MATLAB 使い方入門

桂田 祐史

2005 年 6 月 8 日,2009 年 12 月, 2018 年 9 月

色々な機会にあちこちに書き散らしたせいで、「あれはどこに行った」が多くなってきたので、とりあえず一ヶ所にまとめることにした。現時点では寄せ集め。

(2018 年 9 月) 久しぶりに MATLAB の利用を再開するため、新しく書き直すことにした (http://nalab.mind.meiji.ac.jp/~mk/labo/text/new-intro-matlab/, http://nalab.mind.meiji.ac.jp/~mk/labo/text/new-intro-matlab.pdf)。今後はこちらには手を入れず、必要に応じて、ここに書いてあることを抜き出して、新しい方に (なるべく整理した形で) 移す。

目 次

1 イントロ

(ここは 2005 年に書いたものです。現在では当てはまらないこと、その後勘違いに気がついたこともありますが、一から書き直すのも面倒なので、当面このままに残します。)

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 (現在 Moler は会長兼技師長であるとか)。

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であるが、筆者は最近

案外大したものではないか、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、以下紹介するように「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 は現在も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。簡単な偏微分方程式のシミュレーションへの応用も十分可能なレベルに成長した。

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「準標準」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[2]。残念ながら疎行列専用の処理が用意されていないが (これは大部前の話で、現在はサポートされている)、入門には十分であるし、用途を選べば実用性も高い。ただし、Octave のグラフ表示機能は gnuplot を用いて実現されているが、MATLAB, Scilab と比べるとやや劣っている。

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。疎行列処理も完全ではないが、LU 分解とそれに基づく連立 1

[1] http://www.dspguru.com/sw/opendsp/mathclo2.htm などが参考になる。
[2] 特に `octave --traditional` あるいは `octave --braindead` で MATLAB に近くなるとか (失礼な話だね)。

次方程式の解法程度はサポートしている。疎行列にする命令 sparse() がある。Asp が疎行列として、Aspx=b を解くには、

```
[h,rk]=lufact(Asp); x=lusolve(h,b); ludel(h);
```

とする。開発元から Windows 向けのバイナリーが配布されている。

2 コマンド等覚え書き

以下の記述のほとんどは最初 Octave で実行して試したので (MATLAB では試していないものがある)、MATLAB で動くかどうか保証しない。

- 行単位の編集機能、ヒストリー、タブによる補間
- 名前の大文字、小文字は区別する。
- clear 変数名 で変数を消去。clear variables または clear ですべての変数を消去。
- who とすると使用されている変数名を表示する。
  whos とすると使用されている変数の情報を表示する。
- 変数名 とすると、"変数名= 変数の値" と表示される。[変数名] とすると、"ans = 変数の値" と表示される。(ans=変数名 と同値ということ？)
- 関数呼び出し or 変数 = 関数呼び出し
  前者の場合 ans という変数に結果が入る。
- 行末に ; をつけると結果は表示されない。
- Octave, Scilab などの互換システムがある。
- 横ベクトル $(1,2,3)$ は [1,2,3] あるいは [1 2 3] で、縦ベクトル $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ は [1; 2; 3] で入力できる (あるいは [1,2,3]' のようにしても得られる)。
- 行列 $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ は

  ```
  [1,2,3; 4,5,6; 7,8,9]
  ```

  あるいは

```
[1,2,3
 4,5,6
 7,8,9]
```

で入力できる。

- `ones(`m`,`n`)` で m 行 n 列の、すべての成分が 1 である行列
- `zeros(`m`,`n`)` で m 行 n 列の、すべての成分が 0 である行列
- `eye(`n`,`n`)` で n 次の単位行列[3]
- `hilb(`n`)`, `invhilb(`n`)` はそれぞれ n 次の Hilbert 行列, n 次の Hilbert 行列の逆行列を表わす。
- `hadamard(`n`)` は n 次の Hadamard 行列。
- `rand(`m`,`n`)` で一様乱数行列、`randn(`m`,`n`)` で正規乱数行列を表わす。
- `sylvester_matrix(`n`)` でシルベスター行列を表わす。
- `toeplitz(v,n)`
- `2:5` で `[2 3 4 5]`, `0:0.2:1` で `[0.0 0.2 0.4 0.6 0.8 1.0]` が表せる。
- もっとも、区間 $[a, b]$ の N 等分点は、`linspace(a,b,N+1)` で求めるのがお勧め (結果は横ベクトルになる)。
- ベクトルの成分は、ベクトルの変数名 (インデックス) で表わせる (Fortran 風ですね)。
- `+`, `-`, `*`, `/` は普通の意味の四則を表わす (数、ベクトル、式に対して使える。)
- `A'` は行列またはベクトル `A` のエルミート共役を表す。単なる転置は `A.'` で表す。
- 縦ベクトル x, y の内積は

  ```
  x' * y
  ```

  で求まる。
- 例えば `(1:9)'*(1:9)` で九九の表が作れる
- `det(正方行列)` は行列式
- `trace()` はトレース
- `rank(a, tol)` はランク

[3]Octave では `eye(`n`)` で良いが、Scilab では `eye(`n`,`n`)` とする必要がある。

- kron(行列, 行列) は Kronecker 積
- 特性多項式は poly(行列) で計算できる (結果は多項式の係数の作るベクトル)。
- 多項式の積は conv(係数ベクトル，係数ベクトル) で計算できる。結果も係数の作るベクトルである。
- inv(行列) は逆行列 (でも滅多なことでは使わないこと！)
- 行列については B / A は $\mathtt{BA}^{-1}$, A \ B は $\mathtt{A}^{-1}\mathtt{B}$ を表す。特に連立 1 次方程式 $\mathtt{A}x = \mathtt{b}$ の解 $x = \mathtt{A}^{-1}\mathtt{b}$ は A \ b で計算できる。
- LU 分解するには lu() を用いる。行交換なしの LU 分解 ($A = LU$) と、それを用いた連立 1 次方程式 $Ax = b$ の解法は

```
[L,U]=lu(A);
x=U\(L\b));
```

  で OK. 行交換ありの LU 分解 ($PA = LU$ あるいは $A = PLU$) を用いる場合は

```
[L,U,P]=lu(A);
x=U\(L\(P*b));
```

  で OK. あるいは置換を置換行列をかけることで実現するのは不経済なので、次のような手段も用意されている。

```
[L,U,p]=lu(A,'vector');
x=U\(L\(b(p,:)));
```

- Cholesky 分解をするには chol()

```
u=chol(A)
l=u'
```

  とすると A=l u が成り立つ。
- eig(正方行列) とすると固有値 (を並べた縦ベクトル) が求まる。[p,d]=eig(正方行列) とすると固有ベクトル (を並べた) 行列 p と、固有値を並べた対角行列 d が得られる。

```
A=[1 2 3;4 5 6;8 5 2];
[P,D]=eig(A);
inv(P) * A * P
norm(inv(P) * A * P-D)
```

  (つまり $P^{-1}AP = D$ となることを確認している。)

- eig() は一般化固有値問題 ($Ax = \lambda Bx$) にも対応している。また大規模問題向けの eigs() も用意されている (一部の固有値、固有ベクトルを求める)。
- hess() (行列を Hessenberg 形に変換), schur() (行列の Schur 分解), svd() (行列の特異値分解) なども用意されている。
- 行列変数 (行始まり:行終り , 列始まり:列終り) で部分行列が得られる (行ベクトル、列ベクトル、ブロック等が取り出せる)。
- 行列の名前 (:,j) とすると第 j 列ベクトル、行列の名前 (i,:) とすると第 i 行ベクトルを表わす。
- 行列を 1 次元ベクトル化するには v=A(:) のようにする。
- 1 次元ベクトルを行列化するには

```
A=zeros(m,n);
A(:)=v;
```

  のように型の決まった行列に代入するか、reshape() という関数を用いて

```
A=reshape(v,m,n);
```

  のようにする。
- diag(ベクトル) とすると、ベクトルの成分を対角成分に持つ対角行列が求まる。例えば diag([1 2 3]) とすると、$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$ が得られる。上下にずらすことも出来て、例えば diag([1 2], 1) とすると、$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{pmatrix}$ が得られる (diag(ベクトル, 数) の数が正なら上に、負なら下にずらす)。

  一方、diag(行列) とすると、行列の対角線分を取り出したベクトルが得られる。ゆえに、diag(diag(行列)) とすると、行列の対角線分を取り出した対角行列が得られる。
- triu(行列), triu(行列, 数) は上三角部分を取り出す。
- tril(行列),tril(行列, 数) は下三角部分を取り出す。
- [A B] とか [A; B] は行列を並べて大きい行列を作る (それぞれ横に並べた $\begin{pmatrix} A & B \end{pmatrix}$, 縦に並べた $\begin{pmatrix} A \\ B \end{pmatrix}$ が得られる)。
- cond() は条件数を計算する。rcond() は LINPACK の条件数 (正確には条件数の粗い評価) の逆数を計算する。

- `norm(v,p)` は $\|\cdot\|_p$, `norm(v,inf)` は $\|\cdot\|_\infty$

$$\|\boldsymbol{x}\|_p = \left(\sum_{j=1}^{n} |x_j|^p\right)^{1/p}, \quad \|\boldsymbol{x}\|_\infty = \max_{1\le j\le n} |x_j|\,.$$

- `norm(x)` は `norm(x,2)` に等しい。
- `norm(x,'fro')` は Frobenius ノルム。
- `lookfor` は Octave にはない。
- `roots(`係数を表すベクトル`)` で多項式の根
  例えば `roots([1,0,0,-1])` で x^3-1 の根を並べたベクトルが得られる。
- `sum(`ベクトル`)` で成分の和、`prod(`ベクトル`)` で成分の積、`mean(`ベクトル`)` で成分の平均値、`median(`ベクトル`)` で中央値、`max(`ベクトル`)` で成分の最大値、`min(`ベクトル`)` で成分の最小値、`var(`ベクトル`)` で分散 (ただし要素数 ÷ $(n-1)$ で計算するやつ)、`cov(`ベクトル`)` で共分散行列、`std(`ベクトル`)` で標準偏差 (分散の正の平方根)、`length(`ベクトル`)` でベクトルとしての次元 (列としての長さ)、
- いわゆる特殊関数も揃っている。
- `plot(`ベクトル`)`, `closeplot`
- 制御構文には以下のようなものがある。慣れている人には説明不要であろう。

while

```
while 条件式
  文1
  文2
  ...
  文n
end
```

for

```
for i=1:n
 文1
 ...
 文n
end
```

if

```
if 条件式
  文1
  ..
  文n
else
  文'1
  ...
  文'n
end
```

if, elseif

```
if 条件式1
  文
elseif 条件式2
  文
end
```

- for 変数=初期値:終了値 または for 変数=初期値:増分:終了値
- break はループを抜け出す
- 「かつ」は &,「または」 は |,「等しくない」は~= または != で表わす。つまり否定は ~ または ! ということだろうな。
- キーボード入力

  変数名 = input('何か入力して下さい')

- pause(秒数) で時間待ちをする。pause でキーボードに触れるまで待つ。
- 演算子の前に . をつけると成分ごとという意味？c=[1 2 3] に対して c*c は型がおかしいから計算できない。c .* c とすると [1 4 9]
- 変数名=値
  ベクトルの場合は[変数名1, 変数名2]=値 や[変数名1 変数名2]=値 のようなことができる。
- 1 変数関数のグラフを描くには、ベクトルを 2 本用意する。

```
% y=x^2 のグラフ
x=0:0.1:10;
y=x.^2;
plot(x,y)
```

([0, 10] を 100 等分と考えて、x=linspace(0,10,100+1); とする方が良い？)

```
% y=sin(x) のグラフ
x=0:0.1:10;
y=sin(x);
plot(x,y)
```

- 古い Octave では gnuplot を使ってグラフ描画しているので、
gset term postscript; gset output "foo.ps"; replot
のような gnuplot 風のコマンドが使えたが、最近の Octave ではもう出来ない。Octave 拡張は使わない方が良いだろう。

- contour(z, 等高線のレベル数,x,y)

```
n=200; x=-1:2/n:1; y=x; z=x'*y;
contour(z)                  お手軽に等高線を描く
contour(z,20)               20 本の等高線を描く
contour(x,y,z,20)           x,y をちゃんと指定して、20 本の等高線
```

あるいは

```
[x,y]=meshgrid(-1:0.01:1, -1:0.01:1);
z=x.*y;
contour(x,y,z);
```

等高線のレベルを指定するには、1 次元配列を与えれば良い。レベル h の等高線を 1 本だけ描きたいときは [h h] とする ([h] とすると、h 本の等高線を描いてしまう)。

```
contour(x,y,z,-1:0.1:1)     等高線のレベルを指定
contour(x,y,z,[0.5 0.5])    レベル 0 の等高線
```

- mesh(x,y,z) は 3 次元グラフ (いわゆる鳥瞰図)

- i, I, j, J は虚数単位、pi は円周率、e は自然対数の底 (Napier の数)、inf は無限大、eps は計算機イプシロン

- $3 + 5i$ は 3+5i で表せる。5 と i の間に空白を置いてはいけない。

- real(), imag(), conj(), arg(), angle() などが使える。

- eval(式) は式の評価

```
t='1/(i+j-1)'
for i=1:n
  for j=1:n
    a(i,j)=eval(t);
  end
end
```

- 画面出力の精度は
  `format long`
  `format short`
  `format long e`　長い桁数、指数形式
  `format short e`　短い桁数、指数形式
- 数値積分は `quad('関数名', 初期値, 終端値)` などが用意されている。
- `save -ascii ファイル名 変数名1 変数名2`, `save -ascii -double ファイル名 変数名1 変数名2`, `save -binary ファイル名 変数名1 変数名2`, `save -mat-binary ファイル名 変数名1 変数名2`

(準備中: `disp()`, `fprintf()`)

3　最初の例: 連立1次方程式、逆行列、固有値&固有ベクトル

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}, \quad x = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

に対して、$b = Ax$ を計算してから、$A^{-1}b$ を計算し (もちろん x と等しくなるはず)、A^{-1} を計算して、$A^{-1}A = I$ を確かめ、最後に A の固有値 λ_1, λ_2 と、$P^{-1}AP = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$ となる P を求める。

MATLAB のコマンド・メモ (1)

$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	`a=[1,2;3,4]`　または　`a=[1,2 3,4]`
$A^{-1}b$ を計算する	`a\b`
A^{-1} を計算する	`inv(a)`
A の固有値を計算する	`eig(a)`
A の固有値と固有ベクトルを計算する	`[p lambda]=eig(a)` p は固有ベクトルを並べた行列

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

a =

     1     2
     2     3

>> x=[1;-1]

x =

     1
    -1

>> b=a*x

b =

    -1
    -1

>> a\b

ans =

    1.0000
   -1.0000

>> inv(a)

ans =

   -3.0000    2.0000
    2.0000   -1.0000

>> inv(a)*a

ans =

     1     0
     0     1

>> eig(a)

ans =

   -0.2361
    4.2361

>> [p lambda]=eig(a)

p =

   -0.8507    0.5257
    0.5257    0.8507
```

```
lambda =

   -0.2361         0
         0    4.2361

>> inv(p)*a*p

ans =

   -0.2361   -0.0000
   -0.0000    4.2361

>>
```

最後の $P^{-1}AP$ の計算は、丸め誤差の影響で、完全な対角行列にはなっていない (が、誤差は 10^{-16} のオーダーで十分小さい)。

4 行列が疎でも逆行列が疎とは限らない

微分方程式の数値シミュレーションに現われる連立 1 次方程式の係数行列は、ほとんどの場合、成分に 0 が多い。そのような行列を**疎** (sparse) であるという。疎行列はその性質をうまく利用することで、効率的な計算が可能になり、大規模な問題が解けるようになる。行列が疎であっても、その逆行列は疎であるとは限らない (大抵の場合、疎ではない) ことに注意する。

MATLAB のコマンド・メモ (2)

`1:n`	[1 2 3 ··· n]
`a'`	a の Hermite 共役 (実行列、実ベクトルの場合転置)
`a.'`	a の転置
`eye(m,n)`	(m 行 n 列の) 単位行列
`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)
命令 1 ; 命令 2	一行に複数の命令を書ける。";" があると表示が抑制される。

以下の例では、

$$A = \begin{pmatrix} 2 & -1 & & & 0 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ 0 & & & -1 & 2 \end{pmatrix}$$

という**三重対角行列** (**tridiagonal matrix**) について、連立 1 次方程式を解いたり、逆行列を計算したりしている。

```
>> n=4

n =

     4

>> 1:n

ans =

     1     2     3     4

>> (1:n)'*(1:n)

ans =

     1     2     3     4
     2     4     6     8
     3     6     9    12
     4     8    12    16

>> eye(n,n)

ans =

     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

>> zeros(n,n)

ans =

     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0

>> ones(n-1,1)

ans =

     1
     1
     1

>> diag(ones(n-1,1),1)

ans =

     0     1     0     0
     0     0     1     0
     0     0     0     1
     0     0     0     0
```

```
>> diag(ones(n-1,1),-1)

ans =

     0     0     0     0
     1     0     0     0
     0     1     0     0
     0     0     1     0

>> a=2*eye(n,n)-diag(ones(n-1,1),1)-diag(ones(n-1,1),-1)

a =

     2    -1     0     0
    -1     2    -1     0
     0    -1     2    -1
     0     0    -1     2

>> x=(1:n)'

x =

     1
     2
     3
     4

>> b=a*x

b =

     0
     0
     0
     5

>> ia=inv(a)

ia =

    0.8000    0.6000    0.4000    0.2000
    0.6000    1.2000    0.8000    0.4000
    0.4000    0.8000    1.2000    0.6000
    0.2000    0.4000    0.6000    0.8000

>> ia*a

ans =

    1.0000   -0.0000   -0.0000    0.0000
         0    1.0000   -0.0000    0.0000
         0         0    1.0000    0.0000
         0         0         0    1.0000

>> n=8;a=2*eye(n,n)-diag(ones(n-1,1),1)-diag(ones(n-1,1),-1)
```

```
a =

     2    -1     0     0     0     0     0     0
    -1     2    -1     0     0     0     0     0
     0    -1     2    -1     0     0     0     0
     0     0    -1     2    -1     0     0     0
     0     0     0    -1     2    -1     0     0
     0     0     0     0    -1     2    -1     0
     0     0     0     0     0    -1     2    -1
     0     0     0     0     0     0    -1     2

>> inv(a)

ans =

    0.8889    0.7778    0.6667    0.5556    0.4444    0.3333    0.2222    0.1111
    0.7778    1.5556    1.3333    1.1111    0.8889    0.6667    0.4444    0.2222
    0.6667    1.3333    2.0000    1.6667    1.3333    1.0000    0.6667    0.3333
    0.5556    1.1111    1.6667    2.2222    1.7778    1.3333    0.8889    0.4444
    0.4444    0.8889    1.3333    1.7778    2.2222    1.6667    1.1111    0.5556
    0.3333    0.6667    1.0000    1.3333    1.6667    2.0000    1.3333    0.6667
    0.2222    0.4444    0.6667    0.8889    1.1111    1.3333    1.5556    0.7778
    0.1111    0.2222    0.3333    0.4444    0.5556    0.6667    0.7778    0.8889

>>
```

A が疎であっても、A^{-1} の成分には 1 つも 0 がないことを確認しよう。

任意の正則行列 A に対して、

(1) $$PA = LU,$$

(2) $$P = \text{置換行列}, \quad L = \text{下三角行列}, \quad U = \text{上三角行列}$$

を満たす P, L, U が存在する (LU 分解 — 付録 D.5 を見よ)。

$Ax = b$ であれば、$Pb = PAx = LUx$ であるから、

$$x = U^{-1}(L^{-1}(Pb))$$

となり[4]、L^{-1}, U^{-1} の掛け算は非常に簡単なので (付録 D.5.2 参照)、P, L, U が求まっていれば連立 1 次方程式は簡単に解ける。(1), (2) を満たす P, L, U を求めることを A を LU 分解するという。

MATLAB のコマンド・メモ (3)

`[L u]=lu(a)`	`a` を LU 分解する。`a` = `Lu` を満たす `L`, `u` を求める。
`u\(L\b)`	`ax` = `b` を解く ($U^{-1}\,(L^{-1}b)$ を計算して)。
`[L u p]=lu(a)`	`a` を (行交換ありで) LU 分解する。`pa` = `Lu` を満たす `p`, `L`, `u` を求める。
`u\(L\(p*b))`	`ax` = `b` を解く ($U^{-1}\,(L^{-1}(Pb))$ を計算して)。

[4] ここで、$((U^{-1}L^{-1})P)b$ という順番にかけないのがミソである。$A^{-1} = (U^{-1}L^{-1})P$ を計算するのは計算の手間がかかる。特に A が疎行列の場合は、桁違いの差になるのが普通である。

```
>> [L u]=lu(a)

L =

    1.0000         0         0         0         0         0         0         0
   -0.5000    1.0000         0         0         0         0         0         0
         0   -0.6667    1.0000         0         0         0         0         0
         0         0   -0.7500    1.0000         0         0         0         0
         0         0         0   -0.8000    1.0000         0         0         0
         0         0         0         0   -0.8333    1.0000         0         0
         0         0         0         0         0   -0.8571    1.0000         0
         0         0         0         0         0         0   -0.8750    1.0000


u =

    2.0000   -1.0000         0         0         0         0         0         0
         0    1.5000   -1.0000         0         0         0         0         0
         0         0    1.3333   -1.0000         0         0         0         0
         0         0         0    1.2500   -1.0000         0         0         0
         0         0         0         0    1.2000   -1.0000         0         0
         0         0         0         0         0    1.1667   -1.0000         0
         0         0         0         0         0         0    1.1429   -1.0000
         0         0         0         0         0         0         0    1.1250

>> x=(1:n)'

x =

     1
     2
     3
     4
     5
     6
     7
     8

>> b=a*x

b =

     0
     0
     0
     0
     0
     0
     0
     9

>> u\(L\b)

ans =

    1.0000
```

```
    2.0000
    3.0000
    4.0000
    5.0000
    6.0000
    7.0000
    8.0000

>>
```

A を LU 分解したときの因子 P, L, U が確かに置換行列、下三角行列、上三角行列であることを確かめよう (このケースでは、P は単位行列である)。L, U が疎行列であることも確かめよう。

今回は使わなかったが、よく使われるコマンドを紹介しておく。

MATLAB のコマンド・メモ (4)

`x(i)`	`x` の第 `i` 成分
`a(i,j)`	`a` の第 (`i`, `j`) 成分
`det(a)`	`a` の行列式
`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
`tic; コマンド; toc`	コマンドの実行時間を計測
`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

余裕があれば試してみよう

1. 行列 `a` が大きいとき、`inv(a)`, `det(a)`, `a\` の実行時間を測って比較する。
2. `a\` の実行時間は、未知数の個数 n につれどう変化するか。

```
n=10
for i=1:8  % この 8 をむやみに大きくしないこと
  a=rand(n,n); b=rand(n,1);
  tic; a\b; toc
  n=n*2
end
```

課題

(1) 行列の積の計算や連立 1 次方程式を解く時間を測って、それが行列の寸法 n にどのように依存しているかを調べよ。なお、コマンドの実行時間は `tic; コマンド;toc` で計測できる。例えば

```
octave:1> n=100;a=rand(n,n);b=rand(n,1);tic;x=a\b;toc
```

とすると、連立 1 次方程式の解の計算 `a\b` の実行時間が表示される。n を色々変えて計測し、横軸 n, 縦軸が計算時間であるグラフを描いて[5]分析せよ (対数グラフが適当かも知れない)。

(2) 三角行列の逆行列が三角行列であることを実験で確かめよ[6]。また、疎行列であるという性質は逆行列には遺伝しないことを上の実験で確かめたが、疎である三角行列であるという性質はどうか？これも実験で確かめよ。

5 ある行列の固有値問題から

とある授業の CG 法の計算で、例[7] として取り上げた行列

$$A_c = \begin{pmatrix} A' & O \\ O & cA' \end{pmatrix}, \quad A = \begin{pmatrix} 4 & -1 & & & 0 \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ 0 & & & -1 & 4 \end{pmatrix} \in M(50;\mathbf{R}), \quad c = 1, 10, 100$$

の固有値を調べてみよう。

(連立 1 次方程式を解くための方法である CG 法では、係数行列の固有値の分布が収束の速さに影響する。最大固有値と最小固有値の比が 1 に近ければ収束は速いが、それから外れるに従って収束が遅くなる。それを確かめるために作った例題であり、c を大きくすると、最大固有値と最小固有値の比が大きくなり、収束が遅くなることを見てもらうのが趣旨であるが、MATLAB を使うと、実際に固有値の分布がどう変化するか、一目瞭然である、ということ。)

MATLAB の実行例 (1) すべて対話的に実行

```
>> n=5

n =

     5
```

[5] Octave 自身を使ってもよいし、**gnuplot** を使っても良い。

[6] 与えられた行列の下三角部分を求める `tril()`, 上三角部分を求める `triu()` という関数を利用すると便利かもしれない。

[7] `http://nalab.mind.meiji.ac.jp/~mk/lecture/suurikaisekitokuron/lesson2/node3.html`

```
>> J=diag(ones(n-1,1),1)+diag(ones(n-1,1),-1)

J =

     0     1     0     0     0
     1     0     1     0     0
     0     1     0     1     0
     0     0     1     0     1
     0     0     0     1     0

>> Ap=4*eye(n,n)-J

Ap =

     4    -1     0     0     0
    -1     4    -1     0     0
     0    -1     4    -1     0
     0     0    -1     4    -1
     0     0     0    -1     4

>> n2=2*n

n2 =

    10

>> Ac=zeros(n2,n2)

Ac =

     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0

>> Ac(1:n,1:n)=Ap

Ac =

     4    -1     0     0     0     0     0     0     0     0
    -1     4    -1     0     0     0     0     0     0     0
     0    -1     4    -1     0     0     0     0     0     0
     0     0    -1     4    -1     0     0     0     0     0
     0     0     0    -1     4     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
```

```
>> c=1

c =

     1

>>  Ac(n+1:n2,n+1:n2)=c*Ap

Ac =

     4    -1     0     0     0     0     0     0     0     0
    -1     4    -1     0     0     0     0     0     0     0
     0    -1     4    -1     0     0     0     0     0     0
     0     0    -1     4    -1     0     0     0     0     0
     0     0     0    -1     4     0     0     0     0     0
     0     0     0     0     0     4    -1     0     0     0
     0     0     0     0     0    -1     4    -1     0     0
     0     0     0     0     0     0    -1     4    -1     0
     0     0     0     0     0     0     0    -1     4    -1
     0     0     0     0     0     0     0     0    -1     4

>> lambda=eig(Ac)

lambda =

    2.2679
    2.2679
    3.0000
    3.0000
    4.0000
    4.0000
    5.0000
    5.0000
    5.7321
    5.7321

>> plot(1:n2,lambda)
>>
```

MATLAB の実行例 (2) 自分で定義した関数を呼び出す

前項では、$n = 5$, $c = 1$ で実験したが、色々な数値に対して実験しよう。そのため、関数 report_eigen(n,c) を定義する。

例えば

コマンド ウィンドウから編集開始

```
>> edit report_eigen
```

として現れる Editor ウィンドウに、以下の report_eigen.m の内容を入力し、保存する。

あるいは別途 emacs などのテキスト・エディターを使って、以下のような関数定義ファイル report_eigen.m を作り、適切な場所 (例えば、macOS ならば、~/Documents/MATLAB な

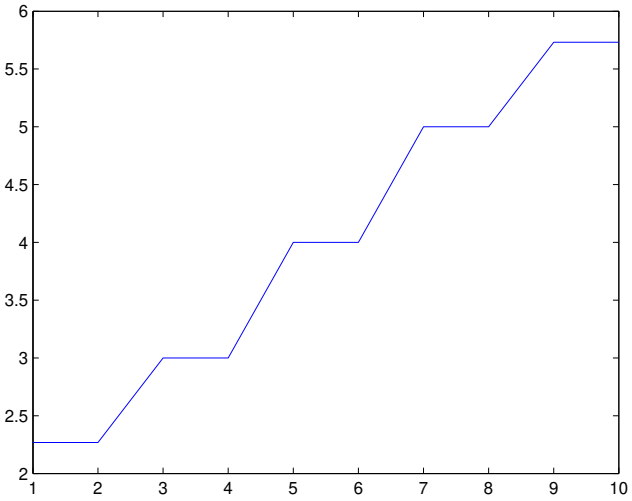


図 1: $c = 1$ の場合の A_c の固有値の分布

ど) におく。

report_eigen.m

```
function lambda = report_eigen(n,c)
  J=diag(ones(n-1,1),1)+diag(ones(n-1,1),-1);
  A=4*eye(n,n)-J;
  n2=2*n;
  Ap=zeros(n2,n2);
  Ap(1:n,1:n)=A;
  Ap(n+1:n2,n+1:n2)=c*A;
  lambda=eig(Ap);
```

```
n=50; c=1; lambda=report_eigen(n,c); plot(1:2*n,lambda)
>> max(lambda)/min(lambda)

ans =
    2.9924

>> n=50; c=10; lambda=report_eigen(n,c); plot(1:2*n,lambda)
>> max(lambda)/min(lambda)

ans =
   29.9243

>> n=50; c=100; lambda=report_eigen(n,c); plot(1:2*n,lambda)
>> max(lambda)/min(lambda)

ans =
  299.2428

>>
```

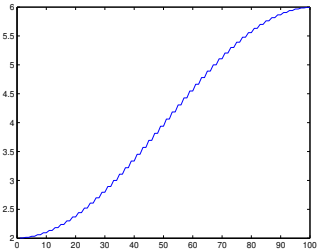


図 2: $n = 50$, $c = 1$

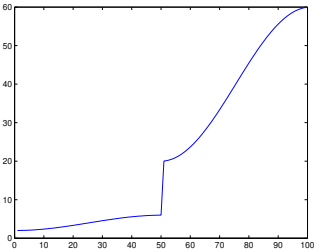


図 3: $n = 50$, $c = 10$

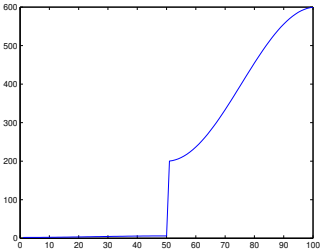


図 4: $n = 50$, $c = 100$

6　正方形領域での Poisson 方程式の Dirichlet 境界値問題

正方形領域 $\Omega = (0,1) \times (0,1)$ における Poisson 方程式の Dirichlet 境界値問題

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

を差分法で解くには、.... 差分法でどういう連立 1 次方程式が導かれるかについては、『Poisson 方程式に対する差分解』[8] を見よ。

$f \equiv 1$ の場合に解くには、次のようなプログラムを用いれば良い。

report_poisson.m

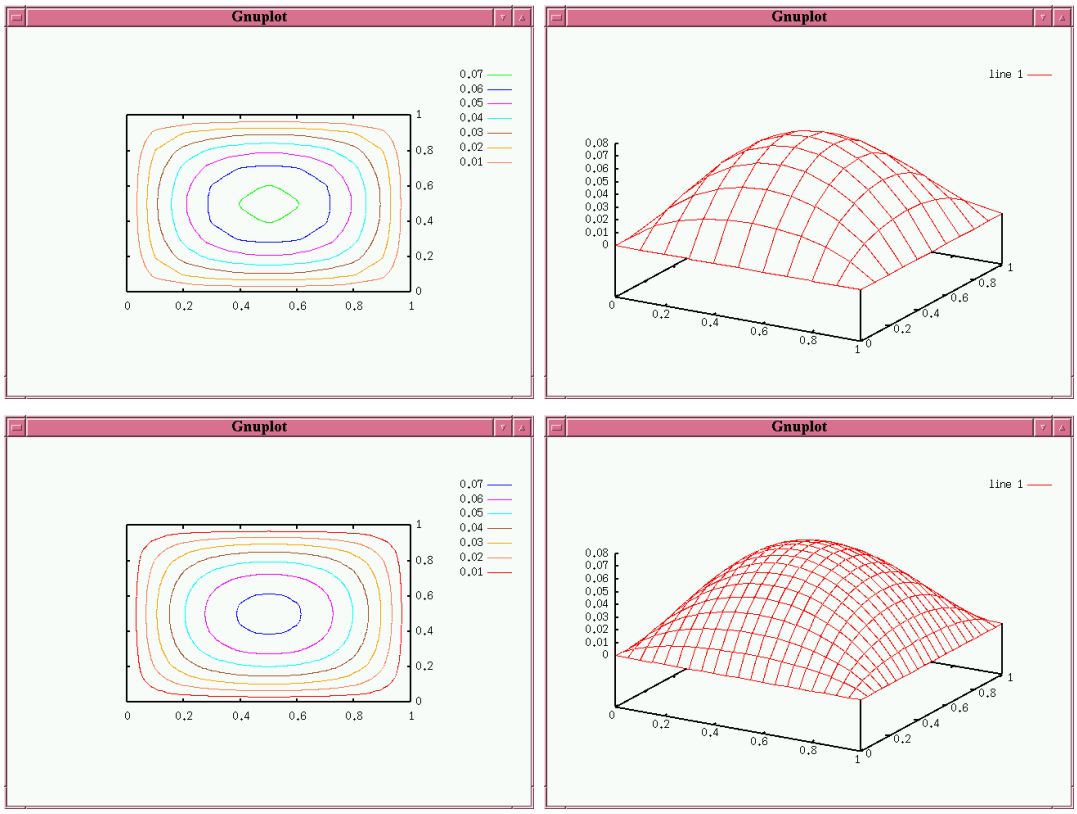
```
% report_poisson.m
%   正方形領域で -△u=f, 同次Dirichlet境界値問題を解く
%   正方形の各辺を n 等分
function u = report_poisson(n)
  h=1/n;
  J=diag(ones(n-2,1),1)+diag(ones(n-2,1),-1);
  Id=eye(n-1,n-1);
  A=-4*kron(Id,Id)+kron(J,Id)+kron(Id,J);
  % b_{ij}=-h^2 f(x_i,y_j)
  b=-h*h*ones((n-1)*(n-1),1);
  U=A\b;
  u=zeros(n+1,n+1);
  for j=1:n-1
    u(2:n,j+1)=U((j-1)*(n-1)+1:j*(n-1));
  end
%endfunction
```

このプログラムは、効率について無頓着な書き方をされているので、$n = 50$ 程度しか解けない ($n = 100$ とすると、未知数の個数は約 100 万で、小さいコンピューターで解くのは困難である)。

[8] http://nalab.mind.meiji.ac.jp/~mk/labo/text/poisson.pdf

```
oyabun% octave
GNU Octave, version 2.0.16 (i386-unknown-freebsd3.4).
Copyright (C) 1996, 1997, 1998, 1999, 2000 John W. Eaton.
This is free software with ABSOLUTELY NO WARRANTY.
For details, type ‘warranty’.

octave:1> n=10; u=report_poisson(n);
octave:2> x=0:1/n:1; y=x; contour(u,10,x,y);
octave:3> mesh(x,y,u);
octave:4> n=20; tic; u=report_poisson(n); toc
ans = 1.1213
octave:5> x=0:1/n:1; y=x; contour(u,10,x,y);
octave:6> mesh(x,y,u);
octave:7> n=40; tic; u=report_poisson(n); toc
ans = 16.870
octave:8> x=0:1/n:1; y=x; contour(u,10,x,y);
octave:9> quit
oyabun%
```



係数行列の表現を、少し一般化して

$$I_n \otimes \left[\frac{1}{h_x^2}(2I_m - J_m)\right] + \left[\frac{1}{h_y^2}(2I_n - J_n)\right] \otimes I_m, \quad m := N_x - 1, \quad n := N_y - 2$$

と変更した (やはり http://nalab.mind.meiji.ac.jp/~mk/labo/text/poisson.pdf を見よ)。それを元に書いたのが次の solve_poisson.m である。

solve_poisson.m

```
% solve_poisson.m --- 正方形領域における Poisson 方程式 (2010/2/11)
function solve_poisson(nx,ny)
     hx=1/nx;
     hy=1/ny;
     m=nx-1;
     n=ny-1;
     Im=eye(m,m);
     In=eye(n,n);
     Jm=diag(ones(m-1,1),1)+diag(ones(m-1,1),-1);
     Jn=diag(ones(n-1,1),1)+diag(ones(n-1,1),-1);
     Lx=(2*Im-Jm)/(hx*hx);
     Ly=(2*In-Jn)/(hy*hy);
     A=kron(Ly,Im)+kron(In,Lx);
     f=ones(m*n,1);
     U=zeros(m,n);
     U(:)=A\f;
%
     u=zeros(nx+1,ny+1);
     u(2:nx,2:ny)=U;
%
     x=0:1/nx:1;
     y=0:1/ny:1;
%
     colormap hsv
     subplot(1,2,1);
     mesh(x,y,u);
     colorbar
%
     subplot(1,2,2);
     contour(x,y,u);
%
     disp('saving graphs');
     print -depsc2 solve_poisson.eps
```

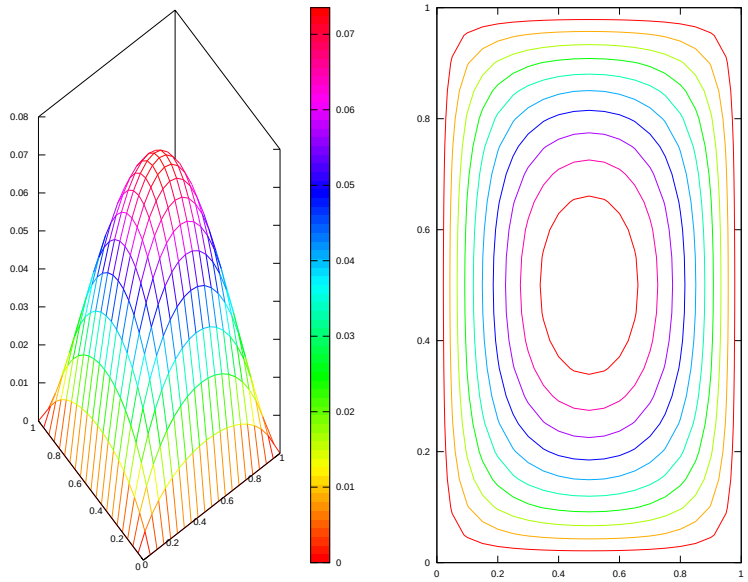


図 5: 正方形領域で $-\triangle u = 1$ を解く

2009 年久しぶりの見直し: 疎行列対応万歳

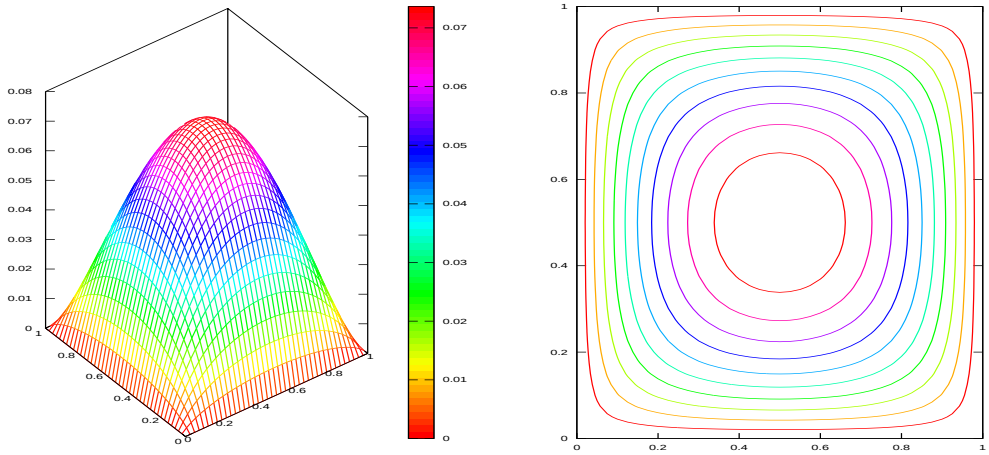
何年か放置していた間に、Octave にも `sparse()` などの疎行列関係の関数が用意されるようになったので、少し勉強してプログラムをちょっと改良した。

sparse_poisson.m

```
% sparse_poisson.m --- 正方形領域における Poisson 方程式 (2009/12/29)
function [x,y,u]=sparse_poisson(n)
  h=1/n;
  J=sparse(diag(ones(n-2,1),1)+diag(ones(n-2,1),-1));
  I=speye(n-1,n-1);
  A=-4*kron(I,I)+kron(J,I)+kron(I,J);
  b=-h*h*ones((n-1)*(n-1),1);
% 2次元化を少し工夫
  U=zeros(n-1,n-1);
  U(:)=A\b;
  u=zeros(n+1,n+1);
  u(2:n,2:n)=U;

  x=0:1/n:1;
  y=x;
% まず鳥瞰図
  subplot(1,2,1);
  colormap hsv
  mesh(x,y,u);
  colorbar
% 等高線
  subplot(1,2,2);
  contour(x,y,u);
% PostScript を出力
  disp('saving graphs');
  print -depsc2 sparsepoisson.eps
```

計算内容は同じであるが、$n = 100$ くらいスイスイ解けるようになった。すばらしい。



- ここで subplot() を使って、グラフの鳥瞰図と等高線を並べて描いてある。subplot(m,n,k) はウィンドウを m 行 n 列の領域に分けて、その k 番目の領域に描画することを意味する。
- print -depsc2 ファイル名 あるいは print('-depsc2', ファイル名を表す文字列) で、EPS 形式で出力する (出力サイズが大きい ?!)。
- print -dpdf ファイル名 あるいは print('-dpdf', ファイル名を表す文字列) で、PDF 形式で出力する。
- colorbar でカラーバーを出している。
- subplot(1,2,2); contour(x,y,u); のところは次のようにすべきだった (2012 年加筆)。

```
right=subplot(1,2,2);
contour(x,y,u);
pbaspect(right,[1 1 1]);
```

次に示すのは、solve_poisson.m の疎行列対応版である。

sparse_poisson3.m (2010/2/14)

```
% sparse_poisson3.m (2010/2/14) かなり効率が高い。描画の方に時間がかかる。
function sparse_poisson3(nx,ny)
% 係数行列の作成 (nx=ny=100 で 0.031 秒)
     hx=1/nx;
     hy=1/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(Ly,speye(m,m))+kron(speye(n,n),Lx);
% 連立方程式を作成して解く (結果は 2 次元配列に格納, nx=ny=100 で 0.3 秒)
     f=ones(m*n,1);
     U=zeros(m,n);
     U(:)=A\f;
% 境界値 0 をつける
     u=zeros(nx+1,ny+1);
     u(2:nx,2:ny)=U;
%
     x=0:1/nx:1;
     y=0:1/ny:1;
%
     clf
     colormap hsv
     subplot(1,2,1);
     mesh(y,x,u);
     colorbar
%
     subplot(1,2,2);
     contour(y,x,u);
%
     disp('saving graphs');
     print -depsc2 sparse_poisson3.eps
```

7 正方形領域における Laplacian の固有値問題

(この節の 7.1, 7.2 に書いてあることは古い。今では疎行列用の命令がちゃんと動くはず。直すのは必要が生じるまでさぼる。)

正方形領域における Laplacian の固有値問題を差分法で解け。

- 差分方程式の導出は自分で行うこと (プログラムと照らし合わせる)。
- Octave あるいは MATLAB を用いると良い。
- 得られた近似固有値の計算精度を調べよ。
- 重複固有値について調べよ。
- 近似固有関数の可視化をせよ。
- 特に絶対値の小さい近似固有関数について、厳密な固有関数と比較せよ。重複固有値に対応するものはどうなっているか？
- MATLAB で疎行列用の命令を使うとどうなるか？

7.1 ある晩 Octave, Scilab で遊んでみました

まずは Octave で解いてみよう。

eigen_square.m

```
## 正方形領域のラプラシアンの固有値
function retval = eigen_square(n)
  h = 1/n;
  B=diag(ones(n-2,1),1)+diag(ones(n-2,1),-1);
  I=eye(n-1,n-1);
  A = - n * n * (- 4 * kron(I,I) + kron(B,I) + kron(I,B));
  retval = eig(A);
endfunction
```

eigen_square2.m

```
## 正方形領域のラプラシアンの固有値、固有関数
function [v,lambda] = eigen_square2(n)
  h = 1/n;
  B=diag(ones(n-2,1),1)+diag(ones(n-2,1),-1);
  I=eye(n-1,n-1);
  A = - n * n * (- 4 * kron(I,I) + kron(B,I) + kron(I,B));
  [v,lambda] = eig(A);
endfunction
```

```
octave:1> eigen_square(10)
```

Scilab では、retval=eig(A) の代わりに retval=spec(A)、[v,lambda]=eig(A) の代わりに [v,lambda]=bdiag(A) くらいで行くか？sparse() という命令があるそうだ…期待に胸が膨らむ。

eigen_square3.m

```
# 正方形領域のラプラシアンの固有値 (Scilab version) バグあり
function retval = eigen_square3(n)
  h = 1/n;
  B=sparse(diag(ones(n-2,1),1)+diag(ones(n-2,1),-1));
  I=speye(n-1,n-1);
  A = - n * n * (- 4 * kron(I,I) + kron(B,I) + kron(I,B));
  retval = spec(A);
endfunction
```

おや、kron(I,I) でエラーになる。

```
-->A = - n * n * (- 4 * kron(I,I) + kron(B,I) + kron(I,B));
    p                              !--error   246
impossible to overload this function for given argument type(s)
       undefined function %sp_kronm
```

つまり、残念ながら疎行列用の kron() はないわけだ。また疎行列用の spec() もないらしい。結局、Octave と本質的には変わらずに

改訂版 eigen_square3.m

```
# 正方形領域のラプラシアンの固有値 (Scilab version)
function retval = eigen_square3(n)
  B=diag(ones(n-2,1),1)+diag(ones(n-2,1),-1);
  I=eye(n-1,n-1);
  A = n * n * (4 * kron(I,I) - kron(B,I) - kron(I,B));
  retval = spec(A);
endfunction
```

とするしかない。timer();a=eigen_square3(10);timer() として計算時間を計測した。

n	計算時間 (秒)
10	0.05
20	2.17
25	8.45
30	28.69
40	191.82

確かに計算時間は n^6 に比例している。ちなみに事前に何もしないと $n = 30$ で異常終了する。スタックがあふれたらしい。stacksize() で調べると、double を一単位として 1M となっていた。つまり 1000 次程度の正方行列が覚えられるリミットである。そこで stacksize(50*1000*1000) としたところ、大きな n に対しても計算できるようになった。これなら 50M 要素 = 400MB だから、主記憶 512MB の oyabun ではそれほど破滅的なことにはならない。

7.2 またある晩のこと

あれから何年経ったのか？必要があって、久しぶりに Octave に触る。

eigen_square4.m

```
% eigen_square4.m
% 2009/12/30 Octave で動かす。
function retval = eigen_square3(n)
  h = 1/n;
  B=sparse(diag(ones(n-2,1),1)+diag(ones(n-2,1),-1));
  I=speye(n-1,n-1);
  A = - n * n * (- 4 * kron(I,I) + kron(B,I) + kron(I,B));
  retval = eig(A);
```

Intel Pentium M 1.2GHz, 1GB RAM というかなり古いマシン (2005 年度購入) したところ、次のような結果になった。

10	0.09375s
20	0.28125s
40	9.75000s
60	112.438s
80	667.938s

ちなみに同じコンピューターで eigen_square.m を動かすと、次のようになる。

10	0.01563s
20	0.2344s
40	9.891s
60	112.8s
80	818.8s

あれれ？あまり変わらない。連立 1 次方程式と違って、固有値の方は sparse でやる強みがない？

7.3 そして…重 Laplacian の固有値問題

(工事中, 2013/1/7)

2012 年春，正方形領域における重調和作用素の固有値問題 (いわゆる Chladni 図形) の数値計算を扱った大学院生の修士論文で実を結んだ。

- 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 —」, 2012 年度明治大学大学院理工学研究科基礎理工学専攻修士論文, 2012 年 2 月[9]
  (MATLAB プログラムとそれを用いて計算したクラドニ図形の図がたくさん含まれています。)

[9] http://nalab.mind.meiji.ac.jp/~mk/labo/report/pdf/2011-hirano.pdf

- 2012 年日本数学会秋季総合分科会での学会発表の資料は Numerical Analysis of Chladni figures[10] にあります。

こんな風に実行する

```
>> N=160
>> A=plate_f1(N,0.3);
>> [v,d]=eigs(A,200,0);
>> plot_n(v(:,201-4),N,N)
```

自由縁を持つ正方形の板 (Poisson 比は 0.3 とする) の固有振動の固有値，固有関数を調べる。正方形の各辺を $N = 160$ 等分して差分法で行列の固有値問題を導く。その行列は `plate_f1()` で計算出来る。MATLAB 標準の `eigs()` を用いて，固有値＆固有関数を (固有値が小さい方から) 200 個計算する (余談であるが、`eigs()` はデフォールトでは、固有値の絶対値が大きい方から指定した個数の固有値、固有ベクトルを求める。日本語ドキュメントでは、絶対値 (`magnitude`) というのが落ちていて、これははっきり誤訳だと思う。3 番目の引数にスカラーを指定すると、それを使ってシフト法をするらしい。)。0 でない最初の固有値である第 4 固有値 λ_4 (0 は三重に縮重している) に属する固有関数の節線を `plot_n()` で描く。

16GB のメモりーを搭載した Mac Pro で，$N = 1280$ で計算することが出来る (1 時間ちょっとかかる)。top でチェックすると，CPU が時々 1600% 近くになっているのは大したものだと思う。

7.3.1 plate_f1.m

```
% plate_f1.m -- Eigenvalue problem of square plates with free edges
%  written by Hirano Yuuki, Meiji University, Feb 2012
%  comments are modified by Masashi Katsurada, 24 June 2012.
%
%  \triangle^2 u=\lambda u
%  0<x<1, 0<y<1
%  mu: Poisson's ratio
%  usage:
%    A=plate_f1(640,0.3);
%    [v,d]=eigs(A,200,0);
%    plot_n(v(:,197),640,640)

function P1=plate_f1(N,mu)
        h=1/N;
        n=N+1;
        a=-2*(mu*mu+2*mu-3);
        b=1-mu*mu;
        c=-2*(mu-1);
        d=15-8*mu-5*mu*mu;
        e=-4*(mu*mu+mu-2);
        f=2-mu;
        g=-2*(mu-3);
        k=-2*(3*mu*mu+4*mu-8);
        In=speye(n,n);
```

[10] http://nalab.mind.meiji.ac.jp/~mk/chladni/

```
Jn=sparse(diag(ones(n-1,1),1)+diag(ones(n-1,1),-1));
J2n=sparse(diag(ones(n-2,1),2)+diag(ones(n-2,1),-2));
j2n=sparse(diag(ones(n-2,1),2)+diag(ones(n-2,1),-2));
j2n(1,3)=sqrt(2);
j2n(3,1)=sqrt(2);
j2n(n-2,n)=sqrt(2);
j2n(n,n-2)=sqrt(2);
An=In;
An(1,1)=b;
An(n,n)=b;
Bn=-8*In+2*Jn;
Bn(1,1)=-e;
Bn(n,n)=-e;
Bn(1,2)=sqrt(2)*f;
Bn(2,1)=sqrt(2)*f;
Bn(n-1,n)=sqrt(2)*f;
Bn(n,n-1)=sqrt(2)*f;
Cn=-g*In+f*Jn;
Cn(1,1)=-a;
Cn(n,n)=-a;
Cn(1,2)=sqrt(2)*f;
Cn(2,1)=sqrt(2)*c;
Cn(n-1,n)=sqrt(2)*c;
Cn(n,n-1)=sqrt(2)*f;
Dn=20*In-8*Jn+j2n;
Dn(1,1)=k;
Dn(n,n)=k;
Dn(2,2)=19;
Dn(n-1,n-1)=19;
Dn(1,2)=-sqrt(2)*g;
Dn(2,1)=-sqrt(2)*g;
Dn(n-1,n)=-sqrt(2)*g;
Dn(n,n-1)=-sqrt(2)*g;
DDn=19*In-8*Jn+j2n;
DDn(1,1)=d;
DDn(n,n)=d;
DDn(2,2)=18;
DDn(n-1,n-1)=18;
DDn(1,2)=-sqrt(2)*g;
DDn(2,1)=-sqrt(2)*g;
DDn(n-1,n)=-sqrt(2)*g;
DDn(n,n-1)=-sqrt(2)*g;
En=k*In-e*Jn+b*j2n;
En(1,1)=2*a;
En(n,n)=2*a;
En(2,2)=d;
En(n-1,n-1)=d;
En(1,2)=-sqrt(2)*a;
En(2,1)=-sqrt(2)*a;
En(n-1,n)=-sqrt(2)*a;
En(n,n-1)=-sqrt(2)*a;
P1=kron(j2n,An)+kron(Jn,Bn)+kron(In,Dn);
P1(1:n,1:n)=En;
P1(1:n,n+1:2*n)=sqrt(2)*Cn';
P1(n+1:2*n,1:n)=sqrt(2)*Cn;
P1(n+1:2*n,n+1:2*n)=DDn;
```

```
        P1(n*(n-2)+1:n*(n-1),n*(n-2)+1:n*(n-1))=DDn;
        P1(n*(n-2)+1:n*(n-1),n*(n-1)+1:n*n)=sqrt(2)*Cn;
        P1(n*(n-1)+1:n*n,n*(n-2)+1:n*(n-1))=sqrt(2)*Cn';
        P1(n*(n-1)+1:n*n,n*(n-1)+1:n*n)=En;
        P1=P1/(h*h*h*h);
end
```

7.3.2 plot_n.m

```
% plot_n.m --- 長方形領域上の問題の差分解の描画 (Neumamm, free edge 境界条件)
%
% 使用例
%  (1) Laplacian の第 n 固有関数
%    [v,d]=eigs(eigp2nsp(nx,ny),10,0);
%    plot_n(v(:,11-n),nx,ny);
%  (2) 重 Laplacian の第 n 固有関数
%    [v,d]=eigs(plate_f1(N,0.3),200,0); 小さい方から 200 個の固有値、固有関数
%    plot_n(v(:,201-n),N,N);             (n=4 は正の最小固有値)

function plot_n(v,nx,ny)
  % メモリー中に v[0][0],v[1][0],...,v[Nx][0],v[0][1],.. と並んでいる。
  % 2 次元配列に収める
  vvv=zeros(nx+1,ny+1);
  vvv(:)=v;
  % 境界での値を修正する (角点では 2 倍することに注意)
  vvv(1,:)=vvv(1,:)*sqrt(2);
  vvv(nx+1,:)=vvv(nx+1,:)*sqrt(2);
  vvv(:,1)=vvv(:,1)*sqrt(2);
  vvv(:,ny+1)=vvv(:,ny+1)*sqrt(2);
  % mesh(), contour() には渡すには、vv は ny+1,nx+1 とする必要がある。
  vv=vvv';
  x=0:1/nx:1;
  y=0:1/ny:1;
% 左側にグラフの鳥瞰図
  subplot(1,2,1);
  colormap hsv;
  mesh(x,y,vv);
% 右側に等高線
  right=subplot(1,2,2);
  contour(x,y,vv);
  pbaspect(right,[1 1 1]);
  end
```

8 動画

実はあまり経験がないけれど、今回ちょっとやってみたので、メモを残しておく (2010 年 2 月記)。

8.1 概要

MATLAB には movie という機能がある。一度表示したイメージを保存しておいて、それを再度再生することでアニメーションを見せるというものである。movie の内容を avi ファイルに出力する movie2avi() という関数が用意されている。

それ以外に、最初にファイルをオープンしておいて、描画するたびにイメージを描き足していき、最後にファイルを閉じるというやり方もある。

8.2 movie2avi() の簡単なサンプル

draw.m

```
function draw(c)
n=10
m=100;
dt = 2 * pi / n;
x=0:2*pi/m:2*pi;
plot(x,sin(x));
for i=1:n
    t=i*dt;
    plot(x,sin(x-c*t));
    % (gcf) がないと幅・高さの警告が出る (gcf=get current figure handle)
    % gca=get current axes handle
    F(i)=getframe(gcf);
    drawnow;
end
disp('movie');
%movie(F,1);
% movie2avi() で圧縮の形式を指定しないと、デフォールトの Indeo5 が選ばれるが、
% Windows Vista では CODEC がないため、圧縮できず、avi ファイルは出来ない。
% Windows XP では一応 avi ファイルができたが、Windows MediaPlayer や
% QuickTime では表示できなかった。RealPlayer では OK だったけれど。
% そういうわけで、Mathworks 推奨の Cinepak を指定することに。
movie2avi(F,'nantoka.avi','compression','Cinepak');
```

いくつか注意点を書いておく (bad knowhow のような気がする)。

- 単に F(i)=getframe; のようにしてあるサンプル・プログラムも散見されるが、指定しなくてはいけない場合がある。
- 実際に画面に表示しているイメージを取り込んでいるらしく、描画中に他の作業をしていると、その内容が動画ファイルに入ってしまうことがある。従って、drawnow (今描け) という命令をつけるのは、バッファリングせずにアニメーションをちゃんと見せると

いう意味もあるが、動画ファイルを作るためにも必須である。なお、スクリーン・セーバーにも気をつける必要があるらしい (何だかなあ…少しお粗末のような気が)。

- `movie2avi()` で出力可能なフォーマットは環境に依存する。少なくとも Windows 版 MATLAB の古いバージョンでは、`Indeo5` というフォーマットがデフォールトであったようだが、現在の Windows には CODEC が含まれなくなっている。そのため上のサンプル・プログラムでは、`'compression','Cinepak'` としている (この選択は Mathworks 社の推奨)。

- 他に使いそうなオプションとして

  fps frame per second. `'fps',5` とすると毎秒 5 フレームになる。デフォールトのフレーム・レートは 15 である。

  quality 圧縮する場合、どの程度きれいにするかはサジ加減ということになるが、デフォールトは 75% で、それを変えたい場合は `'quality',90` のようにして数値を指定する。

- なお Mac 向けの古いバージョンでは、一切の圧縮がサポートされていなかった (思わず目を疑ってしまったが)。

8.3 PowerPoint で使う

avi ファイルが出来た。Windows Media Player でも再生できた。これで一件落着だ、と思ったがそれで済まないことがあった。動画ファイルを PowerPoint に貼り付けたいという人が多くて (個人的には PowerPoint にあまり親近感を持っていないのだけど)、うまく再生できません、という訴えがあった。調べてみると、PowerPoint では、内部で Media Player というプログラム (紛らわしいけれど、Windows Media Player ではない！) を呼び出しているが、それが Windows Media Player よりも「弱い」部分があるということらしい。これについて、http://support.microsoft.com/kb/921249/ja に載っている解決策を採用して何とか乗り切れた。それは Windows ムービーメーカーを用いて、Windows Media Video 形式 (.wmv) に変換するというものである。手順は

1. [ビデオの取り込み] の [ビデオの読み込み] で、avi ファイルを選択する。

2. [コレクション] で指示されているように、クリップを下のストーリーボードにドラッグ・アンド・ドロップする。

3. [ムービーの完了] の [コンピュータに保存] を選択する。[ムービーの設定] の [その他の設定] で適当な画質を選択し (ちなみに「高画質ビデオ (大)」を選択した)、ファイルに書き出す。

Media Player も、流石に Windows Media Video 形式にはきちんと対応しているということらしい。

新しいのを買ってみたいな…

9　行列の解析的性質で遊ぶ

行列の簡単な解析的性質の実験をしてみよう。

ここでいう解析的性質とは、行列の極限にかかわる性質のことを意味している。たとえば、行列の列 A_n が $n \to \infty$ のときの極限とか、無限級数 $\sum_{n=1}^{\infty} A_n$ とか、行列値関数 $f(A)$ の連続性などの話である。線型作用素の解析的性質は通常は関数解析で学ぶが、(行列の段階での)初等的な解説は杉浦・横沼 [8] でも読める。

9.1　行列の等比数列

一つの正方行列 A の冪 A^n で作られる行列の列、いわば行列の等比数列

$$A^0 = I, A^1 = A, A^2, \cdots, A^n, A^{n+1}, \cdots$$

の極限はどうなるだろうか？既に学んだかもしれないが (行列の Jordan 標準形の簡単な応用である)、ここでは実験で試してみよう。

```
isc-xas05% octave
octave:1> n=5
octave:2> a=rand(n,n)
octave:3> a*a
octave:4> a^2
octave:5> a^100
octave:6> a^1000
```

Octave では、行列 a の n 乗は a^n で計算できることがわかるが、この例では n の増加と共に急速に大きくなり、あっと言う間にオーバーフローすることが分かる。

種明かしをすると、行列の固有値の絶対値 (これをスペクトル半径と呼ぶ) を調べると事情が見えてくる。

```
octave:7> eig(a)
octave:8> r=max(abs(eig(a)))
octave:9> a=a/r
octave:10> max(abs(eig(a)))
octave:11> a^100
octave:12> a^1000
```

Octave では max(abs(eig(a))) で a のスペクトル半径が計算できる。a をそのスペクトル半径 r で割ることで、スペクトル半径を 1 に縮めることができる。a^100, a^1000 ともにオーバーフローしていないが、それだけでなく何かが起こっていることに気がつくだろうか？(どうしてそうなるのか考えてみよう。)

今度はスペクトル半径が 1 より小さい行列の冪を調べてみよう。

```
octave:13> a=0.9*a
octave:14> a^100
octave:15> a^1000
octave:16> a^10000
```

9.2 行列の等比級数

今度は正方行列 A (ただし A のスペクトル半径は 1 より小さいとする) の等比級数

$$I + A + A^2 + A^3 + \cdots = \sum_{n=0}^{\infty} A^n \quad (I \text{ は単位行列})$$

を考える。この級数は解析学では **Neumann 級数**と呼ばれる。

以下の実験は前節の続きで行うことを想定している。Octave を終了してしまっている場合は、

この節の実験に先立ち必要なこと

```
n=5
a=rand(n,n)
r=max(abs(eig(a)))
a=a/r
a=0.9*a
```

を実行しておこう。

Neumann 級数の部分和

$$S_n = \sum_{k=0}^{n} A^k$$

を計算する Octave プログラム `neuamnn.m` を用意した。

neumann.m

```
 1  % neumann.m --- 行列の Neumann 級数（等比級数）の第 N 部分和
 2  function s = neumann(a,N)
 3    [m,n] = size(a);
 4    if m ~= n
 5      disp('a が正方行列でない！');
 6      return
 7    end
 8    % 第 0 項  S_0 = I
 9    s = eye(n,n);
10    % 第 1 項  S_1 = I + a
11    t = a; s = s + t;
12    % 第 2〜N 項まで加える（t が a^n になるようにしてある）
13    for k=2:N
14      t = t * a;
15      s = s + t;
16    end
```

これを ~re00018/neumann.m から、Octave を実行するディレクトリィにコピーしよう。

こうしてコピーする

```
isc-xas05% cp ~re00018/neumann.m .
```

```
octave:17> c=eye(n,n)-a
octave:18> b=neumann(a,100)
octave:19> b*c
octave:20> b=neumann(a,1000)  ← もう少し精度を上げてみる
octave:21> b*c
octave:22> format long        ← お望みなら表示桁数を上げて
octave:23> b*c
```

Neumann 級数の (部分) 和 $S_N = \sum_{k=0}^{N} \mathtt{a}^k$ に $C = I - \mathtt{a}$ をかけた結果が単位行列に非常に近いことが分かる。種明かしをすると、

$$\sum_{n=0}^{\infty} A^n = (I - A)^{-1}$$

が成り立つ。これは等比級数の和の公式

$$\sum_{n=0}^{\infty} r^n = \frac{1}{1-r} \quad (\text{ただし } r \text{ は } |r| < 1 \text{ を満たす複素数})$$

の一般化である。

9.3 絶対値最大の固有値を求める — 冪乗法

簡単のため n 次正方行列 A が対角化可能で、その固有値を $\lambda_1, \cdots, \lambda_n$, それらに属する固有ベクトルを $u_1, \cdots, u_n$ とする。さらに $|\lambda_1|$ は他の固有値の絶対値よりも大きいとする。

$$|\lambda_1| > |\lambda_i| \quad (i = 2, 3, \cdots, n)$$

任意のベクトル $x \in \mathbf{C}^n$ は

$$x = c_1 u_1 + c_2 u_2 + \cdots c_n u_n$$

と展開できるが、A^k を作用させると

$$A^k x = c_1 A^k u_1 + c_2 A^k u_2 + \cdots C_n A^k u_n = c_1 \lambda_1^k u_1 + c_2 \lambda_2^k u_2 + \cdots C_n \lambda_n^k u_n.$$

k が大きいとき、右辺第 1 項は右辺の他の項と比べて大きくなることが分かる (ただし $c_1 \neq 0$ とする)。k を十分大きくすると、右辺第 2 項以下は第 1 項と比べて無視できるほど小さくなるだろう。すると $A^k x$ は u_1 の定数倍、すなわち λ_1 に属する固有ベクトルに近くなるはずである。

以上のことを Octave による計算で確かめるためには、$A^k x$ がオーバーフローすることを防ぐため、代りにその長さで割った $\dfrac{A^k x}{\|A^k x\|}$ を作ればよい。

以下では素朴に A をかけていくことで $\dfrac{A^k x}{\|A^k x\|}$ を求めている。

```
octave:5>  x=ones(n,1)
octave:5>  for i=1:100
>  y=a*x
>  x=y/norm(y)
>  end
octave:5>  a*x ./ x        ← a*x の各成分を対応する x の成分で割ってみる
octave:5>  eig(a)          ← 念のため eig() で a の固有値を調べて比較
```

線形代数では、固有値を固有多項式の根として特徴づけるが、普通固有多項式を数値計算で解くのは難しいので、行列の問題のまま各種の反復法を用いることになる。上で見た方法は『冪乗法』と呼ばれ、多くの方法の基礎となっている。

10 リンク

1. 『MATLAB ホームページ』http://www.cybernet.co.jp/matlab/
   日本での MATLAB 取り扱い業者のページ。FAQ など日本語で色々な情報が得られる。

2. 『Octave Home Page』http://www.octave.org/

3. 『Scilab Home Page』http://www-rocq.inria.fr/scilab/

4. 『MATLAB Clones』http://www.dspguru.com/sw/opendsp/mathclo2.htm
5. 『Rlab』http://rlab.sourceforge.net/
6. 『Euler』http://mathsrv.ku-eichstaett.de/MGF/homes/grothmann/euler/index.html
   ドイツ製。区間演算サポート。
7. 『ATLAS』

   ftp://ftp.u-aizu.ac.jp/pub/SciEng/numanal/netlib/atlas/
8. 『数値演算言語 Octave』

   http://adlib.rsch.tuis.ac.jp/~akira/unix/octave/index-j.html
9. 『ATLAS による Octave の高速化』

   http://mackako.im.uec.ac.jp/chiba-f/octave/octave_speed_up_by_atlas_j.html
10. 『Scilab を中心とした MATLAB クローン即席入門講座』

    http://www.bekkoame.ne.jp/~ponpoko/Math/Scilab.html

参考書案内

菊地・山本 [7] には、正方形領域における Laplacian の固有値問題を差分法で解く方法の解説が載っている (プログラムは FORTRAN である)。

Kronecker 積 ($\otimes$) については、伊理 [2] を参照せよ。

Octave, Scilab については、WWW 上に解説文書があふれているが、日本語で読める書籍としては 大石 [3] がある。

参考文献

[1] 有木進, 工学のための線形代数, 日本評論社 (2000).

[2] 伊理正夫, 一般線形代数, 岩波書店 (2003).
伊理正夫, 線形代数 I, 岩波講座 応用数学, 岩波書店 (1993) のリニューアル。

[3] 大石進一, Linux 数値計算ツール, コロナ社 (2000).

[4] 大石進一, MATLAB による数値計算, 培風館 (2001).

[5] 小国力/Dongarra, Jack J., MATLAB による線形計算ソフトウェア, 丸善 (1998).

[6] 小国力, MATLAB と利用の実際 — 現代の応用数学と CG —, サイエンス社 (1995).

[7] 菊地 文雄, 山本 昌宏, 微分方程式と計算機演習, 山海堂 (1991).

[8] 杉浦光夫, 横沼健雄, Jordan 標準形・テンソル代数, 岩波書店 (1990).

11　グラフ描き

もちろん MATLAB, Octave にもグラフ描画機能がある。

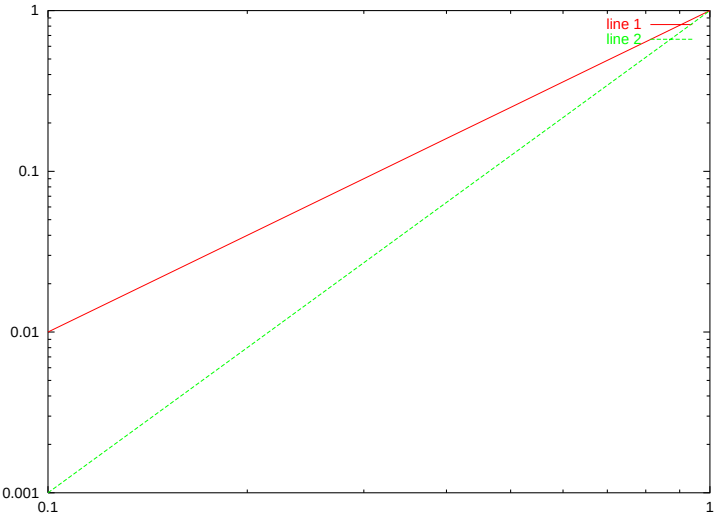
MATLAB, Octave 共通

```
octave:1> x=0:0.1:1          → x = (0, 0.1, 0.2, ··· , 1) となる
octave:2> y=x .* x           ← 成分ごとに 2 乗
octave:3> y=x .* x .* x      ← 成分ごとに 3 乗
octave:4> plot(x,y,x,z)      → y = x^2, y = x^3 のグラフの出来上がり
octave:5> loglog(x,y,x,z)    ← 両側対数目盛を指定
octave:6> semilogx(x,y,x,z)  ← 片側対数目盛を指定 (この場合ナンセンス)
```

古い Octave は gnuplot を呼び出しているので、gnuplot 風の命令が利用できた (もう時代遅れだが、古いスクリプトを読む必要があるだろうから、残しておく)。

Octave 専用

```
octave:1> x=(0:0.1:1)'
octave:2> y=x .* x
octave:3> y=x .* x .* x
octave:4> data=[x,y,z]
octave:5> gplot data, data using 1:3
octave:6> gset logscale xy
octave:7> gplot data, data using 1:3
octave:8> gset term postscript eps color
octave:9> gset output "mygraph.ps"
octave:10> replot
```



12 MATLAB, Octave についてプログラミング上のヒント

12.1 課題の行列を変数に設定するプログラム

50 次の正方行列 A' を

$$A' = \begin{pmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{pmatrix}$$

で定義し、A を

$$A = \begin{pmatrix} c_1 A' & O \\ O & c_2 A' \end{pmatrix}$$

とおくのであった。

この A を作るプログラムをいくつか書いてみよう。

12.1.1 まず A' を作ってから A を作る

定義に素朴に従って、A' を作るところから始めてみよう

A' を MATLAB で作るには (似た例を既にプリントに出してあるけれど)、例えば

```
m=50;
Ap=4*eye(m,m)-diag(ones(m-1,1),1)-diag(ones(m-1,1),-1);
```

とすれば OK. それをもとに A を作るにはどうするか？

B から A を作る方法 (1)

```
c1=1; c2=10
A=[c1*Ap zeros(m,m); zeros(m,m) c2*Ap];
```

または

B から A を作る方法 (2)

```
c1=1; c2=10;
n=2*m;
A=zeros(n,n);
A(1:m,1:m)=c1*Ap;
A(m+1:n,m+1:n)=c2*Ap;
```

12.1.2 C プログラム流に成分を指定することで A を作る

MATLAB プログラムでも、C や Fortran と同様に成分を直接指定することが出来る。

```
n = 2 * m;
a = zeros(n,n);
% 前半の m 行
d = 4 * c1; od = - c1;
a(1,1) = d; a(1,2) = od;
for i=2:m-1
  a(i,i-1) = od; a(i,i) = d; a(i,i+1) = od;
end
a(m,m-1) = od; a(m,m) = d;
% 後半の m 行
d = 4 * c2; od = - c2;
a(m+1,m+1) = d; a(m+1,m+2) = od;
for i=m+2:n-1
  a(i,i-1) = od; a(i,i) = d; a(i,i+1) = od;
end
a(n,n-1) = od; a(n,n) = d;
```

普通は、MATLAB のようなインタープリターで、このように `for` 等の繰り返し命令を使うと効率上の問題が発生すると言われているが、この場合は案外と効率がよい。ループが一重のものだけだからであろう。

12.2 自分で関数を作ろう

MATLAB, Octave にはあらかじめ豊富な関数が揃っているが、自分でも作れる。
関数は、関数名 + ".m" という名前のファイルにその定義を書くことで定義できる。
次に掲げるのは前回配ったプログラム (に少し注釈を加えたもの) である。

cg1.m

```
% cg1.m --- CG 法で A x = b を解くための関数 cg1()
%  2003/5/15 作成, 5/21 修正
%
% 【使い方】
%    x = cg1(A, b, 初期ベクトル, 相対残差)

% 【例】
%  まず問題を用意
%   n=2;a=[2 1;1 3];exact=ones(n,1);b=a*exact
%  初期ベクトルを 0 ベクトル、相対残差を 10^{-12} にして実行
%   cg1(a,b,zeros(n,1),1e-12)
%

function x = cg1(a,b,x0,epsilon)
  x = x0;
  r = b - a * x;
  p = r;
  eps_b = epsilon * norm(b);
  k = 0;
  while norm(r) > eps_b
    ap = a * p;
    pap = ap' * p;
    alpha = p'*r/pap;
    x = x + alpha*p;
    r = r - alpha*ap;
    beta = - (ap'*r)/pap;
    p = r + beta * p;
    k=k+1;  % 最初 k++ としていましたが、それは MATLAB ではダメです
    % 以下の 3 行は中間結果の表示用（不要ならば削除すればよい）
    k
    x
    r
  end
```

- 縦ベクトル x, y の内積は y'*x ($\because x \cdot y = y^T x$)
- ベクトル x の Euclid ノルムは norm(x) (ちなみに p ノルムは norm(x,p) で、max ノルムは norm(x,inf)

12.2.1 問

課題の行列を求める関数 testmat(m,c1,c2) を書け。いくつか書いて実行速度を比べよ。

13 Tips

必要があって調べたことでも、忘れてしまって、後で「あれはどうだったかな？」が多い。定着するまではメモが必要だ。

- 自宅から大学のマシンにアクセスして長時間計算させるときは、MATLAB を走らせるマシンにリモート・ログインして、screen を起動し、その中で `matlab -nodisplay` で起動する。当然グラフィックスは使わない。後から `screen -r` とするわけだ。

- `size()` は引数 1 個で読んだ場合、行列の行の個数、列の個数を返す。

```
[m,n]=size(A);

m=size(A,1);
n=size(A,2);

[n,dummy]=size(A);
```

- 文字列リテラルを作る引用符が何故か single quotation mark である。

```
 s='Hello';
```

- 文字列の連結は何か不思議なやり方で扱う。

```
object='pen';
str=['This ' 'is ' 'a ' object '.']
```

- 数値の文字列への変換は、`num2str()`, `int2str()` という関数を使う。自分ではまだ使ったことがないが、`sprintf()` というのも便利かな？

- 格子点の座標 (n 等分点) を作るときの決まり文句: `x=a:(b-a)/n:b;`
(と思っていたのだけれど、`linspace(a, b, n+1);` とすべきかも)

- 鳥瞰図は `mesh()`, 等高線は `contour()`

```
x=a:(b-a)/nx:b;
y=c:(d-c)/ny:d;
u=zeros(ny+1,nx+1);
...
mesh(x,y,u); または contour(x,y,u);
```

- 鳥瞰図と等高線を並べる。ここはサンプル・プログラムを

plot_n.m

```
% plot_n.m --- 長方形領域上の問題の差分解の描画 (Neumann, free edge 境界条件)
%
% 使用例
%  (1) Laplacian の第 n 固有関数
%    [v,d]=eigs(eigp2nsp(nx,ny),10,0);
%    plot_n(v(:,11-n),nx,ny);
%  (2) 重 Laplacian の第 n 固有関数
%    [v,d]=eigs(plate_f1(N,0.3),200,0); 小さい方から 200 個の固有値、固有関数
%    plot_n(v(:,201-n),N,N);              (n=4 は正の最小固有値)

function plot_n(v,nx,ny)
  % メモリー中に v[0][0],v[1][0],...,v[Nx][0],v[0][1],.. と並んでいる。
  % 2次元配列に収める
  vvv=zeros(nx+1,ny+1);
  vvv(:)=v;
  % 境界での値を修正する (角点では2倍することに注意)
  vvv(1,:)=vvv(1,:)*sqrt(2);
  vvv(nx+1,:)=vvv(nx+1,:)*sqrt(2);
  vvv(:,1)=vvv(:,1)*sqrt(2);
  vvv(:,ny+1)=vvv(:,ny+1)*sqrt(2);
  % mesh(), contour() には渡すには、vv は ny+1,nx+1 とする必要がある。
  vv=vvv';
  x=0:1/nx:1;
  y=0:1/ny:1;
% 左側にグラフの鳥瞰図
  subplot(1,2,1);
  colormap hsv;
  mesh(x,y,vv);
% 右側に等高線
  right=subplot(1,2,2);
  contour(x,y,vv);
  pbaspect(right,[1 1 1]);
  end
```

- Mathematica に差分解を持って行く時のプログラム例。MATLAB は大きいデータ問題ないが、Mathematica では暴走したりするので (2012年現在)、間引いている (`i=1:N/160:N+1;` として、`u=u(i,j);` とすると、行数は 161 になる。)。

```
% dividedata.m --- free_某_1280.mat を分割して poisson_某/u 番号.dat
% 2012/10/13 最初のバージョン (0,0.1,0.2,0.25,0.3,0.35,0.5 で実行する)
% 2012/11/17 小修正
% written by Masashi Katsurada

clear
N=1280;
i=1:N/160:N+1;
j=1:N/160:N+1;
muarray=[0 0.1 0.2 0.25 0.3 0.35 0.5];
maxk=size(muarray,2);
for k=1:maxk
  mu=muarray(k);
  mustr=num2str(mu);
  % mustr 0 0.1 0.2 0.25 0.3 0.35 0.5
  load(['free_' mustr '_1280.mat'])
  for n=1:200
    u=nv_to_dim2(v1280(:,201-n),N,N);
    u=u(i,j);
    save(['poisson_' mustr '/u' int2str(n) '.dat'], 'u', '-ascii')
  end
  e=diag(d1280);
  e=e(200:-1:1);
  save(['poisson_' mustr '/eigen.dat'],'e','-ascii')
end
```

- たくさんグラフィックスのウィンドウ (figure(某)) を出した時は，close(番号) や close all で掃除する。

- ライセンス番号が知りたいとき，

```
>>> license
ans =
むにゃむにゃ
```

- MATLAB を Mac にインストールして，Mac の OS を 10.8 にバージョン・アップすると，アクティベーションのやり直しが必要になる。そのためにはディアクティベーションが必要で，少し面倒である。OS をバージョンアップする前にディアクティベーションをするのが簡単か。

A　数値計算ソフトウェアの発展 (駆け足説明)

主に行列計算関係に焦点を当てて説明する。

A.1　数値計算ライブラリィ

数値計算ライブラリィの歴史 (概略)

1. **サブルーチン**[a](**subroutine**) の誕生、**サブルーチン・ライブラリィ**の誕生
2. (汎用) プログラミング言語[b]の誕生 (**FORTRAN**[c], **LISP** などが最初の例)
3. 固有値計算ライブラリィ **EISPACK**
   (論文誌 “Numerische Mathematic” で発表されたアルゴリズムを元に最初は **ALGOL** で書かれ、後に FORTAN に移植される。主宰者は有名な数値解析学者である Wilkinson である。)
4. 連立 1 次方程式の解法ライブラリィ **LINPACK**
   途中から **BLAS** が生まれ、LINPACK は BLAS の上に構築される。
5. 線形計算ライブラリィ **LAPACK**
   (**メモリー階層**を考慮した BLAS を全面的に採用、EISPACK & LINPACK の現代化)
6. 他のプログラミング言語への移植 — **TNT**[d] (C++) など。

[a]機械語 (machine language) や、Fortran 言語における、あるまとまった処理をするプログラムの単位を呼ぶ言葉。C 言語における「関数」に相当すると考えて構わない。

[b]当時は「自動プログラミング言語」と呼ばれたそうである。

[c]FORTRAN は、IBM が線形計画法のプログラムを効率的に作成するために開発した言語であると言われている。

[d]C++ 向けには **LAPACK++** があったが、C++ 言語の ANSI 規格の進展に伴い、新しく設計し直されたのが TNT である。まだ発展途上で、LAPACK の機能のすべては実装されていない。

ここで名を紹介した EISPACK, LINPACK, LAPACK, TNT はいずれもソースが公開されているフリーソフトである[11]。

数値計算ライブラリィを採用で実現できること

(1) 高い生産性

(2) 高い信頼性 (バグが少ない、高精度、条件が悪い問題でも崩れないタフさ)

(3) 高い効率性 (速度、メモリー利用効率)

[11]このあたりに欧米文化の強さが感じられる。ここで紹介したソフトの中には、博士号クラスの研究者が数十人、何年も作業して始めて開発できたものもある。日本でも大学を中心に様々なライブラリィの開発がされたが、全面公開までこぎつけたものは少なく (途中で企業に売ってしまったものもある)、大変もったいない事態になっていると筆者は感じている。こうなってしまった背景には、ソフトウェアの開発を研究業績とは認めない風潮など、二三の理由が考えられる…(脱線)

なぜ高い実行効率が得られるか

速度をあげるために考えねばならないこととして、講義では**計算量**を説明した。これらのソフトウェアでは計算量の観点から無駄がないことはもちろんであるが、それ以外にも重要な要素がある。

1. ループのアンロールなどのテクニック
2. メモリー階層を考慮したプログラミング

これらを追求すると、利用するコンピューター・システムに合わせたプログラムのチューニングが必要になり、プログラムの汎用性が低くなる恐れがある。しかし、システムごとにチューニングが必要な部分を小さな部分に凝縮し、他と分離することにより、この問題をある程度解決できる。LAPACK については、BLAS がこのチューニングが必要な部分を担当している。LAPACK に付属する BLAS は "reference (参考) BLAS" と呼ばれ、FORTRAN で書かれているので移植性があるが、高速な計算を望む場合は最適化された BLAS に差し換えることになる。例えば Sun Workstation の場合、Sun Microsystem 製ソフトウェア開発環境 Sun Workshop では、コンパイラーと一緒に BLAS が (と実は LAPACK も) 提供されている)。Windows や Intel CPU 向けの Linux の場合は Intel から BLAS が提供されている (無償)。これらは人手で最適化されたものである (と思われる) が、色々なパラメーターを変化させながら性能を測定することで、最適なパラメーターを実験的に「算出」し、その結果を元に BLAS プログラムを自動生成するソフトウェアもある (例えばフリーソフトの **ATLAS**)。

B　ループのアンロールの効果の実験

4 年くらい前に購入したノートパソコン上の FreeBSD で、実験した結果。何も工夫しない場合 `testddot.c` に比べて、ループのアンロールをした場合 `testddot2.c` はスピードが 4 倍以上になる。

```
#include <stdio.h>
#define N 10000

/*
 * 10k*10k=100M 回の乗算
 * 10k*10k=100M 回の加算
 * 200M 回の浮動小数点演算
 * 素朴にコンパイル 47.545u -> 4.2MFLOPS
 * -O4               7.611u -> 26.3MFLOPS
 * -O                9.061u -> 22.1MFLOPS
 */

double ddot(int n, double *x, double *y)
{
  int i;
  double s = 0;
  for (i = 0; i < N; i++)
    s += x[i] * y[i];
  return s;
}

int main()
{
  int i;
  double x[N], y[N], z[N];
  for (i = 0; i < N; i++) {
    x[i] = rand();
    y[i] = rand();
  }
  for (i = 0; i < N; i++)
    z[i] = ddot(N, x, y);
}
```

```
#include <stdio.h>
#define N 10000

/*
 * 10k*10k=100M 回の乗算
 * 10k*10k=100M 回の加算
 * 200M 回の浮動小数点演算
 * 素朴にコンパイル   9.483u ->  21.1 MFLOPS
 * -O                 1.940u -> 103.0 MFLOPS
 * -O4                1.924u -> 104.0 MFLOPS
 */

double ddot(int n, double *x, double *y)
{
  int i, n_div_8 = n / 8;
  double s = 0;
  for (i = 0; i < n_div_8; i++)
    s += x[i] * y[i]
       + x[i+1] * y[i+1]
       + x[i+2] * y[i+2]
       + x[i+3] * y[i+3]
       + x[i+4] * y[i+4]
       + x[i+5] * y[i+5]
       + x[i+6] * y[i+6]
       + x[i+7] * y[i+7];
  for (i = 8 * n_div_8; i < n; i++)
    s += x[i] * y[i];
  return s;
}

int main()
{
  int i;
  double x[N], y[N], z[N];
  for (i = 0; i < N; i++) {
    x[i] = rand();
    y[i] = rand();
  }
  for (i = 0; i < N; i++)
    z[i] = ddot(N, x, y);
}
```

C　参考書案内

Octave, Scilab については、WWW 上に解説文書があふれているが、日本語で読める書籍としては 大石 [3] がある。

参考文献

[1] 有木進, 工学のための線形代数, 日本評論社 (2000).

[2] 伊理正夫, 一般線形代数, 岩波書店 (1993). 伊理正夫, 線形代数 I, 岩波講座 応用数学, 岩波書店 (1993).

[3] 大石進一, Linux 数値計算ツール, コロナ社 (2000).

[4] 大石進一, MATLAB による数値計算, 培風館 (2001).

[5] 小国力/Dongarra, Jack J., MATLAB による線形計算ソフトウェア, 丸善 (1998).

[6] 小国力, MATLAB と利用の実際 — 現代の応用数学と CG —, サイエンス社 (1995).

[7] 菊地 文雄, 山本 昌宏, 微分方程式と計算機演習, 山海堂 (1991).

D 線形計算とは

D.1 線形代数の現状

現在の日本の大学の理工系のカリキュラムでは、1, 2 年次に「線形代数」 (linear algebra) を学ぶことになっている[12]。その主たるテーマは有限次元の線型空間とその間の線型写像の理論であるが[13]、ここでは連立 1 次方程式 $Ax = b$, 固有値問題[14] $Ax = \lambda x$ の解法に焦点を当ててみよう。

(念のために注意しておくと、連立 1 次方程式と固有値問題だけが重要なのではない。ともすると試験問題の花形なのでそう思ってしまう人もいると思うが…)

連立 1 次方程式の「解き方」としては、現在普通の線形代数の本では、

(1) 逆行列を用いる (逆行列については、行列式を使った公式があげられている)

(2) Cramer の公式を用いる

(3) 「掃き出し法」[15]を用いる

などが説明されているが、これらは、計算の効率 (なるべく少ない計算量で計算する) と数値的安定性に問題がある。特に、

連立 1 次方程式を解くために逆行列を求めてはいけない！

[12]実は「線形 (型) 代数」という名前はそれほど古くからあるものではない。かつては、「代数と幾何」、「行列と行列式」、「線形数学」というような名前の本があったが (ちなみに、筆者が学生の時の講義名は「代数と幾何」だった)、現在では「線形 (型) 代数」に収束したようである。個人的には収束と同時に「幾何」の匂いが薄れてきたように感じている (みんな「代数」だと信じるようになったのかな — 例えば 2 次, 3 次の行列式が、それぞれ平行四辺形の符号付き面積、平行六面体の符号付き体積を表わす (もちろん、これらはさらに一般の次元に拡張される) ということを書いていない本が存在するが、証明しなくても事実は教えておいた方が良いのにと思う)。解析屋としては、計量 (内積やノルム) の話があまり出て来ないことが残念である。

[13]ちなみに、解析的にこのテーマを取り扱った “Finite dimensional vector spaces” というタイトルの有名な面白い本がある。

[14]応用上は、一般化固有値問題という、行列 A, B が与えられたときに、$Ax = \lambda Bx$, $x \neq 0$ を満たす λ, x を求める問題も重要 (分野によっては、標準固有値問題なんて目にしないことも) である。

[15]数値線形代数では、**Jordan (ヨルダン) の消去法**と呼ぶ。

また固有値の求め方としては、線形代数の本では

命題 D.1 n 次正方行列 A の固有値は、λ に関する代数方程式

$$\det(\lambda I - A) = 0$$

の根である。

という定理を述べて解決ということにしてある。この方法 (固有多項式に帰着する方法) では、実際にはごくごく小規模な問題しか解くことができない。

行列の固有値を求めるために固有方程式を解いてはいけない！

D.2 連立 1 次方程式の解法概説

コンピューターで行われる数値計算の時間の 90% は連立 1 次方程式を解いている時間だという説があるくらい、連立 1 次方程式は頻出する問題である。例えば微分方程式の問題も、離散化 (有限次元近似) された後は連立 1 次方程式を解くことに帰着されることが多い[16]。ところがこのように工学的な応用で、連立 1 次方程式を解くために実際に利用されている方法は (ほとんどの) 線形代数の本には載っていない。

実際に利用される解法は色々あるが、大きく次の二つに分類される (直接法以外知らない、見当もつかない人が多いと思われるが、それで構わない)。以下では直接法について、やや詳しく説明する。

直接法 (exact method) もし個々の演算に丸め誤差がなければ、有限回の演算で解が得られる方法

- Gauss の消去法 (LU 分解), 対称な問題に対する Cholesky 分解法, QR 分解に基づく方法などある。

反復法 (iterative method) 有限回の演算では真の解が得られないかもしれないが、反復するごとに近似解の精度を上げて行き、十分な精度が得られたところで打ち切る方法

- 古くからある**定常反復法** (Jacobi 法, Gauss-Seidel 法, SOR 法など) と**非定常反復法** (CG 法とその改良版, 親類) に分類される[17]。
- 計算のためには行列 A の成分そのものを知る必要はなく、任意に与えたベクトル x との積 Ax が分かれば良いという特徴を持ち、行列の疎性[18]を利用しやすい。

[16] 昨年書かれた文章によると、$n = 10^8$ (1 億) の問題が解かれているとか。
[17] 今のところ、対称な問題に対しては、CG 法の系統の解法がほとんど決定版とみなされている。
[18] 行列の成分に 0 が多いとき、行列は疎 (sparse) であるという。微分方程式の離散化で現われる連立 1 次方程式の係数行列は、ほぼ例外無く疎行列である。

D.3 固有値問題の解法概説

(授業では説明する時間的余裕はないと思うが…)

固有値を求めるという問題は代数方程式に帰着されるわけだが、その逆に任意の代数方程式の問題は行列の固有値を求めるという問題に帰着できる (この事実はしばしば常微分方程式の本に書いてある)。

それゆえ、**ある意味で固有値問題は代数方程式と等価であり**、例えば「5 次以上の代数方程式の、四則とべき根を有限回用いた根の公式は存在しない」という有名な定理から、5 次以上の行列の固有値問題は、有限回の四則演算とべき乗演算では解けないことが分かる。それゆえ連立 1 次方程式の解法と対比してみると、(5 次以上の) **固有値問題には直接法は存在せず、反復法を使うしかない**ことが分かる。

もう一つ重要なのは、行列から固有多項式を作ると、もともとの行列が持っていた情報の多くが消えてしまうということである。例えば対称な固有値問題は、そうでない問題よりもうまく (速く安定に) 解けるのだが、それを固有多項式にしてしまうと、対称性のうま味がなくなってしまう。その他にも理由があって、結局は

固有値を求めるのに固有方程式を解こうとしてはいけない！

それでは実際にどうするか？現在では次の 2 ステップで攻略するのが良いと言われている。

(1) 与えられた行列 A を、実直交行列 U で相似変換して、Hessenberg 行列 (あるいは A が対称の場合は三重対角行列) $\widetilde{A}$ に変換する (つまり $\widetilde{A} = U^T A U$)。

(2) $\widetilde{A}$ に対して、

   (a) 冪乗法の系統の方法 (逆反復法, シフト法, 減次)

   (b) Strum の方法 (bisection method)

   (c) QR 法

   などの反復法を施して、固有値を求める。

$A = (a_{ij})$ が **Hessenberg 行列**とは、

$$i > j + 1 \quad \Longrightarrow \quad a_{ij} = 0$$

が成り立つこと (対角線より二つ以上下は 0) を言う。大ざっぱに言うと、後一歩で上三角行列になる行列ということである。

D.4 Gauss の消去法

Gauss の消去法というのは、要するに中学校で習った (未知数を一つ一つ消去して減らしていく) 消去法である。

例として次の方程式を取りあげて説明しよう。

$$(3)\qquad \begin{cases} 2x_1 + 3x_2 - x_3 = 5 \\ 4x_1 + 4x_2 - 3x_3 = 3 \\ -2x_1 + 3x_2 - x_3 = 1. \end{cases}$$

線形代数で習う掃き出し法では、係数行列と右辺のベクトルを並べた行列を作り、それに

(1) ある行に 0 でない定数をかける。

(2) 二つの行を入れ換える。

(3) ある行に別の行を加える。

のような操作 — **行に関する基本変形**と呼ぶ — をほどこして、連立方程式の係数行列に相当する部分を単位行列にするのであった。

$$\begin{pmatrix} 2 & 3 & -1 & 5 \\ 4 & 4 & -3 & 3 \\ -2 & 3 & -1 & 1 \end{pmatrix} \to \begin{pmatrix} 1 & \frac{3}{2} & -\frac{1}{2} & \frac{5}{2} \\ 4 & 4 & -3 & 3 \\ -2 & 3 & -1 & 1 \end{pmatrix} \to \begin{pmatrix} 1 & \frac{3}{2} & -\frac{1}{2} & \frac{5}{2} \\ 0 & -2 & -1 & -7 \\ 0 & 6 & -2 & 6 \end{pmatrix} \to$$

$$\to \begin{pmatrix} 1 & \frac{3}{2} & -\frac{1}{2} & \frac{5}{2} \\ 0 & 1 & \frac{1}{2} & \frac{7}{2} \\ 0 & 3 & -1 & 3 \end{pmatrix} \to \begin{pmatrix} 1 & 0 & -\frac{5}{4} & -\frac{11}{4} \\ 0 & 1 & \frac{1}{2} & \frac{7}{2} \\ 0 & 0 & -\frac{5}{2} & -\frac{15}{2} \end{pmatrix} \to \begin{pmatrix} 1 & 0 & -\frac{5}{4} & -\frac{11}{4} \\ 0 & 1 & \frac{1}{2} & \frac{7}{2} \\ 0 & 0 & 1 & 3 \end{pmatrix} \to$$

$$\to \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{pmatrix}, \quad \text{ゆえに} \begin{pmatrix} x_2 \\ x_3 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}.$$

ガウスの消去法も、前半の段階はこの方法に似ていて、同様の変形を用いて掃き出しを行なうのだが、以下のように対角線の下側だけを 0 にする。

$$\begin{pmatrix} 2 & 3 & -1 & 5 \\ 4 & 4 & -3 & 3 \\ -2 & 3 & -1 & 1 \end{pmatrix} \to \begin{pmatrix} 2 & 3 & -1 & 5 \\ 0 & -2 & -1 & -7 \\ 0 & 6 & -2 & 6 \end{pmatrix} \to \begin{pmatrix} 2 & 3 & -1 & 5 \\ 0 & -2 & -1 & -7 \\ 0 & 0 & -5 & -15 \end{pmatrix}.$$

最後の行列は

$$2x_1 + 3x_2 - x_3 = 5, \quad -2x_2 - x_3 = -7, \quad -5x_3 = -15$$

ということを表しているので、後の方から順に

$$x_3 = \frac{-15}{-5} = 3, \quad x_2 = \frac{-7 + x_3}{-2} = 2, \quad x_1 = \frac{5 - 3x_2 + x_3}{2} = \frac{5 - 3 \times 2 + 3}{2} = 1$$

と解くことが出来る。前半の対角線の下側を 0 にする掃き出しの操作を**前進消去** (forward elimination)、後半の代入により解の値を求める操作を**後退代入** (backward substitution) と呼ぶ。

D.5　行列の分解について

D.2 で行列の色々な「分解」が出て来た。

D.5.1　LU 分解

最初に二つの言葉を定義する。

行列 $L = (\ell_{ij})$ が**下三角行列** (lower triangluar matrix) であるとは、対角線の上にある成分がすべて 0 である、つまり

$$i > j \quad \Longrightarrow \quad \ell_{ij} = 0$$

が成り立つことと定義する。

同様に、$U = (u_{ij})$ が**上三角行列** (upper triangular matrix) であるとは、対角線の下にある成分がすべて 0 である、つまり

$$i < j \quad \Longrightarrow \quad u_{ij} = 0$$

が成り立つことと定義する。

目で見えるように書くと

$$L = \begin{pmatrix} \ell_{11} & 0 & \cdots & \cdots & 0 \\ \ell_{21} & \ell_{22} & 0 & \cdots & 0 \\ \vdots & & \ddots & & \\ \ell_{n1} & \cdots & \cdots & \ell_{nn} & \end{pmatrix}, \quad U = \begin{pmatrix} u_{11} & & & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & 0 & \ddots & \\ 0 & 0 & & u_{nn} \end{pmatrix}$$

ということである。

LU 分解

行列 A に対して、

$$A = LU$$

を満たす下三角行列 L と上三角行列 U を求めることを (これはいつもできるとは限らない — 後述)、A を **LU 分解**すると呼ぶ。

Gauss の消去法の前進消去段階では、係数行列に行に関する基本変形を施して上三角行列 (それを U とおく) に変形したが、行に関する基本変形は、基本行列を左からかけることで表現できる。Gauss の消去法では、これら基本行列はすべて正則な下三角行列である。そこで、この変形は

$$L_k \cdots L_2 L_1 A = U \quad (L_j \text{ は正則な下三角行列}, U \text{ は上三角行列})$$

とかける。これから

$$A = {L_1}^{-1} {L_2}^{-1} \cdots {L_k}^{-1} U$$

となるが、$L := {L_1}^{-1} {L_2}^{-1} \cdots {L_k}^{-1}$ は実は下三角行列である。これは次の補題から分かる。

補題 D.2 (1) 下三角行列全体は和と積に関して閉じている。

(2) 正則な下三角行列の逆行列は下三角行列である (ゆえに正則な下三角行列全体は乗法について群をなす)。

さて、それでは Gauss の消去法はいつでも出来るかというと、(掃き出し法との類推ですぐ分かるように) 対角成分に 0 が現われたらダメになるわけである。この場合は、行を適当に交換することで消去が出来るようになる。

命題 D.3 A を n 次正則行列とする。

(1) A が LU 分解できるための必要十分条件は、A のすべての主座小行列式が 0 でないことである。特に任意の正値対称行列は LU 分解可能である。

(2) 適当な置換行列 P が存在して、PA は LU 分解できる。すなわち適当な下三角行列 L, 上三角行列 U が存在して

$$PA = LU$$

が成り立つ。

正則行列 A の LU 分解はたとえ存在しても一意ではないが、A を

$A = LDU$ (L は対角成分が 1 の下三角行列、U は対角成分が 1 の上三角行列、D は対角行列)

の形に分解する **LDU 分解**は一意である。

D.5.2 三角行列係数の連立 1 次方程式

三角行列係数の連立 1 次方程式は非常に簡単に解くことができる。これを上三角行列の場合に見てみよう。

$$Ux = b$$

において $U = (u_{ij})$ が上三角行列とする。

$$\begin{aligned} u_{11}x_1 &+ u_{12}x_2 &+ \cdots &+ u_{1n}x_n = b_1 \\ & u_{22}x_2 &+ \cdots &+ u_{2n}x_n = b_2 \\ & & \ddots & \quad \vdots \\ & & & u_{nn}x_n = b_n \end{aligned}$$

は下の方程式から順に

$$\begin{aligned} x_n &= b_n/u_{nn}, \\ x_{n-1} &= (b_{n-1} - u_{n-1,n}x_n)/u_{n-1,n-1}, \\ &\vdots \\ x_i &= \left(b_i - \sum_{j=i+1}^{n} u_{i,j}x_j\right)/u_{ii} \\ &\vdots \\ x_1 &= \left(b_1 - \sum_{j=2}^{n} u_{1,j}x_j\right)/u_{11} \end{aligned}$$

と解ける。要するにこれは Gauss の消去法の後退代入過程と同じである。この計算は数値的安定性に関しても申し分のないものになっている。

行列 A の LU 分解 $A = LU$ があるとき、$Ax = b$ の解は

$$x = U^{-1}(L^{-1}b)$$

と書けるので、三角行列係数の方程式

$$Ly = b,$$
$$Ux = y$$

を順に解けば得られることが分かる。よって、これは非常に簡単に計算できる。

D.5.3 Cholesky 分解

A が正値対称行列である場合、

$$A = LL^T$$

を満たす下三角行列 L が存在する。これを **Cholesky 分解**と呼ぶ。L の対角成分は正であるように取ることができるが、そういうものに限ると分解は一意的である。

L^T は上三角行列であるから、Cholesky 分解は一種の LU 分解である。

Cholesky 分解は、通常の LU 分解の半分程度の計算量で計算できる (具体的な計算法は省略)。

D.5.4 QR 分解

A を実正則行列とするとき、実直交行列 Q と、上三角行列 R で

$$A = QR$$

を満たすものが存在する。特に R の対角成分は正であるように取ることができ、そういうものに限ると分解は一意的である。これを A の **QR 分解**と呼ぶ[19]。

[19] この分解について言及してある線形代数の教科書は結構ある。Schmidt 分解とか Gram-Schmidt 分解と呼ばれることが多いが、数値線形代数の世界ではもっぱら QR 分解と呼ばれる。

$A = (a_1\ a_2 \cdots a_n)$ とするとき、$a_1, \cdots, a_n$ から、Gram-Schmidt の直交化を行って正規直交基底 $q_1, \cdots, q_n$ を作る計算は、A の QR 分解を求めていることになる。

しかし QR 分解を求める場合、この素朴な Gram-Schmidt の直交化法を適用することはない[20]。

LU 分解と同様に QR 分解があれば連立 1 次方程式は簡単に解ける。例えば $Ax = b$ を解きたいときに、$A = QR$ という QR 分解が得られたとしよう。

$$x = R^{-1}(Q^{-1}b) = R^{-1}(Q^T b)$$

であるから、x の計算は簡単である (つまり、実直交行列の逆行列はもとの行列の転置行列に他ならないから、計算するまでもなく分かっているわけ)。

D.5.5 連立 1 次方程式に対する直接法についてのまとめ

色々なことを書いたが、要するに、

与えられた行列 A を逆行列の計算の簡単な行列の積に分解 (factorize) することが要点

D.6 なぜ逆行列を計算してはいけないか

- 逆行列の計算には、Gauss の消去法に基づく LU 分解の計算よりも多くの計算量 (約 3 倍) が必要である。
- 元の係数行列 A が疎である場合、逆行列は (ほとんどすべての場合に) 疎ではないが、LU 分解したときの因子 L, U は疎性を保っている (実例を見せる)。そのため、係数行列が疎である場合には、計算量に大差 (次数 n のベキが異なることもあるので、「桁違い」の差になるのが普通) が生じる。

D.7 固有値問題の解法を理解するための緒命題

与えられた行列をまず直交行列による相似変換で「簡単な形」に変形していく。もしも上三角行列に出来ればめでたしめでたしであるが…

[20] 修正 Gram-Schmidt 法や、基本的な直交変換 (例えば超平面に関する対称移動を表わす Householder 行列や、二次元平面の回転を表わす Givens 行列) を次々にかけていく方法が使われる。

命題 D.4 (1) A を正則行列 P で相似変換した $\widetilde{A}$ (つまり $P^{-1}AP$) は、A と同じ固有値を持つ。

(2) 実直交行列 U の逆行列は U^T であるから、U による相似変換は $\widetilde{A} = U^T AU$ であるが、A が実対称である場合は $\widetilde{A}$ も実対称である。

(3) unitary 行列 U の逆行列は $U^* = \overline{U}^T$ であるから、U による相似変換は $\widetilde{A} = U^* AU$ であるが、A が Hermite 行列である場合は $\widetilde{A}$ も Hermite 行列である。

(4) 実直交行列の積は実直交行列であり、実直交行列による相似変換を有限回繰り返したものは、一つの実直交行列による相似変換に等しい。unitary 行列についても同様のことが成り立つ。

(5) 任意の正方行列 A に対して適当な unitary 行列 U が存在して、

$$U^* AU = R \quad (R \text{ は上三角行列})$$

となる (**Schur 分解**)。上三角行列の固有値はその対角成分に他ならないから、この分解が得られれば A の固有値が求められたことになる。

既に述べたように、固有値問題は代数方程式と等価であるから、有限回の四則演算と冪乗演算で Schur 分解を求めることはできない。しかし、上三角にすることをあきらめ、一歩手前の Hessenberg 行列で我慢することにすると、比較的計算量の少ない計算で済ませることが可能である。その詳細は省略するが、次のような基本的な直交行列による変換を繰り返すことで実現される。

(1) 超平面 $(u, x) = u_1 x_1 + u_2 x_2 + \cdots + u_n x_n = 0$ ($u = (u_i)$ は単位ベクトル) に関する対称移動 (鏡映、鏡像とも言われる) を表わす $H = I - 2u^T u$ (**Householder 行列**と呼ばれる)。

(2) $x_p x_q$ 平面内の角 $-\theta$ の回転を表わす $G = (g_{ij})$.

$$g_{ij} = \begin{cases} \cos\theta & ((i,j) = (p,p), (q,q)) \\ \sin\theta & ((i,j) = (p,q)) \\ -\sin\theta & ((i,j) = (q,p)) \\ 1 & (i = j \notin \{p,q\}) \\ 0 & (\text{それ以外}) \end{cases}$$

(**Givens の回転行列**と呼ばれる。)

E　線形計算ソフトウェアの発展 (駆け足説明)

E.1　線形計算ライブラリィの誕生と発達

1. **サブルーチン**[21](**subroutine**) の誕生、**サブルーチン・ライブラリィ**の誕生
2. (汎用) プログラミング言語[22]の誕生 (**FORTRAN**[23], **LISP** などが最初の例)
3. 固有値計算ライブラリィ **EISPACK**
   (論文誌 “Numerische Mathematic” で発表されたアルゴリズムを元に最初は **ALGOL** で書かれ、後に FORTAN に移植される。主宰者は有名な数値解析学者である Wilkinson である。)
4. 連立 1 次方程式の解法ライブラリィ **LINPACK**
   途中から **BLAS** が生まれ、LINPACK は BLAS の上に構築される。
5. 線形計算ライブラリィ **LAPACK**
   (**メモリー階層**を考慮した BLAS を全面的に採用、EISPACK & LINPACK の現代化)
6. 他のプログラミング言語への移植 — **TNT**[24] (C++) など。

ここで名を紹介した EISPACK, LINPACK, LAPACK, TNT はいずれもソースが公開されているフリーソフトである[25]。

数値計算ライブラリィの採用で実現できること

(1) 高い生産性

(2) 高い信頼性 (バグが少ない、高精度、条件が悪い問題でも崩れないタフさ)

(3) 高い効率性 (速度、メモリー利用効率)

なぜ高い実行効率が得られるか

速度をあげるために考えねばならないこととして、講義では**計算量**を説明した。これらのソフトウェアでは計算量の観点から無駄がないことはもちろんであるが、それ以外にも重要な要素がある。

---

[21] 機械語 (machine language) や、Fortran 言語における、あるまとまった処理をするプログラムの単位を呼ぶ言葉。C 言語における「関数」に相当すると考えて構わない。

[22] 当時は「自動プログラミング言語」と呼ばれたそうである。

[23] FORTRAN は、IBM が線形計画法のプログラムを効率的に作成するために開発した言語であると言われている。

[24] C++ 向けには **LAPACK++** があったが、C++ 言語の ANSI 規格の進展に伴い、新しく設計し直されたのが TNT である。まだ発展途上で、LAPACK の機能のすべては実装されていない。

[25] このあたりに欧米文化の強さが感じられる。ここで紹介したソフトの中には、博士号クラスの研究者が数十人、何年も作業して始めて開発できたものもある。日本でも大学を中心に様々なライブラリィの開発がされたが、全面公開までこぎつけたものは少なく (途中で企業に売ってしまったものもある)、大変もったいない事態になっていると筆者は感じている。こうなってしまった背景には、ソフトウェアの開発を研究業績とは認めない風潮など、二三の理由が考えられる…(脱線)

1. ループのアンロールなどのテクニック
2. メモリー階層を考慮したプログラミング

これらを追求すると、利用するコンピューター・システムに合わせたプログラムのチューニングが必要になり、プログラムの汎用性が低くなる恐れがある。しかし、システムごとにチューニングが必要な部分を小さな部分に凝縮し、他と分離することにより、この問題をある程度解決できる。LAPACK については、BLAS がこのチューニングが必要な部分を担当している。LAPACK に付属する BLAS は “reference (参考) BLAS” と呼ばれ、FORTRAN で書かれているので移植性があるが、高速な計算を望む場合は最適化された BLAS に差し換えることになる。例えば Sun Workstation の場合、Sun Microsystem 製ソフトウェア開発環境 Sun Workshop では、コンパイラーと一緒に BLAS が (と実は LAPACK も) 提供されている)。Windows や Intel CPU 向けの Linux の場合は Intel から BLAS が提供されている (無償)。これらは人手で最適化されたものである (と思われる) が、色々なパラメーターを変化させながら性能を測定することで、最適なパラメーターを実験的に「算出」し、その結果を元に BLAS プログラムを自動生成するソフトウェアもある (例えばフリーソフトの **ATLAS**)。

E.2 MATLAB とその互換システムの登場

MATLAB は EISPACK の開発にも関わった C.Moler が作成したシステムで、彼が創立した MathWorks 社から販売されている。

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, REDUCE のような**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であるが、筆者は最近

案外大したものではないか、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、以下紹介するように「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、工学の世界では浸透している。

MATLAB は現在も改良が続けられていて、行列計算関係では、**疎行列向けの処理**や**反復法**なども採り入れられている。簡単な偏微分方程式のシミュレーションへの応用も十分可能なレベルに成長した。

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「準標準」となっている。MATLAB と似たシステムとして、Octave を紹介する。これは MATLAB との互換性が高く、残念ながら疎行列専用の処理が用意されていないが、入門には十分であるし、用途を選べば実用性も高い。

F　疎行列関係の話

full() 疎行列形式のデータから普通の行列形式のデータを作る。

find() 行列の非零要素を求める。`[i,j,s]=find(A);` としたとき、$A=(a_{ij})$, $\mathtt{i}=(i_1,\ldots,i_N)$, $\mathtt{j}=(j_1,\ldots,j_N)$, $\mathtt{s}=(s_1,\ldots,s_N)$ とおくと、$a_{i_kj_k}=s_k$ $(k=1,\cdots,N)$ であり、これが A の非零要素全体となる。

spy() スパース・パターンの表示 (可視化)

sparse() 疎行列形式のデータを作る。

1. `A` が普通の行列であるとき、`S=sparse(A)` とすると、`S` は数学的には同じ (`fulll(sparse(A))` は `A` と同じということか) 内容の疎行列になる。

```
[i,j,s]=find(A);
[m,n]=size(A);
S=sparse(i,j,s,m,n);
```

2. `sparse(i,j,s,m,n,nzmax)` (`i`, `j`, `s` は次元が同じ (N とする) ベクトルで、`i` と `j` は成分が自然数 ($1\le \mathtt{i}\le m$, $1\le \mathtt{j}\le n$, $\mathtt{nzmax}\ge N$) は $\mathtt{m}\times\mathtt{n}$ 型の行列で、非零成分全体が $a_{i(k)j(k)}=s(k)$ $(k=1,2,\ldots,N)$ であるような疎行列データを作る。`nzmax` を省略した `sparse(i,j,s,m,n)` は、`sparse(i,j,s,m,n,`N`)` と同じである。さらに `m` と `n` を省略した `sparse(i,j,s)` では、`sparse(i,j,s,`$\max\{\mathtt{i}(k)\}$, $\max\{\mathtt{j}(k)\}$`)` となる。

疎行列形式の単位行列を作るには、`speye(n,n)` あるいは `sparse(1:n,1:n,1)` とする。

spalloc() `S=spalloc(m,n,nzmax);` は、`S=sparse([],[],[],m,n,nzmax)` と等価。

spdiag()

```
n=5;
e=ones(n,1);
S=spdiag([-e,2*e,-e],-1:1,n,n);
```

G MATLABで計算したデータを Mathematica に持って行く

こちらの知識が足りないのか、ときどき Mathematica に持って行ってそちらで処理したくなることがある。

ごく普通に ASCII データで出力して、それを読めば良いかと。

MATLAB 側で

```
>> u=1:12;
>> u=reshape(u,3,4)

u =

     1     4     7    10
     2     5     8    11
     3     6     9    12

>> save('~/matlab.dat','u','-ascii')
```

どんな中身？

```
% cat matlab.dat
   1.0000000e+00   4.0000000e+00   7.0000000e+00   1.0000000e+01
   2.0000000e+00   5.0000000e+00   8.0000000e+00   1.1000000e+01
   3.0000000e+00   6.0000000e+00   9.0000000e+00   1.2000000e+01
%
```

Mathematica では

```
In[1] := u=Import["matlab.dat"]
Out[1] = {{1., 4., 7., 10.}, {2., 5., 8., 11.}, {3., 6., 9., 12.}}
```

```
In[53]:= u = Import["matlab.dat"]
Out[53]= {{1., 4., 7., 10.}, {2., 5., 8., 11.}, {3., 6., 9., 12.}}

In[54]:= MatrixForm[u]
Out[54]//MatrixForm=
1.  4.  7.  10.
2.  5.  8.  11.
3.  6.  9.  12.
```

メモリーの中での並び順は「転置されて」しまっているけれど、MATLAB の `u(i,j)` は Mathematica の `u[[i,j]]` と等しいので、混乱することはないだろう。

単に `-ascii` とすると上に見るように 8 桁精度なので、それが不満な場合は

```
>> save('~/matlab.dat','u','-ascii', '-double')
```

とする。こうすれば 16 桁精度で出力する (当然ファイルは大きくなる)。

(追記) 実はこれを書いている今知ったのだが、Mathematica は、MATLAB の MAT ファイルの古い形式 (v4, v5) をサポートしている。MATLAB で

```
>> save('matlab.dat','u', '-mat')

あるいは

>> save('matlab.mat','u')

(拡張子 .mat を選ぶと、自動的に , '-mat' というオプションを指定したのと同じことになる)
```

として出力したファイルを、Mathematica では

```
In[ ] := u=Import["matlab.dat","MAT"]

あるいは

In[ ] := u=Import["matlab.mat"]
```

で読める。(例えば `u` が 2 次元配列であれば、`ListPlot3D[u]` とすればグラフが描ける。)

以下、余談 (と言っても実際に必要が生じて、やる羽目になったので、案外無駄にならないメモかも知れない)。

MATLAB で計算する際は、精度を上げたいので、大規模な計算をするけれど、その計算結果 (2GB 弱だったかな) をそのまま Mathematica に渡すと、Mathematica がパンクする (「読めない」と言うのではなくて、プロセスが終了してしまう!) ということがあった。実際は可視化したいだけで、その目的のためには、計算結果が全部は必要ないという場合は、適当に

間引いて軽いデータにしてから Mathematica に渡してやれば良い。そこで次のようなことをした。

正方形上の関数があり、各辺を 1280 等分して作ったメッシュで関数値を計算した。外部に出力するときに、各辺を 160 等分して作ったメッシュ上の関数値を抜き出したい。

```
N=1280
u=zeros(N+1,N+1);
...
i=1:N/160:N+1;
j=1:N/160:N+1;
smallu=u(i,j);
save('nantoka.dat','smallu','-ascii');
```

`smallu=u(i,j);` という書き方が出来るのが MATLAB の面白いところ。

H misc

そのうちまとめる (まとめたい) けれど、とりあえずここに突っ込んでおこう。

H.1 2 変数関数のグラフの鳥瞰図 mesh(), meshc(), surf(), surfc()

2 変数関数のグラフの鳥瞰図を線画で描くには、`mesh()` という関数を使う。`meshc()` は同時に等高線も描く。

シェーディングされた面の集まりとして描くには、`surf()`, `surfc()` を使う。

- `mesh(X,Y,Z)` のように使う。
- `Z` は 2 次元配列であるが、関数 f のグラフを描く場合、`Z(j+1,i+1)` に $f(x_i, y_j)$ を収めるというのが味噌。
- `X` と `Y` は 1 次元配列 (それぞれ `X(i+1)`=x_i, `Y(j+1)` = y_j) の場合も、2 次元配列 (それぞれ `X(j+1,i+1)` = x_i, `Y(j+1,i+1)` = y_j) の場合もある。後者は `meshgrid()` で作るのが簡単。
- `meshgrid()` の使い方。

  $[a,b] \times [c,d]$ の各辺を m, n 等分するグリッドを作る

  ```
  [X,Y]=meshgrid(a:(b-a)/m:b,c:(d-c)/n:d);
  ```

  `size(meshgrid(a:(b-a)/m:b,c:(d-c)/n:d))` とすると、`[n+1 m+1]` という結果が返ってくる。$a = 0$, $b = 5$, $c = 0$, $d = 3$, $m = 5$, $n = 6$ の場合、7 行 6 列の行列が返る。

```
>> a=0;b=5;c=0;d=3;m=5;n=6;
>> size(meshgrid(a:(b-a)/m:b,c:(d-c)/n:d))
ans =
      7   6
>> [X,Y]=meshgrid(a:(b-a)/m:b,c:(d-c)/n:d)
X =

     0     1     2     3     4     5
     0     1     2     3     4     5
     0     1     2     3     4     5
     0     1     2     3     4     5
     0     1     2     3     4     5
     0     1     2     3     4     5
     0     1     2     3     4     5


Y =

         0         0         0         0         0         0
    0.5000    0.5000    0.5000    0.5000    0.5000    0.5000
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
    1.5000    1.5000    1.5000    1.5000    1.5000    1.5000
    2.0000    2.0000    2.0000    2.0000    2.0000    2.0000
    2.5000    2.5000    2.5000    2.5000    2.5000    2.5000
    3.0000    3.0000    3.0000    3.0000    3.0000    3.0000
```

素朴な発想？

```
% [-2,2] × [-1,1] で x^2-y^2 のグラフを描く
nx=50;
ny=40;
hx=4/nx;
hy=2/ny;
X=-2:hx:2;
Y=-1:hy:1;
Z=zeros(ny+1,nx+1);
for i=0:nx
    x=-2+i*hx;
    for j=0:ny
        y=-1+j*hy;
        Z(j+1,i+1)=x^2-y^2;
    end
end
clf
mesh(X,Y,Z)
```

MATLAB らしい書き方？

```
nx=50;
ny=40;
hx=4/nx;
hy=2/ny;
[X,Y]=meshgrid(-2:hx:2,-1:hy:1);
Z=X.^2-Y.^2;
clf
mesh(X,Y,Z)
```

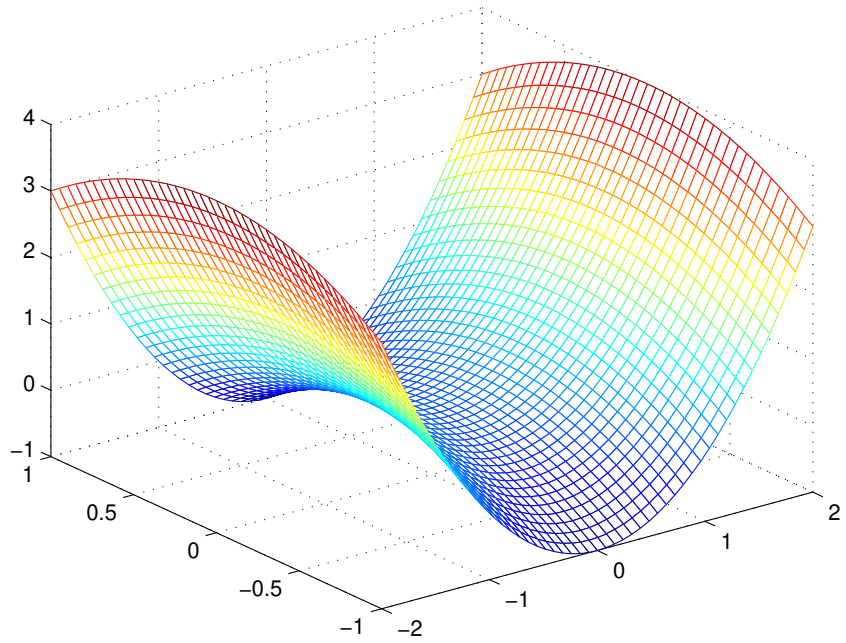


図 6: $z = f(x, y) = x^2 - y^2$ のグラフの鳥瞰図

`mesh()`, `meshc()`, `surf()`, `surfc()` の違いを見ておこう。

H.2 等高線を描く

グラフの鳥瞰図に添えるように `meshc()` という関数を使えば良い場合も多いだろうが、真上からのぞいた「正確な地図」が見たい場合は、`contour()` を使って描くと良い。

- とにかくざっと描ければ良いなら `contour(Z)` とする。
- 等高線の本数 (正確にはレベルの個数) `n` を指定するには、`contour(Z,n)` とする。
- レベルの数値を具体的に指示するには、レベルを収めた配列を `Ls` として、`contour(Z,Ls)` とする。

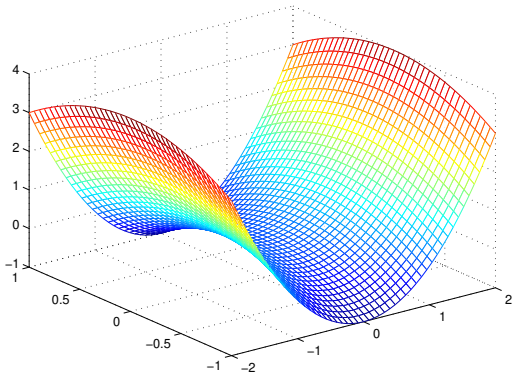


図 7: グラフの鳥瞰図 (mesh())

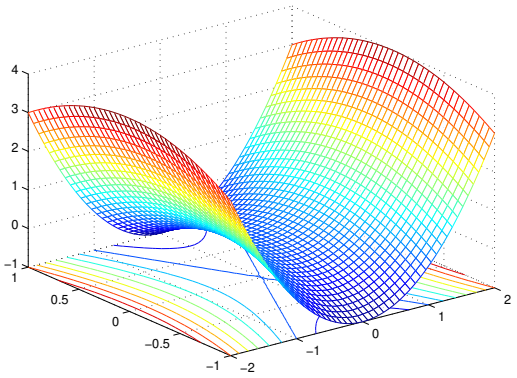


図 8: グラフの鳥瞰図 (meshc())

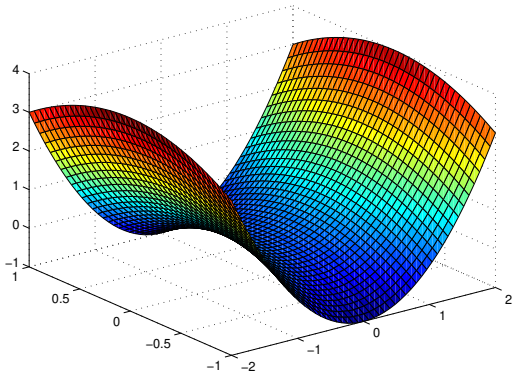


図 9: グラフの鳥瞰図 (surf())

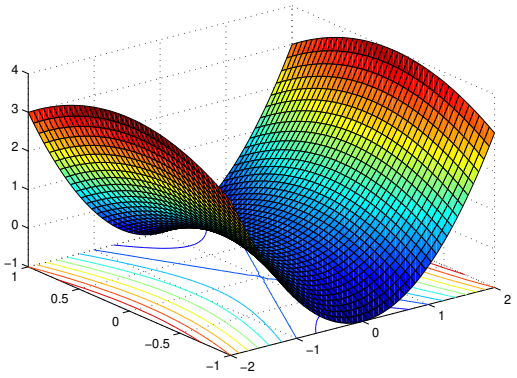


図 10: グラフの鳥瞰図 (surfc())

- 例えば a, b, c という 3 つのレベルを指定するには contour(Z, [a b c]) とする。L という一つのレベルを指定するには、二つ並べたベクトルを指定して contour(Z, [L L]) とする (contour(Z,[L]) とすると L が本数と解釈されてしまう)。

H.3 複数のグラフィックスを並べて描く

subplot(m,n,p) を使う。

$z = f(x, y) = x^2 + y^2$ のグラフの鳥瞰図と等高線

```
nx=50;
ny=40;
hx=2/nx;
hy=2/ny;
[X,Y]=meshgrid(-1:hx:1,-1:hy:1);
Z=X.^2-Y.^2;
subplot(1,2,1)
surfc(X,Y,Z)
subplot(1,2,2)
contour(X,Y,Z)
axis square
```

H.4 数値実験結果を表示するときの注意

(工事中)

ずっと以前、学生達 (含む自分?) が良くやっていた失敗は、とにかくグラフを印刷して (まだ何でもすぐ印刷する時代でした)、後になって何を印刷したのか分からなくなる。この二つの図、どちらがどっちのパラメーターで描いたのだっけ?それを注意したら、シャープペンでパラメーターを書くのだけど、そもそもそのメモが正しい保証はどこにあるでしょう。案の上、時々間違える。

というわけで、そのグラフが何であるかの情報 (パラメーターなど含む) は、そのグラフィックスに含めるように努めるべきです。それもなるべく最初からそういうふうにプログラムを書いておくべきです (プログラムは子沢山になりがちなので。

H.5 等角写像 (写像関数) を可視化する (等高線を描く機能の応用)

複素平面内の任意の単連結領域 Ω ($\neq \mathbf{C}$) に対して、Ω から単位円盤 $D = \{z \in \mathbf{C}; |z| < 1\}$ の上への等角写像 $f\colon \Omega \to D$ が存在する。それを Ω の写像関数と呼ぶ。

適当な追加条件を課すと写像関数は一意には定まる。例えば $z_0 \in \Omega$ を任意に一つ選ぶと、

$$f(z_0) = 0, \quad f'(z_0) > 0$$

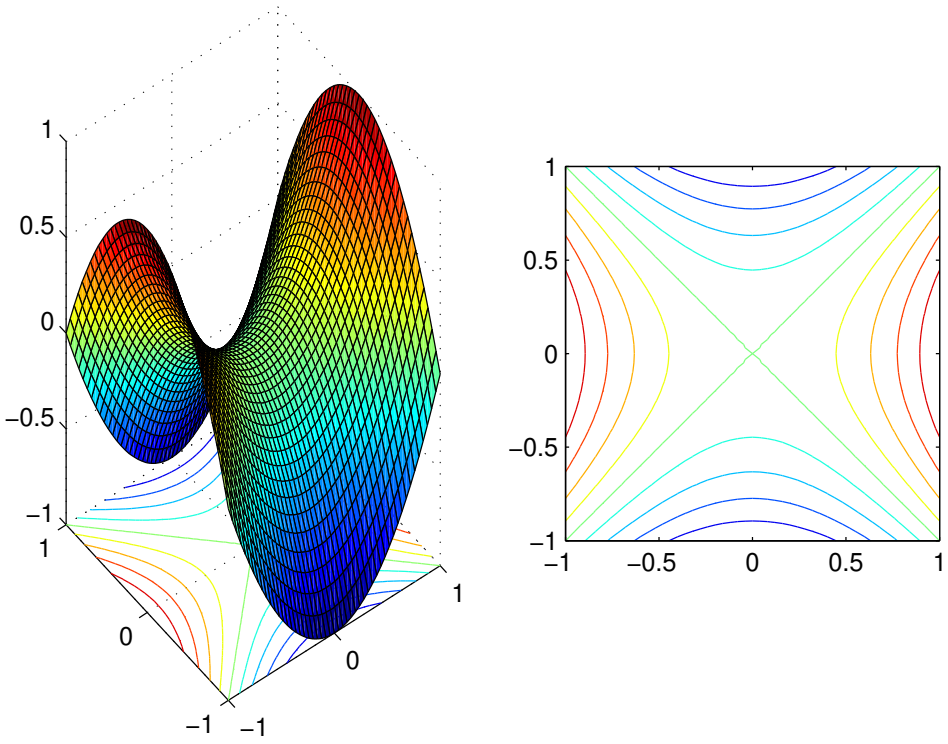


図 11: $z = f(x, y) = x^2 + y^2$ のグラフの鳥瞰図と等高線

を満たす写像関数 f は一意的に存在することが知られている。

$\Omega = D$ のとき、関数論で学ぶように

$$f(z) = \frac{z - z_0}{1 - \overline{z_0} z}.$$

これを可視化してみよう。

一般に写像 $f\colon \Omega \to D$ を可視化するために、Ω か D の中に何かを描いて、もう一方にはそれを写像したものを描く、という方法がある。今の場合、D は単位円盤であるから、原点中心の同心円群 $\{w \in \mathbf{C}; |w| = r_i\}$ と、原点を通る放射線群 $\{w \in \mathbf{C}; \arg w = \theta_j\}$ を採用することが考えられる。ただし $\{r_i\}$, $\{\theta_j\}$ は $[0,1]$, $[-\pi, \pi]$ の分割点である:

$$0 = r_0 < r_1 < \cdots < r_m = 1, \quad -\pi = \theta_0 < \theta_1 < \cdots < \theta_n = \pi.$$

$w = f(z)$ の実部と虚部の等高線を描く

```
[x,y]=meshgrid(-2:0.01:2,-2:0.01:2);
z=x+1i*y;

z0=0.5;
w=(z-z0)./(1-conj(z0)*z);

clf
hold on
m=20;
n=10;
levelangle=linspace(-pi,pi,m+1);
levelabs=linspace(0,1,n+1);

contour(x,y,angle(w),levelangle)
contour(x,y,abs(w),levelabs)
axis square
```

これで一応、写像の “状況” は分かるわけだが、Ω からはみ出ているのが嫌味である。

うまい解決策が分からないので、次のようにして逃げている。指定した点が指定した多角形の内部に含まれているかどうかを判定する `in=inpolygon(x,y,xv,yv)` という関数を利用する。

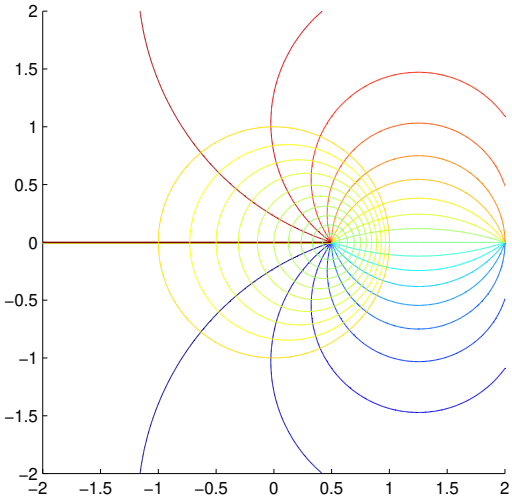


図 12: $w = f(z) = \dfrac{z - z_0}{1 - \overline{z_0} z}$, $z_0 = 1/2$ (はみ出る)

はみ出ないように

```
% メッシュを準備
[x,y]=meshgrid(-1.2:0.01:1.2,-1.2:0.01:1.2);
z=x+1i*y;

% 写像関数
z0=0.5;
w=(z-z0)./(1-conj(z0)*z);

% 等高線のレベル
m=20;
n=10;
levelangle=linspace(-pi,pi,m+1);
levelabs=linspace(0,1,n+1);

% 領域の内部にあるかどうか判定
t=linspace(0,1,100+1);
bx=cos(2*pi*t);
by=sin(2*pi*t);
indisk=inpolygon(x,y,bx,by);
outdisk=(1-indisk)*10;

clf;
subplot(1,2,1)
hold on
contour(x,y,angle(w)+outdisk,levelangle);
contour(x,y,abs(w)+outdisk,levelabs)
axis square

subplot(1,2,2)
hold on
contour(x,y,angle(z)+outdisk,levelangle);
contour(x,y,abs(z)+outdisk,levelabs)
axis square
```

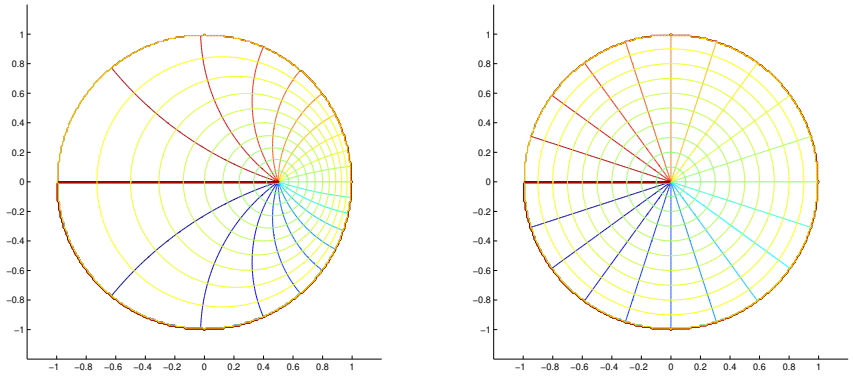


図 13: $w = f(z) = \dfrac{z - z_0}{1 - \overline{z_0} z}$, $z_0 = 1/2$ (はみ出ない)

H.6

```
a=(1:10)'*(1:10)
i=2:2:10
j=1:2:9
a(i,j)
```

```
 i=1:N/160:(N+1)
 aa=a(i,i);
```

```
a=zeros(5,3);
for i=1:5
  for j=1:3
    a(i,j)=10*i+j;
  end
end
a
a(:)
```

MATLAB の配列は a(1,1), a(2,1), .. と行の番号が早く変化する。FORTRAN 流だ。

H.7 nargin で引数を数える

C のプログラムの argc のように使える変数 (?) nargin がある。

```
function myfunc(a1, a2)
if nargin == 0
  display('usage: myfunc(a1,a2)')
  display(' a1: なんとか')
  display(' a2: かんとか')
  return
end
if nargin == 1
  a2=1
end
```

みたいなことが出来る。

H.8 自作ライブラリィの扱い

基本的なことだけど。
適当なディレクトリィを用意して、そこに必要なファイルを置き、addpath() して使う。

```
% mkdir ~/Documents/MATLAB/なんとか
```

~/Documents/MATLAB/startup.m

```
addpath('~/Documents/MATLAB/なんとか')
```

なんとか/かんとか.m を編集

```
>> edit なんとか/かんとか
```

I 疎行列用の命令のまとめ

- それほど大きくなければ、作ってから spA=sparse(A); のようにして、疎行列に直せば良い。
- spdiag(a)

J MATLAB と Octave の違い

昔、Octave についてきた文書に、MATLAB の文法は "brain dameged" なんてことが書いてあって、にやっとさせられたけれど、使う身になってみると、Octave で使えたものが使えないのは困りもの。知識の矯正をしないといけない。

- MATLAB で行末までの注釈は % を使う (シェルスクリプト風の # は使えない)。

- MATLAB で「等しくない」は `~=` を使う (C 言語風の `!=` は使えない)。
- MATLAB で `if` を閉じるのは `end` (`endif` は使えない)。
- Octave にある `endfunction` は MATLAB では使えない。