_{i-1,j-1} = (x_{i-1}, y_{j-1})$ の x 座標 x_{i-1}、y 座標 y_{j-1} を記憶している、

`X` についていうと、メモリー中で

$$\begin{aligned}&\texttt{X(1,1)} = x_0, \texttt{X(2,1)} = x_0, \ldots, \texttt{X(ny+1,1)} = x_0,\\ &\texttt{X(1,2)} = x_1, \texttt{X(2,2)} = x_1, \ldots, \texttt{X(ny+1,2)} = x_1,\\ &\qquad \cdots,\\ &\texttt{X(1,nx+1)} = x_{\rm nx}, \texttt{X(2,nx+1)} = x_{\rm nx}, \ldots, \texttt{X(ny+1,nx+1)} = x_{\rm nx}\end{aligned}$$

と並んでいる。

`Y` についていうと、メモリー中で

$$\begin{aligned}&\texttt{Y(1,1)} = y_0, \texttt{Y(2,1)} = y_1, \ldots, \texttt{Y(ny+1,1)} = y_{\rm ny},\\ &\texttt{Y(1,2)} = y_0, \texttt{Y(2,2)} = y_1, \ldots, \texttt{Y(ny+1,2)} = y_{\rm ny},\\ &\qquad \cdots,\\ &\texttt{Y(1,nx+1)} = y_0, \texttt{Y(2,nx+1)} = y_1, \ldots, \texttt{Y(ny+1,nx+1)} = y_{\rm ny}\end{aligned}$$

と並んでいる。

`meshc(X,Y,U)` のようにして描画する場合、`U` は `X`, `Y` と同じような順番でデータを格納する必要がある。これはつまり

$$U_{0,0}, U_{0,1}, U_{0,2}, \cdots, U_{0,\mathrm{ny}}, U_{1,0}, U_{1,1}, U_{1,2}, \cdots, U_{1,\mathrm{ny}}, \cdots, U_{\mathrm{nx},0}, U_{\mathrm{nx},1}, U_{\mathrm{nx},2}, \cdots, U_{\mathrm{nx},\mathrm{ny}}$$

という順に並べるということである。

$i \in \{0, 1, \cdots, \mathrm{nx}\}$, $j \in \{0, 1, \cdots, \mathrm{ny}\}$ とするとき

$$\ell = i + j(\mathrm{ny} + 1)$$

で定まる ℓ は $\{0, 1, (\mathrm{nx}+1)(\mathrm{ny}+1) - 1\}$

E　グラフィックス

- `close(番号)` で特定のウィンドウを閉じる。`close all` で全てのウィンドウを閉じる。
- たくさんウィンドウを出してしまったら、`close all hidden` でお掃除。
- `fig=gcf; figure(fig)` あるいは、`figure(gcf)` でウィンドウが visible になる。
- `clf` でウィンドウから全てオブジェクトを削除する。`clf('reset')` とすると、プロパティもデフォールトに戻す。
- `title(文字列)` でウィンドウのタイトルを描く。

```
title(['heat equation, t=', num2str(tau*n, '%4.2f')])
```

それくらいなら `sprintf()` を使うのかな？

```
title(sprintf('heat equation, t=%4.2f', tau*n))
```

- `axis([x1 x2 y1 y2])`
- `drawnow; shg;` と言うおまじないを覚えよう。

F　固有値関係の命令

MATLAB のオンライン・マニュアルの日本語版には、長いこと直されていない誤訳がある。“magnitude” を無視しているけれど、これは「絶対値」という意味である。

参考文献

[1] 桂田祐史：1 次元の Poisson 方程式, https://m-katsurada.sakura.ne.jp/labo/text/poisson1d.pdf (2024/8/25).

[2] 桂田祐史：Poisson 方程式に対する差分法, https://m-katsurada.sakura.ne.jp/labo/text/poisson.pdf (2000 年?〜).

[3] 桂田祐史：2 次元 Poisson 非同次 Dirichlet 境界値問題を解く差分法プログラム (MATLAB), https://m-katsurada.sakura.ne.jp/labo/text/poisson2d-nonhomo.pdf (2024/8/24).

[4] 桂田祐史：2 次元 Poisson 非同次 Neumann 境界値問題を解く差分法プログラム (MATLAB), 準備中 (色々なことが分かってきてまとまらないので) (2024/8/25).

[5] Walter Ritz, von : Theorie der Transversalschwingungen einer quadratischen Platte mit freien Rändern, *Annalen der Physik Volume 333, Issue 4*, pp. 737–786, (1909), Ritz の方法が述べられている.

[6] Rump, S.: INTLAB - INTerval LABoratory, in Csendes, T. ed., *Developments in Reliable Computing*, pp. 77–104, Kluwer Academic Publishers, Dordrecht (1999), http://www.ti3.tuhh.de/rump/.

[7] Hargreaves, G. I.: Interval analysis in MATLAB, Master's thesis, Manchester Institute for Mathematical Sciences, School of Mathematics, The University of Manchester (2002), http://www.ti3.tuhh.de/rump/intlab/narep416.pdf.

[8] Moore, R. E., Kearfott, R. B. and Cloud, M. J.: *Introduction to INTERVAL ANALYSIS*, SIAM (2009), http://www-sbras.nsc.ru/interval/Library/InteBooks/IntroIntervAn.pdf から入手できる。

[9] 福澤誠人：定常 Stokes 方程式の有限要素解の事後誤差評価と事前誤差評価, https://m-katsurada.sakura.ne.jp/labo/report/open/2004-fukuzawa.pdf (2005).