4. 連立方程式と行列計算

連立方程式は中学ぐらいから勉強しているので特に前置きはいらないと思うが、数値計算では連続量の代わりに離散量を扱うので、ほとんどの場合、連立方程式=行列が計算中に現れる。しかも、そのサイズは数百、数万という巨大な行列となるので、手計算とは違う手法が必要になる。ここで少しその辺りのところを考えてみたい。

4.1. 逆行列で解く

連立方程式 $\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$ は行列なので、行列 **A** に対する逆行列を求め、左から逆行列をかけてやれば $\boldsymbol{x}=\mathbf{A}^{-1}\boldsymbol{b}$ となって、連立方程式の解 $\boldsymbol{x}$ が求められる。逆行列をどのようなアルゴリズムで求めるかはさておき、実際計算してみよう。Octave では、以前紹介したように、**inv(A)** あるいは、**A^-1** とすれば逆行列が求められる。

(例題)　$\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$ を求めよ。ただし、

$$\mathbf{A}=\begin{pmatrix}1&2&0\\1&-1&1\\2&0&1\end{pmatrix}\quad \boldsymbol{b}=\begin{pmatrix}1\\-1\\-2\end{pmatrix}$$

```
> A=[1 2 0;1 -1 1; 2 0 1];
> b=[1; -1; -2];
> inv(A)*b
ans =
-3
 2
 4
```

簡単すぎる…。

4.2. ガウス=ジョルダン法

上の例のような 3×3 程度の行列計算をするなら、どのように計算してもたいした差は出てこないが、行列のサイズが、数百、数万になった場合は、計算時間が少ないアルゴリズム(時間的リソース)かどうか、メモリをなるべく消費しないアルゴリズムか(物理的リソース)どうかという点が問題になる。その観点からすると、上の **4.1.**で示した、「逆行列を求め、左からかけて連立方程式を解く」というアルゴリズムは、実は、数値計算では良くない方法とされている。それは「連立方程式を直接解く」アルゴリズムが別にあって、そちらの方が逆行列を求めるアルゴリズムより計算量が少なくて済むので、わざわざ逆行列を求めるのが無駄だからだ。連立方程式を解くアルゴリズムもいくつかあるが、まずは、ガウスジョルダン法を紹介したい。たいそうな名前が付いているが中身はたいしたことはない。**4.1** の例を使ってどういうものか、手順を追って見てみよう。

① まず行列 **A** の(1,1)要素が 1 になるように、1 行目全体を(1,1)要素で割る。(行全体を割るので、恒等変換) 今の例はすでに 1 になっているので、特になにもしなくていい。

② (1,1)以外の 1 列目の要素をすべて 0 にするため、行同士の引き算を行う。

$$\begin{pmatrix}1&2&0\\1&-1&1\\2&0&1\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}=\begin{pmatrix}1\\-1\\-2\end{pmatrix}$$

二行目から一行目を引く

$$\begin{pmatrix}1&2&0\\0&-3&1\\2&0&1\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}=\begin{pmatrix}1\\-2\\-2\end{pmatrix}$$

三行目から一行目の 2 倍を引く

$$\begin{pmatrix}1&2&0\\0&-3&1\\0&-4&1\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}=\begin{pmatrix}1\\-2\\-4\end{pmatrix}$$

③ (2,2)要素を 1 にするように、二行目全体を(2,2)要素で割る。今の例では-3。

$$\begin{pmatrix}1&2&0\\0&1&-1/3\\0&-4&1\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}=\begin{pmatrix}1\\2/3\\-4\end{pmatrix}$$

④ (2,2)要素以外の二列目の要素がすべて 0 になるように、行同士の引き算をする。

$$\begin{pmatrix}1&0&2/3\\0&1&-1/3\\0&-4&1\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}=\begin{pmatrix}-1/3\\2/3\\-4\end{pmatrix}$$

$$\begin{pmatrix}1&0&2/3\\0&1&-1/3\\0&0&-1/3\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}=\begin{pmatrix}-1/3\\2/3\\-4/3\end{pmatrix}$$

⑤ (3,3)要素が 1 になるように、三行目を(3,3)要素で割る。

$$\begin{pmatrix}1&0&2/3\\0&1&-1/3\\0&0&1\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}=\begin{pmatrix}-1/3\\2/3\\4\end{pmatrix}$$

⑥ 3 列目の(3,3)要素以外が 0 になるように、行同士の引き算をする。

$$\begin{pmatrix}1&0&2/3\\0&1&0\\0&0&1\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}=\begin{pmatrix}-1/3\\2\\4\end{pmatrix}$$

$$\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}=\begin{pmatrix}-3\\2\\4\end{pmatrix}$$

となって、単位行列に変形できたので、

$$\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}=\begin{pmatrix}-3\\2\\4\end{pmatrix}$$

めでたく答えが出た。

今の例では、対角要素が 0 になる場合がなかったので、問題はなかったが、対角要素が 0 になる場合は、0 でない要素を持つ列を、対角要素が 0 でない別の行と、行ごと入れ替えるという操作が必要である。これをピボット選択といい、通常は「ピボット選択つきガウスジョルダン法」が使われる。ピボット(pivot)とは軸とか中心とかいう意味。

4.3. LU 分解法

ガウスジョルダン法の場合は、**A** を単位行列に変換する際、***b*** の部分も同時に計算する必要があった。だから、例えば同じ **A** を使い、***b*** だけ変えて繰り返し計算したい場合など、すべて初めのステップから計算しないといけないので、同じ行列を使って何度も繰り返し実行したいときなどは、ガウスジョルダン法は、良いアルゴリズムとは言えない。そこで、次に **A** だけを操作して、解きやすい形にするという、LU 分解法を次に紹介する。L とは、下三角行列(Lower triangular matrix)、U とは上三角行列(Upper triangulaer matrix)の略で、

$$L=\begin{pmatrix}*&0&0&0\\*&*&0&0\\*&*&*&0\\*&*&*&*\end{pmatrix},\ U=\begin{pmatrix}*&*&*&*\\0&*&*&*\\0&0&*&*\\0&0&0&*\end{pmatrix}$$

という形をした、非対角要素がそれぞれ左上と右下で 0 のような行列のこと。(アスタリスク(*)はどんな数字が入ってもいいことを表す。) 行列計算では、「元の行列を分解して楽に解く方法」、いわゆる「○○分解」という方法が頻繁に出てくるので、そういう意味でもまず LU 分解に慣れておくといいだろう。

具体的な行列で例を示すとアルゴリズムの理解がしやすいので、**4.1** で取り上げた問題をそのまま流用しよう。

①LU 分解

まずは単位行列 **I** と **A** を並べて書く。つまり、これから **A=LU** に分解したいので、まずは **A=IA** と分解したところからスタートしようというわけ。ステップを経ながら **I** をだんだんと **L** に、**A** をだんだんと **U** に変形していくという手法をとる。分かりやすいように、行列の肩にステップの番号を括弧書きでつけておく。

$$\mathbf{I}=\mathbf{L}^{(1)}=\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\qquad\mathbf{A}^{(1)}=\begin{pmatrix}1&2&0\\1&-1&1\\2&0&1\end{pmatrix}$$

ステップ 1　**A** を **U** に変形したいのだから、行列 $\mathbf{A}^{(1)}$の(2,1)要素と、(3,1)要素を 0 にしたい。そのため、**A** の行同士の引き算を行う。今の例では、(1,1)要素が 1 なので、二行目から一行目を引き、三行目からは、一行目を 2 倍して引く。$\mathbf{L}^{(0)}$に対しても、同じ行同士の計算をする。ただ、$\mathbf{L}^{(0)}$は単位行列なので、行ごとの計算をしたとき、何倍して引いたか、その数を $\mathbf{L}^{(0)}$の(2,1)と(3,1)要素に書くだけでいい。これを新たに $\mathbf{L}^{(1)}$とする。上の例では、$\mathbf{A}^{(1)}$の(2,1)要素を 0 にするには、一行目を 1 倍して二行目から引き、$\mathbf{A}^{(1)}$の(3,1)要素を 0 にするには、一行目を 2 倍して三行目から引けばよい。この結果を $\mathbf{A}^{(2)}$ とする。また $\mathbf{L}^{(1)}$の(2,1)、(3,1)要素は、それぞれ 1、2 と書けばよい。(このときマイナスが付かないことに注意)

$$\mathbf{L}^{(2)}=\begin{pmatrix}1&0&0\\1&1&0\\2&0&1\end{pmatrix}\qquad\mathbf{A}^{(2)}=\begin{pmatrix}1&2&0\\0&-3&1\\0&-4&1\end{pmatrix}$$

ステップ 2　次に、$\mathbf{A}^{(2)}$の(3,2)要素を 0 にしたい。そのため、$\mathbf{A}^{(2)}$の(2,2)要素(=−3)について同じことを行う。二行目を 4/3 倍して三行目から引けば $\mathbf{A}^{(2)}$の(3,3)要素が 0 になる。したがって $\mathbf{L}^{(3)}$の(3,2)要素が 4/3 になる。

$$\mathbf{L}^{(3)}=\begin{pmatrix}1&0&0\\1&1&0\\2&4/3&1\end{pmatrix}\qquad\mathbf{A}^{(3)}=\begin{pmatrix}1&2&0\\0&-3&1\\0&0&-1/3\end{pmatrix}$$

これで、$\mathbf{L}^{(3)}$ と $\mathbf{A}^{(3)}$が求まった。

ステップ 3 以降　3×3 より行列が大きいときは、さらにこのステップを行列のサイズ分だけ繰り返す。今の例では、$\mathbf{A}^{(3)}$が上三角行列 **U** になったので **U** =$\mathrm{A}^{(3)}$として終了。$\mathbf{L}^{(3)}$が下三角行列 **L** になっている。本当に LU 分解されたのか? 検算してみる。

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

確かに **L**×**U** は **A** に等しくなっており、LU 分解できている。ここまでは、***b*** が計算に含まれていないことに注意したい。従って、

$\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$ の $\boldsymbol{b}$ が変わっても、LU 分解は一回で済む。これが LU 分解法が、ガウスジョルダン法より優れた点だ。(それ以外にも、LU 分解は、ガウス方より若干高速であるという利点もあるので、LU 分解がよく使われる。)　ちなみに LU 分解は、一意に決まるものではないので、例と同じ値になっていなくても、**A=LU** となっていれば問題はない。

②前進消去

A =LU と分解できたので、連立方程式 $\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$ は $\mathbf{A}\boldsymbol{x}=\mathbf{LU}\boldsymbol{x}=\boldsymbol{b}$ と書き直せる。$\mathbf{U}\boldsymbol{x}=\boldsymbol{c}$ とおけば、さらに $\mathbf{L}\boldsymbol{c}=\boldsymbol{b}$ となる。まず $\boldsymbol{c}$ を求めよう。

$$\boldsymbol{c}=\begin{pmatrix}c_1\\c_2\\c_3\end{pmatrix}$$

とかけば、$\mathbf{L}\boldsymbol{c}=\boldsymbol{b}$ は、

$$\begin{pmatrix}1&0&0\\1&1&0\\2&4/3&1\end{pmatrix}\begin{pmatrix}c_1\\c_2\\c_3\end{pmatrix}=\begin{pmatrix}1\\-1\\-2\end{pmatrix}$$

つまり、

$$\begin{cases}c_1=1\\c_1+c_2=-1\\2c_1+4c_2/3+c_3=-2\end{cases}$$

となるので、c_1=1 を次の行に代入すると、c_2 が得られ、得られた c_2 をその次の行に代入していけば、一ステップごとに、c_2, c_3 がそれぞれ求まっていく。これを前進消去という。結局 $\boldsymbol{c}$ は、

$$\begin{cases}c_1=1\\c_2=-2\\c_3=-4/3\end{cases}$$

と求まる。なぜ L と U に分解したかったのか、その理由は、前進消去したかったからだということが、分かってもらえただろうか。

③後退消去

次は $\mathbf{U}\boldsymbol{x}=\boldsymbol{c}$ を求める。

$$\begin{pmatrix}1&2&0\\0&-3&1\\0&0&-1/3\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}=\begin{pmatrix}c_1\\c_2\\c_3\end{pmatrix}=\begin{pmatrix}1\\-2\\-4/3\end{pmatrix}$$

$$\begin{cases}x_1+2x_2=1\\-3x_2+x_3=-2\\-x_3/3=-4/3\end{cases}$$

なので、今度は一番下の行から x_3=4 が求まり、上の行につぎつぎと代入していく。これを後退消去という。したがって、

$$\begin{cases}x_1=-3\\x_2=2\\x_3=4\end{cases}$$

と答えが出る。手計算しようとすると、それなりに面倒なのだが、連立方程式を解く解法として優れたアルゴリズムであることが知られている。LU 分解のプログラム化は省略するので、各自手を動かして作ってみてほしい。

Octave には、LU 分解してくれる関数 **lu** がビルトイン関数として用意されているので、それを試しに使ってみる。使い方は、

```
> A=[1 2 0;1 -1 1; 2 0 1];
> [L U P]=lu(A)
L =
 1.00000  0.00000  0.00000
 0.50000  1.00000  0.00000
 0.50000 -0.50000  1.00000
U =
 2.00000  0.00000  1.00000
 0.00000  2.00000 -0.50000
 0.00000  0.00000  0.25000
P =
 0  0  1
 1  0  0
 0  1  0
```

という感じで 3 つの出力、**L** と **U** と **P** が求まる。3 つ目の **P** は、対角要素が **0** になってしまう場合に行う、ピボット選択のための行列で、置換行列といい、**PA= LU** となっている。**P** はどの行にも、どの列にも 1 が一つずつある行列で、$\mathbf{P}^{-1}=\mathbf{P}^{\mathrm{T}}$ である。**A** が正則なら、ピボット選択つき LU 分解で、必ず LU 分解できることが知られている。

4.3.逆行列で求める場合との比較

逆行列で連立方程式を解くのは良くない方法だという話を先にしたが、どのくらい良くないのか、実際にコンピュータ上で実験して比較してみよう。

連立方程式を解くのに Octave で便利なコマンドは**¥**だ[1]。(**mldivide** というビルトイン関数と同じ)　これは、行列の左除算を表す演算で、例えば $\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$ を求めたいなら、

[1]なぜ**¥**マークを使うかというと、JIS 規格でバックスラッシュ\を、**¥**マークに割り当ててしまったからだ。多くの日本語パソコンのキーボードにあるひらがなの「ろ」のキーには、[-\ろ] と書かれていると思うのだが、普通はこれを押しても**¥**しか表示されない。これは、****と**¥**が同じ ACSII コードだからだ。Backspace の左横にも¥が書かれたキーボード [|¥ -] と

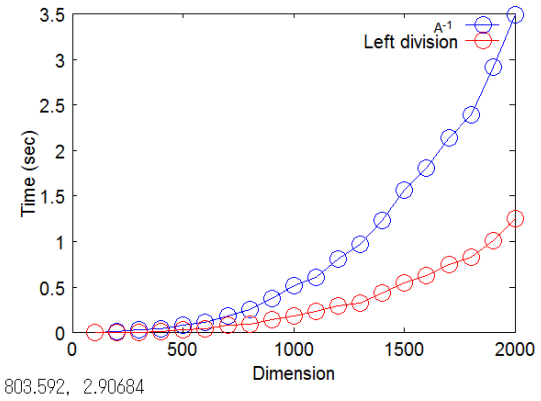
```
> x=A¥b
```

とすればよい。これは、A の逆行列を左からかける A^-1*b と、数学的には同じ意味なのだが、A¥b の場合は、逆行列を求めず、A の左除算を行うという点で、数値計算上は異なるアルゴリズムを使った演算だ[2]。

ではこれを使って連立方程式の解を求めるプログラムを書き、逆行列で求める方法と、どちらが早く計算できるか比較してみよう。関数 cputime を使って、実際に CPU が稼働している時間を測定する。

```
% Leftdiv.m
clear ;max=20; %最大の行列サイズを 2000×2000 に。
d=(1:max).*100;
for k=1:max
  A=rand(100*k);     %行列 A をランダムに作る。
  b=rand(100*k,1);   %ベクトル b をランダムに作る
  t=cputime;         %実行直前の CPU 時間の計測
  A^-1 * b;     %逆行列
  invs(k)=cputime-t; %計算時間を invs に記録
  t=cputime;
  A¥b;          %左除算
  ldiv(k)=cputime-t;  %計算時間を ldiv に記録
end
plot(d,invs,'bo-', d,ldiv,'ro-');
xlabel('Dimension');
ylabel('Time (sec)');
legend('A^{-1}','Left division');
```

これを実行すると、図 4.3.1 のような結果になるはず。(乱数を使っているので毎回結果は微妙に違う。) 使っているパソコンによっては、2000×2000 程度のサイズの行列の逆行列を求めるときに、メモリが足りないというエラーが出るかもしれない。そのときは、max=20 の部分を減らして試してみるとよい。



803.592, 2.90684

図 4.3.1 逆行列と左除算による計算時間の比較

横軸が行列のサイズで、縦軸がその問題を解くのに要した時間。理論的には、両者とも行列のサイズの 3 乗に比例して、計算量が増えていくので、実験の結果と良く合っている。逆行列を使う解法(A^-1)では、左除算(Left division)より 3 倍近く遅くなっている様子が分かるだろう。 この相対的な遅さが、アルゴリズムのまずさに起因しているわけだ。もちろん、絶対的な速さは、パソコンの CPU やメモリ、言語、ソフトウェアに依存する。Octave のソースコードは市販の MATLAB で動くので、試しに Leftdiv.m を MATLAB で実行したところ、Left division の結果はほとんど同じだが、A^-1 の方は、Octave を使ったときの方が速かった。

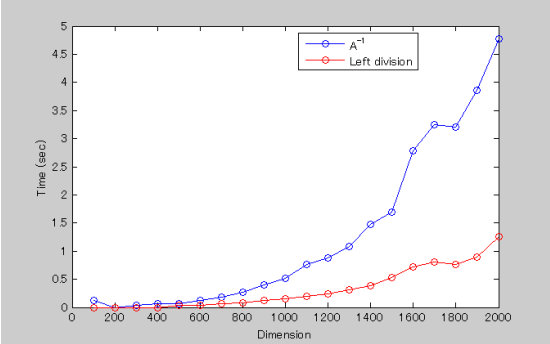


図 4.3.2 MATLAB で実行したときの計算速度

LU 分解そのものを自分でプログラミングすることは少ないと思うが、LU 分解は市販のプログラム内部で使われていることが非常に多いので、パソコンや、ソフトウェアの性能を比較するための指標(ベンチマーク)としても利用されている。

いうキーがあり、これを押しても¥が表示されるが、これも同じ ACSII コードを表している。

[2] A がどういう行列かによって、¥を実行するのに使われるアルゴリズムが若干異なるので、すべて場合で LU 分解が使われているわけではない。