2012 年度情報処理 II　第 13 回

MATLAB 体験 (2) 行列の解析的性質

桂田 祐史 (かつらだ まさし)

2012 年 7 月 11 日

この授業用の WWW ページは http://www.math.meiji.ac.jp/~mk/syori2-2012/

1　連絡事項

- 前回の内容の後始末 (主に数学的解説) を付録 A に書いておきました。
- 来週の授業をどうするかについては、今日の授業の最後にアナウンスします。
  無事にシラバスの内容を全部終えたので、18 日 (水曜)3 限はレポートの質問相談日にします。

2　行列の解析的な性質

今回は MATLAB を使って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 [1] でも読める。

2.1　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[1]

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

の極限はどうなるでしょうか？既に学んだかもしれませんが (行列の Jordan 標準形の簡単な応用です)、ここでは実験で試してみましょう。

[1] 点の列は点列と言うから、行列の列は行列列？そういう本はないみたいです。

注意: 以下の実験では、乱数行列 rand() を使っているので、小さい確率で、こちらが想定した状況にならない可能性があります。桂田のパソコンのスクリーンのようにならない場合は、a=rand(5,5) からやり直して下さい。

```
>> a=rand(5,5)
>> a*a
>> a^2
>> a^100
>> a^1000
>> a^10000
```

MATLAB では、行列 a の n 乗は a^n で計算できることが分かりますが、この例では (大抵の行列 a に対して) 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してしまいます。

種明かしをすると、行列の固有値の絶対値 (その最大値をスペクトル半径と呼ぶ) を調べると事情が見えて来ます。

```
(上に引き続き)
>> eig(a)
>> r=max(abs(eig(a)))
    (固有値 eigs() の、絶対値 abs() の、最大値 max())
>> a=a/r
>> max(abs(eig(a)))
    1 になるはずです…
>> a^10
>> a^100
>> a^1000
>> a^10000
    あれ? と思うはずです。
>> for i=1:10
  a^i
  end
```

MATLAB では max(abs(eig(a))) で a のスペクトル半径が計算できます。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ます。a^100, a^1000 ともにオーバーフローしていないはずですが、それだけでなく何かが起こっていることに気がつくでしょうか? (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ましょう。

(上に引き続き)

```
>> a=0.9*a
>> a^10
>> a^100
>> a^1000
>> a^10000
```

0 になるはずです。

ここで等比数列の極限 $\lim_{n\to\infty} r^n$ がどうなるか思い出して、比較してみて下さい。

2.2 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 (ノイマン) 級数**と呼ばれる。

以下の実験は上の続きで行うことを想定している。MATLAB 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

```
a=rand(5,5)
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 MATLAB プログラム `neuamnn.m` を用意した。

neumann.m

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

```
>> edit neumann
```

とすると編集ウィンドウが現れるので、上の neumann.m の内容をコピー＆ペーストして保存して下さい (ドキュメント ライブラリの MATLAB フォルダに保存されるはずです)。その後 neumann() が使えるようになるはずです。

```
>> c=eye(5,5)-a
>> b=neumann(a,100)
>> b*c
>> b=neumann(a,1000)   ← もう少し精度を上げてみる
>> b*c
>> format long         ← お望みなら表示桁数を上げて
>> b*c
```

種明かしは次のページに。

2.2.1 種明かし

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ります。種明かしをすると、一般に

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

が成り立つからです。これは等比級数の和の公式

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

の行列への一般化です。

2.3 絶対値最大の固有値を求める — 冪乗法

(ここは時間の関係で入れないでしょう。)

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 $u_1, \cdots, u_n$ とします。さらに $|\lambda_1|$ は他の固有値の絶対値よりも大きいと仮定します:

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

任意のベクトル $x \in \mathbf{C}^n$ は

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

と展開できますが、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ります (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と期待できます。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す。

以上のことを MATLAB 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 $\dfrac{A^k x}{\|A^k x\|}$ を作ればよいでしょう。

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

```
>> x=ones(5,1)
>> for i=1:100
> y=a*x
> x=y/norm(y)
> end
>> a*x ./ x      ← a*x の各成分を対応する x の成分で割ってみる
>> 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ますが、固有多項式を数値計算で解くのは多くの場合難しいので、行列の問題のまま各種の反復法を用いるのが普通です。上で見た方法は『冪乗法』と呼ばれ、多くの固有値計算方法の基礎となっています。

3　レポート課題 13

締め切りは 7 月 20 日 (金) 18:00。Oh-o! Meiji レポートシステムを使って、kadai13.pdf を提出する。

`neumann.m` を参考に、行列の指数関数

$$\exp A = \sum_{n=0}^{\infty} \frac{1}{n!} A^n$$

を計算する関数 `exponential()` を作って下さい。例えば

$$A = tJ, \quad J := \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

のとき、$\exp A$ が何になるか調べて (対馬先生のテキストの付録 `http://www.math.meiji.ac.jp/~tsushima/senkei.html` の第 9 章の章末問題 2 参照)、それを MATLAB の計算結果と比較しなさい (t は適当に 1 つ 2 つ選べば良い。$t = 1$ とか $t = \pi/4$ とか。)。

A　前回の後始末

前回、前半の Mathematica の課題の説明と MATLAB の導入に時間を取られて、説明がはしょりすぎていたので、説明の補足を書いておきます。

A.1　Gauss の消去法

(他の授業で習ったのではないか？と思うけれど、念のため。)

連立 1 次方程式の解法として、線形代数の教科書には **クラーメル (Cramer) の公式**や**掃き出し法** (**Jordan (ヨルダン) の消去法**ともいう) が説明されていることが多いが、**Gauss の消去法**は、掃き出し法を改良したものである[2]。

例として次の方程式を取りあげて説明しよう。

$$\left\{ \begin{array}{rcrcrl} 2x_1 & + & 3x_2 & - & x_3 & = 5 \\ 4x_1 & + & 4x_2 & - & 3x_3 & = 3 \\ -2x_1 & + & 3x_2 & - & x_3 & = 1. \end{array} \right. \tag{1}$$

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

[2]見方によっては、ガウスの消去法は中学校で習う加減法 (初めて習う連立 1 次方程式の解法！) そのものであり、大学の線形代数で習う解法は、実用性では退化していると言えなくもない (？)。

1. ある行に 0 でない定数をかける。

2. 2 つの行を入れ換える。

3. ある行に別の行の定数倍を加える。

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。

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

$$\to \begin{pmatrix} 1 & \frac{3}{2} & -\frac{1}{2} & \frac{5}{2} \\ 0 & 1 & \frac{1}{2} & \frac{7}{2} \\ 0 & 6 & -2 & 6 \end{pmatrix} \to \begin{pmatrix} 1 & 0 & -\frac{5}{4} & -\frac{11}{4} \\ 0 & 1 & \frac{1}{2} & \frac{7}{2} \\ 0 & 0 & -\frac{5}{2} & -\frac{15}{2} \end{pmatrix} \to \begin{pmatrix} 1 & 0 & -\frac{5}{4} & -\frac{11}{4} \\ 0 & 1 & \frac{1}{2} & \frac{7}{2} \\ 0 & 0 & 1 & 3 \end{pmatrix} \to$$

$$\to \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{pmatrix}, \quad \text{ゆえに} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}.$$

Gauss 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

$$\begin{aligned} 2x_1 + 3x_2 - x_3 &= 5, \\ -2x_2 - x_3 &= -7, \\ -5x_3 &= -15 \end{aligned}$$

ということを表しているので、下の方から順に

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

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。

以下簡単に 3 つの方法の比較をしよう。

クラーメルの公式を適用するには $n+1$ 個の行列式を求める必要があるため、計算の手間がかかる (大きな計算量が必要になる、という)。実際、行列式を一つ計算するための手間は、連立方程式を一つ解くための手間と本質的に同等であることが分かっているので、クラーメルの公式を使うことに固執すると、本来必要な計算量の n 倍程度の計算をする羽目になり、n が大きい場合には、大変な損をすることになる。そのため、実際の数値計算では、ごく特殊な

例を除いて、クラーメルの公式が利用されることはない。クラーメルの公式は、理論的な問題を扱う場合に、真価が発揮されるものである。

このクラーメルの方法に比べれば、掃き出し法は、かなりの実用性を持っているが、多くの微分方程式の数値計算に現れる疎行列の場合には、Gauss の消去法の方が断然有利である。それは、Gauss の消去法を採用すると、掃き出しの途中に現れる行列が疎行列のままであることから、計算量が少なくてすむためである。

A.2 LU 分解とは

連立 1 次方程式を解くのに、逆行列を一度計算してしまえば、後は楽じゃない？

という考えに「それは勘違いです。LU 分解を使うべきです。」ということを、以下の数小節を費やして説明する。

対角線より上にあるすべての成分が 0 であるような行列を**下三角行列**と呼ぶ。すなわち $L = (\ell_{ij})$ が下三角行列であるとは、

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

が成り立つことをいう。

同様に対角線より下にあるすべての成分が 0 である行列を**上三角行列**と呼ぶ。

正則な下三角行列の全体は乗法に関して群をなす。上三角行列についても同様である。

行列 A に対して、

$$A = LU, \quad L \text{ は下三角行列}, \quad U \text{ は上三角行列}$$

をみたす L, U が存在するとき、L と U の組を A の **LU 分解**とよび、L と U の組を求めることを A を **LU 分解する**、という。

- 与えられた正則行列 A が LU 分解を持つためには、A のすべての首座小行列式 $\det A_k$ が 0 でないことが必要十分である。
- 任意の正則行列 A に対して、適当に行の交換を行った行列 PA は、LU 分解を持つ (P はいわゆる置換行列で、左からかけることで行の交換がおこる)。

A.3 Gauss の消去法は LU 分解をしていることに他ならない

実は、Gauss の消去法は LU 分解を計算していることに相当する。このことを理解すれば、LU 分解の計算法を知っていることになる。

ここでは例で納得してもらう。

Gauss の消去法の例としてあげた

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

を考えてみよう。もちろん係数行列の変形だけ取り出すと

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

となる。

$$\begin{aligned} q_{21} &= \frac{4}{2} = 2, \quad q_{31} = \frac{-2}{2} = -1, \\ q_{32} &= \frac{6}{-2} = -3 \end{aligned}$$

を並べて

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -3 & 1 \end{pmatrix}$$

を作り、Gauss の消去法で最後に残った係数行列の部分を

$$U = \begin{pmatrix} 2 & 3 & -1 \\ 0 & -2 & -1 \\ 0 & 0 & -5 \end{pmatrix}$$

とおくと、

$$LU = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -3 & 1 \end{pmatrix} \begin{pmatrix} 2 & 3 & -1 \\ 0 & -2 & -1 \\ 0 & 0 & -5 \end{pmatrix} = \begin{pmatrix} 2 & 3 & -1 \\ 4 & 4 & -3 \\ -2 & 3 & -1 \end{pmatrix} = A.$$

実はこのことは一般的に成り立つ (証明は例えば『「発展形の数値解析」の続き』[3])。

A.4 LU 分解があれば連立 1 次方程式はすぐ解ける (逆行列の代わりになる)

n 次正則行列 A が $A = LU$ と LU 分解されているとき、連立 1 次方程式

$$Ax = b$$

は少ない計算量で解くことが出来る。以下、このことを説明する。

$$Ax = b \qquad (\Leftrightarrow \quad LUx = b)$$

は

$$Ly = b, \quad Ux = y$$

という二つの問題に分解される。

[3] http://www.math.meiji.ac.jp/~mk/labo/text/heat-fdm-0-add.pdf

まず $Ly = b$ は

$$\begin{array}{lllll} \ell_{11}y_1 & & & & = b_1 \\ \ell_{21}y_1 & +\ell_{22}y_2 & & & = b_2 \\ \ell_{31}y_1 & +\ell_{32}y_2 & +\ell_{33}y_3 & & = b_3 \\ \vdots & & & \ddots & \vdots \\ \ell_{n1}y_1 & +\ell_{n2}y_2 & +\ell_{n3}y_3 & \cdots \quad +\ell_{nn}y_n & = b_n \end{array}$$

ということであり、これは上から順に

$$\begin{aligned} y_1 &= b_1/\ell_{11}, \\ y_2 &= \left(b_2 - \ell_{21}y_1\right)/\ell_{22}, \\ y_3 &= \left(b_3 - \ell_{31}y_1 - \ell_{32}y_2\right)/\ell_{33}, \\ &\cdots \\ y_i &= \left(b_i - \sum_{j=1}^{i-1} \ell_{ij}y_j\right)/\ell_{ii}, \\ &\cdots \\ y_n &= \left(b_n - \sum_{j=1}^{n-1} \ell_{nj}y_j\right)/\ell_{nn} \end{aligned}$$

と解くことが出来る (A が正則であると仮定したことから、L も正則で、すべての i について $\ell_{ii} \neq 0$ が成り立つことに注意)。

これを計算するには、$1 + 2 + \cdots + n = n(n+1)/2$ 回の乗除算で十分である[4]。

同様に $Ux = y$ は

$$\begin{array}{lllll} u_{11}x_1 & \cdots & +u_{1,n-2}x_{n-2} & +u_{1,n-1}x_{n-1} & +u_{1n}x_n & = y_1, \\ & \ddots & & & \vdots & \vdots \\ & & u_{n-2,n-2}x_{n-2} & +u_{n-2,n-1}x_{n-1} & +u_{n-2,n}x_n & = y_{n-2}, \\ & & & u_{n-1,n-1}x_{n-1} & +u_{n-1,n}x_n & = y_{n-1}, \\ & & & & u_{n,n}x_n & = y_n \end{array}$$

ということである。やはり A が正則という仮定から、$u_{ii} \neq 0$ $(i = 1, \ldots, n)$ が導かれること

[4]計算にどれくらい手間がかかるかを計るために、基本的な演算の回数を数えるという方法がある。しばしば、四則演算のそれぞれについて数える代りに、乗除算の回数だけを数えて目安にする、という手段が採用される (4次元ベクトルよりは1つの数値が分かりやすい)。もう少し詳しいことが知りたければ、例えば桂田 [?] を見よ。

に注意すると、この連立方程式は、下から順に

$$
\begin{aligned}
x_n &= y_n/u_{nn}, \\
x_{n-1} &= \left(y_{n-1} - u_{n-1,n}x_n\right)/u_{n-1,n-1}, \\
x_{n-2} &= \left(y_{n-2} - u_{n-2,n-1}x_{n-1} - u_{n-2,n}x_n\right)/u_{n-2,n-2}, \\
&\vdots \qquad\qquad \vdots \\
x_i &= \left(y_i - \sum_{j=i+1}^{n} u_{ij}x_j\right)/u_{ii}, \\
&\vdots \qquad\qquad \vdots \\
x_1 &= \left(y_1 - \sum_{j=2}^{n} u_{1j}x_j\right)/u_{11}
\end{aligned}
$$

と解くことができる。

これも計算するには、$1+2+\cdots+n = n(n+1)/2$ 回の乗除算で十分である。

まとめると、$n(n+1)/2 + n(n+1)/2 = n(n+1)$ 回の乗除算で連立 1 次方程式が解けることになる[5]。これは連立 1 次方程式を「普通に」解く場合に、n^3 に比例する回数の乗除算が必要なことと比較して、(n が大きな場合は) かなり少ない回数となる。

普通に解く $O(n^3)$　v.s.　LU 分解 (が与えられていてそれ) を使った場合 n^2 程度

A.5　不思議な常識: 微分方程式を解くときに逆行列は使わない

前小節までに、LU 分解があれば、逆行列と同じ計算量で連立 1 次方程式が解けることを説明したが、実は微分方程式を解く場合には、現れる大抵の行列が疎行列であることから、LU 分解を使うのが断然有利である。

前回

```
>> n=10
>> a=2*eye(n-1,n-1)-diag(ones(n-2,1),1)-diag(ones(n-2,1),-1)
```

のような行列を例に出しましたが、毎回こう入力するのが面倒なので、関数を作りましょう、

```
>> edit Lap1d
```

とすると、編集ウィンドウが現れます。そこに次のように入力し、ファイルを保存します。

Lap1d.m

```
function a=Lap1d(n)
  a=2*eye(n-1,n-1)-diag(ones(n-2,1),1)-diag(ones(n-2,1),-1);
```

[5]細かい話をすると、L の対角成分が 1 である場合は、割り算の回数が n 回減って、n^2 回の乗除算で解けることになる。逆行列を知っている場合、連立 1 次方程式は n^2 回の掛け算で解けるが、それとまったく互角であることが分かる。

```
>> a=Lap1d(10)
>> a=Lap1d(20)
```

前回説明した \ 演算子で連立 1 次方程式を解いてみます。

\ 演算子を使って解く

```
>> n=1000
>> tic; Lap1d(n) \ rand(n-1,1); toc
経過時間は 0.069753 秒です。

>> n=2000
>> tic; Lap1d(n) \ rand(n-1,1); toc
経過時間は 0.381871 秒です。

>> n=4000
>> tic; Lap1d(n) \ rand(n-1,1); toc
経過時間は 2.136665 秒です。
```

未知数の個数 n はこれくらいが無難です。n がある程度大きくなると、メモリーが不足して、

```
??? メモリが足りません。オプションについては、HELP MEMORY と入力してください。

エラー ==> Lap1d at 2
  a=2*eye(n-1,n-1)-diag(ones(n-2,1),1)-diag(ones(n-2,1),-1);
```

のようにギブアップされたり、スワップを始めて、計算自体は行うものの、非常に遅くなります。Mathematica と違って、MATLAB は簡単に停止できません (Control-C を打ってしばらく待つか、MATLAB を強制的に終了することになります)。

逆行列を用いて連立 1 次方程式を解いてみましょう。

逆行列 inv() を使って解く

```
>> n=1000
>> tic; inv(Lap1d(n))*rand(n-1,1); toc
経過時間は 0.300695 秒です。

>> n=2000
>> tic; inv(Lap1d(n))*rand(n-1,1); toc
経過時間は 2.283513 秒です。

>> n=4000
>> tic; inv(Lap1d(n))*rand(n-1,1); toc
経過時間は 16.784775 秒です。
```

最後の $n = 4000$ の場合、8 倍程度余計に時間がかかっていることに注目。

一方、LU 分解を用いて解いて見ると、

LU 分解 lu() を使って解く

```
>> n=1000
>> tic; [L u p]=lu(lap1d(n)); u\(L\(p*rand(n-1,1))); toc
経過時間は 0.165507 秒です。
>> clear
>> n=2000
>> tic; [L u p]=lu(lap1d(n)); u\(L\(p*rand(n-1,1))); toc
経過時間は 0.483060 秒です。
>> clear
>> n=4000
>> tic; [L u p]=lu(lap1d(n)); u\(L\(p*rand(n-1,1))); toc
経過時間は 3.649599 秒です。
```

逆行列を使って解くより速く、\ 演算子を使って解くのと同じオーダーの時間で解けています。

参考文献

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