連立 1 次方程式の基礎知識

~ および Gauss の消去法の安定性について ~

渡部 善隆 [†]

学会や研究集会での発表や、そこでの議論を聞いていると、(時間が限られていることもあって) ある「知識」を持っていることが前提となって話が進んでいく場合がよくあります。特に数値計算の分野では、「××法」と名のつく方法が山のように登場します。その方法の「意味」がわかっているなら問題なく話について行けるのですが、何を言っているのかイメージすら掴めないことが往々にしてあります。この場合、そこからの議論に取り残されてしまい、“ぼやっ ~” と聞いているうちに完全に興味を失ってしまい、「セッションが終ったら何を食べにいこうかなぁ」などと違うことを考えるようになります。それはよろしくありません。

この記事は、科学技術計算の主役の一つである連立 1 次方程式の数値解法に登場する手法および用語の解説です。この分野の用語の特徴は、“略語だらけ” で、“よく知らない人の名前が多い” ことです。解説は最低限の意味の説明にとどめ、数学的な証明や具体的なアルゴリズムはすべて参考文献に押しつけています。さらに詳しく知りたい方は、記事の終りに文献紹介および「優れた文献を紹介している文献を紹介」しますので、そちらを御覧下さい。

なお、最後の章で Gauss の消去法の安定性に関する興味ある (ある面で恐ろしい) 話題を紹介します。こちらも「知識」として持っていると、例えば座長をしていて質問に窮したときや、懇親会で知らない人に囲まれたときなどに役立つかも知れません。

この記事によって、多少ともみなさんの “ぼやっ ~” とする時間が減ることになれば幸いです。

目次

1 Introduction 293
- 1.1 線形作用素 293
- 1.2 近似モデル 293
- 1.3 非線形を線形にするからくり 294
- 1.4 Newton-Raphson 法と連立 1 次方程式の関係 295
- 1.5 行列の次元 295
- 1.6 精度の問題 296

2 連立 1 次方程式 297
- 2.1 行列の種類 297
  - 2.1.1 密行列 297
  - 2.1.2 疎行列 297
  - 2.1.3 帯行列 297
  - 2.1.4 三重対角行列 297
  - 2.1.5 対称行列 298
  - 2.1.6 対角行列 298
  - 2.1.7 下三角行列 298
  - 2.1.8 上三角行列 298
  - 2.1.9 Hessenberg 行列 298
  - 2.1.10 正定値行列 298
  - 2.1.11 対角優位行列 298
  - 2.1.12 M 行列 299
  - 2.1.13 置換行列 299
  - 2.1.14 直交行列 299
- 2.2 解法の大まかな分類 299
- 2.3 直接法 300
- 2.4 反復法 300
- 2.5 共役勾配法 300
- 2.6 各方法の組合せ 300

3 直接法の概要 301
- 3.1 Cramer の公式 301
- 3.2 Hamilton-Cayley の定理に基礎をおく方法 301
- 3.3 三角化に基礎をおく方法 301
  - 3.3.1 Gauss の消去法 302
  - 3.3.2 Crout 法 303
  - 3.3.3 Doolittle 法 303
  - 3.3.4 Cholesky 法 303
  - 3.3.5 修正 Cholesky 法 303
  - 3.3.6 Bunch の方法 304
  - 3.3.7 越境法 304
- 3.4 対角化に基礎をおく方法 305
  - 3.4.1 Gauss-Jordan の消去法 . . . 305
  - 3.4.2 逆行列の積表現 305
  - 3.4.3 合同変換による方法 305

[†] 九州大学大型計算機センター・研究開発部

3.5 直交化法 305
3.5.1 直交ベクトル法 305
3.5.2 直交行列法 305
3.6 分割法 306
3.6.1 分割計算法 306
3.6.2 縁どり法 306
3.6.3 エスカレータ法 306
3.6.4 階数零化法 306
3.7 三重対角行列専用の方法 306
3.7.1 Gauss の消去法, Cholesky 法 306
3.7.2 cyclic reduction 法 306
3.8 疎行列, 帯行列専用の方法 306
3.9 現在人気の解法 307

4 反復法の概要 309
4.1 反復法の創始者 309
4.2 Jacobi 法 309
4.3 Richardson 法 310
4.4 Gauss-Seidel 法 310
4.5 SOR 法 310
4.5.1 SLOR 法 310
4.5.2 SSOR 法 311
4.5.3 SBOR 法 311
4.6 ADI 法 311
4.7 Aitken の δ^2 法 311
4.8 Chebyshev 加速法 311
4.9 モンテカルロ法 312
4.10 反復改良法 312

5 共役勾配法の概要 313
5.1 鮮烈なデビューとその後 313
5.2 基本的な考え方 313
5.3 計算の手順 314
5.4 最急降下法 314
5.5 共役方向法 314
5.6 共役勾配法 315
5.7 前処理付き共役勾配法 315
5.7.1 ICCG 法 316
5.7.2 MICCG 法 316
5.8 共役残差法 316
5.8.1 ILUCR 法 316
5.8.2 MILUCR 法 317
5.9 GMRES 法 317
5.10 双対共役勾配法 317
5.10.1 ILUBCG 法 317
5.10.2 MILUBCG 法 317
5.11 QMR 法 317
5.12 BiCGStab 法 317
5.13 共役勾配法の復活 318
5.14 反復解法の汎用性 319
5.15 INSPEC を用いた実験 319

6 Gauss の消去法のアルゴリズム 321
6.1 Gauss の消去法 321
6.2 消去過程 322
6.3 LU 分解との関係 322
6.4 ピボット選択 323
6.5 ピボット選択の利点 323
6.6 ピボット選択が必要ない行列 324
6.7 スケーリング 324
6.7.1 部分ピボット選択の場合 . . . 324
6.7.2 完全ピボット選択の場合 . . . 325
6.8 プログラム例 325

7 GECP のスピードアップ作戦 327
7.1 実行時間の調査 327
7.2 行列の値の交換 328
7.2.1 親切なコンパイラ 328
7.2.2 最適化指示行の挿入 328
7.2.3 本当に行と列の値を交換する 329
7.3 添字の動き 329
7.4 GECP プログラム 330
7.5 実行時間の比較 332

8 Gauss の消去法は安定か？ 333
8.1 消去法の歴史 333
8.2 後退誤差解析 334
8.2.1 マシンイプシロン 334
8.2.2 条件数 334
8.2.3 誤差評価 335
8.2.4 悪条件の場合の対策 335
8.2.5 Hilbert 行列 335
8.3 不安定因子の上限 336
8.4 Wilkinson さんの例題 336
8.5 なぜ部分ピボット選択法が使われるのか？ 338
8.6 random matrix での実験 339
8.7 病理学的な例 340
8.7.1 Wright さんの例 340
8.7.2 Foster さんの例 341

illustration : Hidaki, Naoko

1 Introduction

序章では、連立 1 次方程式が数値計算に登場する背景について考えてみます。

なお、厳密に話を進めると定義がたくさん必要になり、読む人を退屈させますので、かなり荒っぽい説明になることをご勘弁願います。

また、以下特に断わらない限り、小文字は実数または実数のベクトル、大文字は実数の正方行列を表します。

1.1 線形作用素

連立 1 次方程式[1]($linear\ equations$) は、数値計算を試みる人の前に頻繁に立ち現れる問題です。

方程式は

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \qquad i = 1, \cdots, n \tag{1}$$

で表記されます。未知数は n 個の $x_j (j = 1, \cdots, n)$ です。

(1) を行列形式で書けば

$$Ax = b \tag{2}$$

です。連立 1 次方程式の数値解法 ($numerical\ solution\ of\ linear\ equations$) とは、行列 A とベクトル b に対し、(2) をみたす x を数値的に求める方法です。

なお、この記事は全て実数の世界で考えますが、基本的には複素数の世界にも拡張できます[2]。

「1 次」とは「線形[3]」のことです。ある作用が線形($linear$) であるとは、それが比を保ち、さらに和に対する作用の結果がそれぞれに対する作用の和に一致することを意味します[4]。

行列 A は、n 次元実ベクトル空間 $\mathbb{R}^n$ から同じく n 次元実ベクトル空間 $\mathbb{R}^n$ への線形な作用素として見ることができます。

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \xrightarrow{A} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

線形作用素が逆作用素をもつとき、この作用素は正則($non\text{-}singular$) であるといいます。行列 A が正則であるとは、

$$A^{-1}A = AA^{-1} = I \text{ (単位行列)}$$

となる逆行列 ($inverse\ matrix$) A^{-1} がただひとつ存在することです。

連立 1 次方程式 (2) は、逆行列の値がきちんとわかるならば、行列とベクトルの積

$$x = A^{-1}b \tag{3}$$

で求めることができます。

ただし、実際の数値計算で逆行列 A^{-1} を露骨に計算にいくことはそんなに多くありません。A^{-1} がきちんとわからなくても、(2) をみたす x をもとめる効率のよい方法がたくさんあるからです。

それらの方法に比べ、逆行列を計算し、積 $A^{-1}b$ の計算で x を求める方法は、多くの手間がかかり、精度もあまりよくない上、普通は記憶領域を大量に消費します ([41] 参照)。

そのため、逆行列の値がすべて必要になるなどの特別なことがない限り、(3) の計算は行なわないのが "常識" になっています。

連立 1 次方程式の数値解法は、次章以降で概観することにします。なお "線形性" についてのイメージが欲しい方は [64] がお勧めです。

1.2 近似モデル

微積分の概念は、自然現象や社会現象を数学の言葉でモデル化することを簡単にしました。

微分方程式や積分方程式のような無限次元[5]の関

[1]「連立一次方程式」と書くこともあります。

[2]四則演算を実数の定義から複素数の定義に切り替えればほとんどそのまま通用します。これは、複素数 z が平面上の点 (x, y) にちょうど対応し、無理なく計算手順が拡張できるからです。

[3]「線型」とも書きます。要は意味が通ればいいのです。

[4]記号で書くと $T(v_1 + v_2) = Tv_1 + Tv_2$, $T(av) = aT(v)$

[5]有限個の基底で表現できないという意味です。連続系ともいいます。

数方程式を数値的に解く場合は、数学モデルを有限次元の近似モデルに置き換える[6]ことが必要です。

情報処理技術の急速な発達は、有限要素法・境界要素法・差分法などの近似モデル化の研究を大いに助けました。

連立 1 次方程式は、この近似モデルを具体的な数値にする段階でしばしば登場します。そして、他の計算[7]に比べ、計算時間がたくさん必要なことと、ちゃんと方程式が解けていないと後でたいへん困るという意味で、数値解析の中心部を占める大切な解法のひとつです。

もう少し詳しく見ていきましょう。

1.3 非線形を線形にするからくり

立てた数学モデルが線形になることはあまりありません。たいていは非線形(*nonlinear*)な方程式になります。

例えば、流体関係の学会などに顔を出すと、こんな微分方程式をよく見かけます。

$$\left\{\begin{array}{rl} \dfrac{\partial v}{\partial t} + (v\cdot\nabla)\,v & = -\mathrm{grad}\,p + \dfrac{1}{\mathrm{Re}}\Delta v \\ \mathrm{div}\,v & = 0 \\ & (v \text{ と } p \text{ の境界条件}) \end{array}\right. \tag{4}$$

(4) は、 Navier-Stokes 方程式[8]とよばれる有名な方程式です (詳しい説明はパスします。気になる方は [46] をお読み下さい)。

(4) などの非線形な方程式を直接解くことは、かなりの困難をともないます[9]。そこで数値解析の常道として、非線形方程式の代わりにある線形方程式の計算をくりかえすことで、近似解の収束列を作り出すということをよくやります。

その代表的な方法である **Newton-Raphson** 法を例にとって説明します。

Newton-Raphson 法

うまい具合に関数空間[10]と作用素 F を定義することで、どんな非線形方程式も

$$F(u) = 0 \tag{5}$$

をみたす u を求めるという問題に書くことができます。 F は設定した空間上の非線形作用素です。

設定した関数空間の要素 $u^{(0)}$ で F が微分可能なとき、 $F(u)$ を $u^{(0)}$ のまわりで Taylor 展開[11]し、 3 項目以降を切り捨てて近似します[12]。

$$F(u) \approx F(u^{(0)}) + F'(u^{(0)})(u - u^{(0)}) \tag{6}$$

$F'(u^{(0)})$ は F の $u^{(0)}$ での微分として決まる “線形” な作用素で、逆写像 $F'(u^{(0)})^{-1}$ を持つと仮定します[13]。

ここで、 Taylor 展開で近似した (6) の右辺を 0 にするような $u^{(1)}$ を考えます。つまり、線形方程式

$$F(u^{(0)}) + F'(u^{(0)})(u^{(1)} - u^{(0)}) = 0 \tag{7}$$

を満足する $u^{(1)}$ を見つけることにします。

$F'(u^{(0)})^{-1}$ を (7) の両辺に作用させ、変形すると

$$u^{(1)} = u^{(0)} + F'(u^{(0)})^{-1}F(u^{(0)})$$

で $u^{(1)}$ が求まります。この $u^{(1)}$ は、最初に適当に決めた $u^{(0)}$ よりも真の解 u に近いことが期待されるものです。

そこで、今度は $u^{(1)}$ のまわりで同じように Taylor 展開し、つぎの $u^{(2)}$ を

$$u^{(2)} = u^{(1)} + F'(u^{(1)})^{-1}F(u^{(1)})$$

で求めます。

Newton-Raphson 法とは、このように適当な初期値 $u^{(0)}$ から出発し、 $k = 1, 2, \cdots$ について次々に近似解の列

$$u^{(k)} = u^{(k-1)} + F'(u^{(k-1)})^{-1}F(u^{(k-1)}) \tag{8}$$

を作っていく方法です。

すると、いつかは $u^{(k)}$ と $u^{(k-1)}$ が見わけがつかなくなるくらいに近くなるかも知れません。その時、 $u^{(k)} = u^{(k-1)}$ としてしまい、繰り返しの式 (8) に放り込むと

$$u^{(k)} = u^{(k)} + F'(u^{(k)})^{-1}F(u^{(k)})$$

[6] この段階を「離散化」といいます。

[7] 行列やベクトルを作ったり、収束性を調べたりメッシュを切ったりする作業。

[8] Louis Marie Henri Navier さん (1785–1836) と George Gabriel Stokes さん (1819–1903) の名前に由来します。

[9] 解析的な解を求めるときも、数値的な近似解を求めるときもです。

[10] よくみかける空間を記号だけ紹介すると、 $\mathbb{R}^n$(自分から自分への写像だと思って下さい), S_h, L^p, L^∞, $W^{m,p}$, H^m, H_0^m, H^{-1}, C^m, C_0^∞ などなどです。

[11] イギリスの数学者 Brook Taylor さん (1685–1731) に由来します。

[12] 微分の定義や Taylor 展開の話、また、なぜ近似になるのかなどの説明は省略します。要は大学の最初に習う 1 変数の実関数 $f(x)$ の延長だと思って下さい。

[13] なぜ “線形” かというと、訳は簡単で、線形になるように決めているからです。関数解析学では $F'(u^{(0)})$ を線形化作用素 (*linearized operator*)ともいいます。実際に定式化する場合、 F の微分可能性や関数空間の設定を注意深く説明しないと、関数空間論の人から怒られます。

つまり

$$F'(u^{(k)})^{-1}F(u^{(k)})=0$$

となります。この両辺に $F'(u^{(k)})$ を引っかけて

$$F(u^{(k)})=0$$

が得られます。つまり $u^{(k)}$ は $F(u^{(k)})=0$ の解に限りなく近いに違いない、という作戦です。

Newton-Raphson 法は "$F(u)$ はごく狭い範囲では 1 次式で近似できる" という考え方をもとにした方法で、出発点がもし真の解に十分近ければ、極めて急速に収束することが知られています。

この節の最初に登場した Navier-Stokes 方程式 (4) も、もちろん (8) の形に記述することができます (具体的な方法は [16] を御覧下さい)。

Newton-Raphson 法は、簡単に Newton 法と呼ばれることもあり、修正 Newton 法、準 Newton 法、区間 Newton 法など、いろいろな変形版も加え、広く非線形方程式の数値解法に使われています[14]。

Newton-Raphson 法をはじめとする非線形方程式の反復解法、近似解の精度的保証などの数学的な考察は [47] をお読み下さい。

なお、 Newton-Raphson 法は、有名な Sir Isaac Newton さん (1642–1727) と Joseph Raphson さん (1648?–1715?) に由来します。

1.4 Newton-Raphson 法と連立 1 次方程式の関係

ここで、本稿の主役の連立 1 次方程式が Newton-Raphson 法の何処に顔を出すのか探してみましょう。

先ほど考えていた非線形作用素 F は、特に $\mathbb{R}^n$ 上の作用素でなくとも構いません。空間が $\mathbb{R}^n$ の場合、 $F'(u^{(k)})$ は F の Jacobi 行列となります。

Sobolev 空間[15]などの無限次元の関数空間の場合は、 $F'(u^{(k)})$ を F の Fréchet 微分[16]とすることで、そのまま Newton-Raphson 法が適用できます (無限次元の関数空間の議論は、 [40], [44] などをお読み下さい)。

しかし、無限次元での定式化の場合は、数値計算のために、何らかの方法で無限次元を有限次元に "射影する" ことが不可避の作業ですので、計算機の中では結局有限次元の問題、さらに最終的には $\mathbb{R}^n$ での線形計算に落ち着きます[17]。

(8) が有限次元空間、特に $\mathbb{R}^n$ での反復だとすれば、線形作用素は行列の形で書くことができます。従って、

$$u^{(k)}=u^{(k-1)}+\underline{F'(u^{(k-1)})^{-1}F(u^{(k-1)})}$$

のアンダーラインの部分は、「ベクトル $F(u^{(k-1)})$ に行列 $F'(u^{(k-1)})$ の逆行列を左から引っかける」ことを意味しています。

よって反復式 (8) は、連立 1 次方程式

$$F'(u^{(k-1)})\hat{u}^{(k-1)}=F(u^{(k-1)})$$

を $\hat{u}^{(k-1)}$ について解き、この $\hat{u}^{(k-1)}$ を使って、次の $u^{(k)}$ を

$$u^{(k)}=u^{(k-1)}+\hat{u}^{(k-1)}$$

で求めることと同じです。

そのようなわけで、非線形方程式の数値解法では連立 1 次方程式を解く必要がしばしば生じます[18]。

1.5 行列の次元

有限次元で例えば変数が 1 つの

$$f(x)=x^3+x^2+x+1=0$$

を Newton-Raphson 法で解く場合、反復列は

$$x^{(k)}=x^{(k-1)}+f'(x^{(k-1)})^{-1}f(x^{(k-1)})$$

です。このとき、変数は 1 個なので、 $f'(x^{k-1})$ は 1×1 の行列です。よって、逆行列はいたって簡単で

$$f'(x^{(k-1)})^{-1}=\frac{1}{f'(x^{(k-1)})}$$

で求まります。

[14] 昔の数値計算の教科書には「Newton-Raphson 法」と書かれていましたが、現在は「Newton 法」とだけ書く本が増えました。しかし、 [50] によれば、 Newton さんは確かに方法は示していたが、一般的な公式にまとめたのは Raphson さんだそうです。ここでは、 2 年程前に開催されたシンポジウムで、ある数値計算の大御所が「Raphson 氏の偉大な業績を忘れて、皆 Newton 法と呼ぶのはけしからん！ Newton-Raphson 法と呼べ！」と吠えていたので、 Newton-Raphson と書いておきます。

[15] Sergeǐ L'vovich Sobolev さん (1908–?) は、 (当時) ソビエトの数学者です。

[16] René Maurice Fréchet さん (1878–1973) は、フランスの数学者です。

[17] つまり、すでに有限次元の非線形問題になっているものを Newton-Raphson 法で解くか、無限次元で定式化した Newton-Raphson 法を離散化して解くかの違いです。手法は数値解に対してどのレベルの誤差評価を期待するかで異なります。

[18] この大きな流れを押さえていないばっかりに、修士論文の発表会で「結局それは何を計算するんですか？」という (温かい) 質問に絶句してしまい、著しく心証を悪くした人を知っています。ご注意下さい。

一方、多次元の連立非線形方程式

$$f(x) = 0$$
$$x = (x_1\ x_2\ \cdots\ x_n)^T$$
$$f = (f_1\ f_2\ \cdots\ f_n)^T$$

の場合[19]、$f'(x^{(k-1)})$ は $\partial f_i/\partial x_j$ を (i,j) 成分とする $n \times n$ の Jacobi 行列になります (詳しくは [60])。

もともとの問題が無限次元の場合、方程式の次元は離散化する近似空間の基底の数に一致します。差分法や有限要素法などを用いる場合、未知数は領域にばらまいた node の数に対応しますので、この総数が連立 1 次方程式の次元になります[20]。

1.6 精度の問題

さて、いざ連立 1 次方程式を解き、数値解を求める段階にくると、「誤差」の問題が浮上します。つまり、計算者はどれくらいの精度を数値結果に要求するかという問題です。

数学モデルとしての非線形方程式の解の存在や分岐現象などの挙動に興味のある人でしたら、収束性や誤差解析も含めて、数値解に高い精度を要求するはずです。

一方、もともと自然現象や社会現象を数学モデル化し、最終的な数値解をシミュレーションとして利用したいと思う人の立場から見れば、

> 既に方程式を立てた段階である程度の誤差が入りこんでいるので、連立 1 次方程式をはじめとする数値計算を完璧な精度で求める必要はないし、また意味もない。

と思うでしょう。

微分方程式などのモデルから出発したとしても、計算にかかるコストの問題から、

> どうせ離散化の段階で誤差が入るので、連立 1 次方程式の解法に高い精度を追求して計算時間を無駄にするより、多少の誤差の混入には目をつむって、高速なプログラムを組もう。

と思う人もいるでしょう。

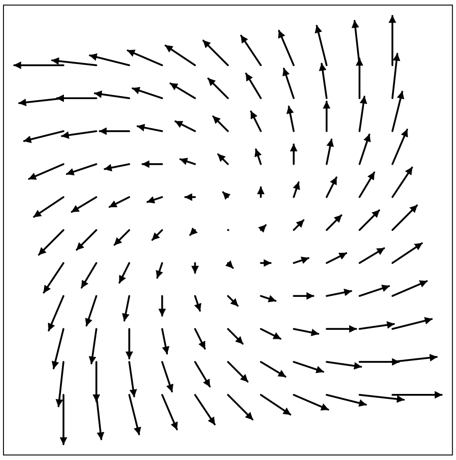
しかしながら、自分のプログラミング技術はともかく、数値計算の普通の演算方法である浮動小数点演算を脳天気に信用するのはちょっと危険です[21]。

少なくとも、学会や論文発表では「なんとなく実験結果と似ているから大丈夫でしょう」という必殺の切札に加えて、「連立 1 次方程式を含む数値計算が丸め誤差の混入によって致命的な精度損失を (多分) 起こしていないであろう」ことを強弁するだけの根拠[22]が必要です。なぜなら、

> もし、連立 1 次方程式を含む数値計算が丸め誤差によって致命的な精度損失を起こしていた場合、モデルを設定し、シミュレーションした意味がなくなってしまう

からです。

連立 1 次方程式の解の信用性については、最後の章で列変換をともなう Gauss の消去法の安定性の話題を切り口にして、もっと掘り下げて考えたいと思います。



線形作用素 $A = \begin{pmatrix} 1/2 & -1/2 \\ 1/2 & 1/2 \end{pmatrix}$ によって定義される 2 次元ベクトル場 $(-5 \leq x_1, x_2 \leq 5)$

[19] 記号 "T" は、転置を意味します。

[20] テレビのヒーローもので、4 次元や 5 次元の世界からやってくるのがありますが、まだまだ甘いといえるでしょう。スーパーコンピュータでは数万次元の世界を日常的に扱っています (もっとも、設定している空間の意味が全然違いますが...)。ただし、2 次元世界 (鏡の国だそうです) からやってきた正義の味方「ミラーマン」はすごいと思います。

[21] 興味のある方は [62] を御覧下さい。

[22] 例えば、問題自体の安定性、行列の性格の良さ、数値計算ライブラリの信用性、拡張精度との比較、誤差評価、優秀な学生がプログラムしたから大丈夫、この私が言うのだから間違いない！etc.

2　連立 1 次方程式

前章では、数値計算の中で連立 1 次方程式が出現してくる手順を簡単な例に沿って見てきました。

連立 1 次方程式の係数行列 A は、 n 次元ベクトル空間 $\mathbb{R}^n$ 上の線形作用素であり、 A の形は非線形問題の線形化の過程で決定されます。そして、モデルや離散化の方法によって行列 A の形は異なります。

連立 1 次方程式の数値解を求める段階になると、行列 A の形にあわせた様々な解法が登場します。これまでに提案され、現在も誰かが利用している手法だけでも 100 種類以上あるのは確実でしょう。

この章では、線形作用素である行列 A の性格を分類した後、連立 1 次方程式の数値解法を概観したいと思います。

個別の解法の説明は 3, 4, 5 章で行います。

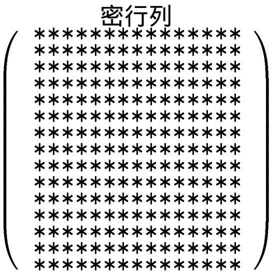
2.1　行列の種類

連立 1 次方程式の解法は、行列 A の形に大きく依存します。そのための基礎知識として、行列用語を簡単に説明します。

2.1.1　密行列

密行列 (*dense matrix*) は、とくに規則性や特徴のない一般の行列のことです。「密」とは、 0 以外の要素が行列の中に充満していることを意味します。

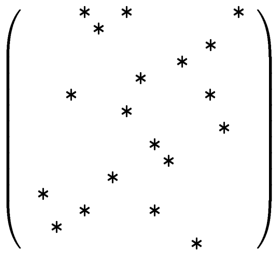
密行列



2.1.2　疎行列

疎行列 (*sparse matrix*) は、成分のほとんどが 0 の行列をいいます。偏微分方程式を離散化して連立 1 次方程式に帰着させる場合、だいたい疎行列になります。

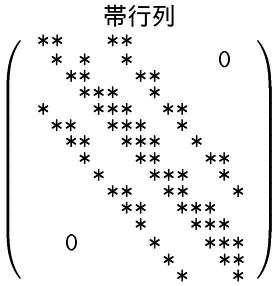
疎行列



2.1.3　帯行列

帯行列 (*band matrix*) は、疎行列であって、かつ 0 でない成分が行列の対角線上と対角線の比較的近くに分布している行列です。これもよく見かけます。

帯行列

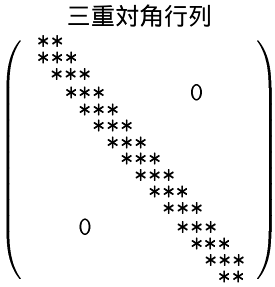


2.1.4　三重対角行列

三重対角行列 (*tridiagonal matrix*) は、帯行列の特別な場合で、 $|i-j|>1$ の成分がすべて 0 な行列です。つまり、 0 でない成分が行列の対角線上と、対角線の前後だけに分布している行列です。三項行列ともいいます。

A が三重対角行列の場合、連立 1 次方程式や固有値問題があっという間に解けます。

三重対角行列



2.1.5　対称行列

対称行列 (*symmetric matrix*) は、$a_{ij} = a_{ji}$ すなわち $A = A^T$ となる行列です。

行列が対称だと、解法の速度、メモリー、安定性などいろいろいいことがあります。

2.1.6　対角行列

対角行列 (*diagonal matrix*) は、対角成分以外がすべて 0 の行列です。以下、対角行列を表す場合は、“diagonal” の頭文字をとって “D” と書きます。

対角行列は大変素晴らしい行列です。なぜなら、逆行列は各成分の逆数をとって並べ直したものですし、固有値にいたっては最初から対角成分に並んでいるからです。

対角行列

```
*
 *
  *         0
   *
    *
     *
      *
       *
        *
         *
          *
           *
    0       *
             *
              *
```

2.1.7　下三角行列

下三角行列 (*lower triangular matrix*) は、対角成分より上の部分

$$\{a_{ij} \,|\, i < j\}$$

がすべて 0 の行列です。

以下、下三角行列を表す場合は、“lower” の頭文字をとって “L” と書きます。

下三角行列

```
*
**
***
****
*****
******         0
*******
********
*********
**********
***********
************
*************
**************
***************
```

2.1.8　上三角行列

上三角行列 (*upper triangular matrix*) は、対角成分より下の部分

$$\{a_{ij} \,|\, i > j\}$$

がすべて 0 の行列です。

以下、上三角行列を表す場合は、“upper” の頭文字をとって “U” と書きます。

上三角行列

```
***************
 **************
  *************
   ************
    ***********
     **********
      *********
       ********
        *******
   0     ******
          *****
           ****
            ***
             **
              *
```

2.1.9　Hessenberg 行列

Hessenberg 行列[23]は、$i > j+1$ ならば $a_{ij} = 0$ となる行列です[24]。

つまり Hessenberg 行列は、上三角行列に

$$a_{21}, a_{32}, a_{43}, \cdots, a_{nn-1}$$

の部分を加えたものです。

Hessenberg 行列

```
***************
***************
 **************
  *************
   ************
    ***********
     **********
      *********
       ********
   0    *******
         ******
          *****
           ****
            ***
             **
```

2.1.10　正定値行列

正定値行列 (*positive definite matrix*) は、固有値が全部プラスの値をとるような行列です。いい換えると、任意のベクトル $x \neq 0$ に対し、2 次形式

$$x^T A x > 0$$

を満足する行列です。正値行列、正定符号行列ともいいます。

A が正定値で、しかも対称行列だったなら、素晴らしくいいことがおきます[25]。

2.1.11　対角優位行列

対角優位行列 (*diagonal dominant matrix*) は、

$$|a_{ii}| > \sum_{i \neq j} |a_{ij}|, \qquad i = 1, \cdots, n$$

をみたす行列です。つまり、対角成分が (絶対値の意味で) 他の成分を圧倒して “でかい” 行列のことです。

優対角行列ともいいます。

---

[23] K. Hessenberg さん (1874–1925) に由来します。

[24] 上 **Hessenberg** 行列 という場合もあります。

[25] ただし、後ほど出てくる Hilbert 行列のように、対称かつ正定値であっても、どうしようもなく性格が悪い行列もあります。

2.1.12 M 行列

M 行列 (*M-matrix*) は、正則かつすべての $i \neq j$ に対し $a_{ij} \leq 0$ であり、かつ逆行列の成分すべてが 0 以上の行列のことです。M 行列は、物理の世界によく出てきます。対称な M 行列は正定値行列になります[26]。

2.1.13 置換行列

置換行列 (*permutation matrix*) は、どの行にも、またどの列にも、ちょうど 1 個ずつ 1 があり、他はすべて 0 となる行列です。

以下、置換行列を表す場合は、“permutation” の頭文字をとって “P” と書きます。

置換行列 P を A の左からかけるということは、A の行の順序を並べかえることを意味し、右からかけるということは、A の列の順序を並べかえることを意味します。

具体的な例をあげます。行列 A を

$$A = \begin{pmatrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ b_1 & b_2 & b_3 & b_4 & b_5 & b_6 \\ c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \\ d_1 & d_2 & d_3 & d_4 & d_5 & d_6 \\ e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \\ f_1 & f_2 & f_3 & f_4 & f_5 & f_6 \end{pmatrix}$$

置換行列 P を

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

とします。すると PA は

$$PA = \begin{pmatrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \\ b_1 & b_2 & b_3 & b_4 & b_5 & b_6 \\ f_1 & f_2 & f_3 & f_4 & f_5 & f_6 \\ e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \\ a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ d_1 & d_2 & d_3 & d_4 & d_5 & d_6 \end{pmatrix}$$

となります。行列 A の《1 行と 3 行》《3 行と 6 行》《6 行と 4 行》《4 行と 5 行》を順番に入れ換えると PA になります[27]。2 行目はそのままです。

次に、P を A の右からかけると

$$AP = \begin{pmatrix} a_5 & a_2 & a_1 & a_6 & a_4 & a_3 \\ b_5 & b_2 & b_1 & b_6 & b_4 & b_3 \\ c_5 & c_2 & c_1 & c_6 & c_4 & c_3 \\ d_5 & d_2 & d_1 & d_6 & d_4 & d_3 \\ e_5 & e_2 & e_1 & e_6 & e_4 & e_3 \\ f_5 & f_2 & f_1 & f_6 & f_4 & f_3 \end{pmatrix}$$

となります。今度は、行はそのままで列が入れ換わっています。順番に、行列 A の《1 列と 5 列》《5 列と 4 列》《4 列と 6 列》《6 列と 3 列》を入れ換えると AP になります[28]。2 列目はそのままです。

置換行列は、行列の分解の際に大変重要な役割を果たしますが、“単位行列もどき” の形をしているため、プログラムの中で配列として登場することはなく、あくまでも裏方に徹します。

2.1.14 直交行列

直交行列 (*orthogonal matrix*) は、転置行列が逆行列となるような行列のことです。つまり

$$A^{-1} = A^T$$

となる行列です。

以下、直交行列を表す場合は “Q” と書きます。

“直交” の意味は、A の第 i 行と第 j 列の内積が

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

となることからきています。

直交行列を複素数に拡張した行列がユニタリ行列 (*unitary matrix*) です。

直交行列は固有値問題の解法で主役をはりますが、連立 1 次方程式では単なる脇役の一人に過ぎません。

2.2 解法の大まかな分類

連立 1 次方程式の数値解法は、次のように大別できます。

- 連立 1 次方程式の数値解法
  - 直接法
  - 反復法
  - 共役勾配法
  - 各方法の組合せ

以下、各解法の特徴をあげます。

[26] 対称であることを M 行列の定義に加えている本もあります。

[27] 巡回置換 (cycle) の記号で書くと (13645) です。

[28] こちらは (15463) です。

具体的な手法は、章を改めて概観します。

2.3 直接法

直接法 (*direct method*) は、もしも丸めの誤差がなければ[29]有限回の演算で厳密な解が得られる解法のことです。

直接法に必要な演算回数は、行や列の入れ換え操作が多少入るので不確定なところがありますが、計算する前からわかります。

そのため、行列の性質や精度に関係なく、ほぼ同じ計算時間で答が出てきます。

$$A,\ b \longrightarrow \boxed{\text{有限回の演算}} \longrightarrow x$$

行列が密行列であったり、都合のいい性質を持つかどうかわからない得体の知れないものであっても、計算が失敗しない限り、とりあえず解を出してくれます。そのため、直接法は標準的な解法とされています。

直接法の弱点としては、行列が疎行列であっても "0 が多い" という利点を生かしにくく、計算機の記憶容量が反復法に比べたくさん必要なことです。ただし、この困難を克服するためのいろいろな工夫も提案されています。

また、演算回数は普通次数 n の 3 乗に比例します。そのため、次数が多くなればなるほど、計算機の性能をうまく引き出したプログラムを書いておかないと、"有限回で終了する" といいいながらいつまで経っても計算が終らない状況になります。

2.4 反復法

反復法 (*iterative method*) は、適当に選んだ初期値から出発して、真の解に収束していく近似解の列を逐次作成していく手法です。

反復の各段階では、それまでに得られた近似解の情報を駆使し、より良い近似解を生成する工夫が施されています。

$$x^0 \rightarrow x^1 \rightarrow x^2 \rightarrow \cdots \rightarrow x$$

反復法の計算は、行列が 0 でない部分を格納する記憶領域があれば可能です。反復法は、理屈の上では無限解の反復をしない限り厳密解を得ることはできません。そのため、適当なところで反復を打ち切る必要があります。

[29]つまり、すべてを有理数演算で処理するか、無限桁の精度を持つ夢のような計算機で浮動小数点演算が実行できたならば

また、なかなか収束しないような "たち" の悪い行列もあります。そのため、実際に計算をやってみないと演算回数はわかりません。

2.5 共役勾配法

共役勾配法 (*conjugate gradient method*) は、直接法と反復法の長所をあわせ持った手法として、最近とみに注目されています。

原理的には、丸めの誤差がなければ高々 n 回の反復で厳密解が得られます。そのため、直接法の一種といえます。

$$x^0 \rightarrow x^1 \rightarrow \cdots \rightarrow x^n = x$$

が、実際は丸め誤差のためそうはいかず、欲しい精度になるまで反復を続けます。従って、反復解法に分類されることもあります。

共役勾配法は、行列の形を整える "前処理"をほどこすことにより、格段の収束率の向上が得られ、特に正定値対称な疎行列の場合に威力を発揮します。

また、非対称行列にも拡張が行われています。

2.6 各方法の組合せ

浮動小数点演算において発生する丸め誤差を考えれば、直接法を用いたとしても厳密解が得られるわけではありません。

出てくる数値結果からできるだけ誤差をなくすために、直接法で得られた結果を反復法で改良する方法もあります。

また、反復法や共役勾配法の過程で、方程式の一部を直接法を用いて解く方法などもあります。

その他、どんな方法でもいいですが、得られた近似解をもとに、ある集合の中に厳密な解が存在することを誤差限界とともに証明する精度保証付き数値計算法 (*numerical methods with guaranteed accuracy*) もあります。これは区間解析 (interval arithmetic) と不動点定理をもとにした大変興味ある数値計算法です。

最近、有理数演算を含むいくつかの精度保証付きソフトウェアが利用できる環境になりましたので[30]、機会があれば広報で紹介したいと思います。

とりあえずの文献として [25], [65] をあげておきます。

[30]著作者の許可が得られるなら、センターでも公開したいと考えています。是非使ってみたいとお思いの方は、要望を `request@cc.kyushu-u.ac.jp` まで、どしどしお寄せ下さい。

3　直接法の概要

3 章から 5 章までを使って、前章で大まかに分類した連立 1 次方程式の数値解法をさらに細かく見ていきましょう。まずは直接法です。

分類は Westlake さんのハンドブック [34] を参考に、最近の数値計算の本によく載っているものだけを紹介します[31]

これから紹介する方法は、プログラミングの演習問題としては面白いかもしれませんが、実際の研究の手助けとするには心細い方法もあります。

3.1　Cramer の公式

名前は大変よく知られていますが、数値計算で使われることは滅多にありません。 Gabliel Cramer さん (1704–52) が 1750 年に発表した次の定理によっています[32]。

Cramer の公式

A が正則であれば, $Ax = b$ は唯一の解を持ち, 解は

$$x_i = \frac{|A_i|}{|\,A\,|} \qquad i = 1, \cdots, n$$

で与えられる． ただし A_i は A の第 i 列を b で置き換えたもの, $|\cdot|$ は行列式を表す.

むしろ Cramer の公式といえば、上の公式より導かれる 2×2 の逆行列の公式

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad - bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

の方が有名でしょう。

しかし、 Cramer の公式を数値計算で使おうとすると、 $n+1$ 個の n 次行列式を計算せねばなりません。これは、他の解法と比べて反則のように手間がかかる上に、高精度の演算が要求されます。そのため、数値計算の手段としてはほとんど役に立ちません。しかし、線形代数のプログラミングの題材にはいいかも知れません。

なお、 [48] に Cramer の公式の Fortran プログラムが載っています。

3.2　Hamilton-Cayley の定理に基礎をおく方法

William Rowan Hamilton さん (1805–65) と Arthur Cayley さん (1821–95) の名前がついたこの定理も、これまた線形代数の世界ではよく知られています[33]。

Hamilton-Cayley の定理

正方行列 A の固有多項式 (characteristic polynomial) $F(\lambda) = |\lambda I - A|$ をとれば, $F(A) = 0$ となる.

この定理を用いて、 A^{-1} を

$$A^{-1} = \sum_{i=1}^{n} p_i A^{i-1}$$

の形で求めることができます。なお p_i は A^i の対角要素の和を用いて順番に求めることができる実数です。

この方法も、 A のベキを記憶する必要があることと演算回数の問題などから実用的とは言えません。

が、線形代数プログラミングの題材にはいいかも知れません。詳しくは [3] を参照下さい。

3.3　三角化に基礎をおく方法

直接法で現在もっともよく使われるのが、三角化に基礎をおく方法です。プログラムが容易に書ける上、一般の行列に広く適用でき、速度と精度もなかなかのものです。

この手法のもとになる定理は、行列の分解に関する次の定理です。

[31] 連立 1 次方程式の分類および文献としては、 Forsythe さんが 1953 年に発表した有名な論文 [9], [10] があります。その中で紹介してある解法だけでも既に 400 を超えています。

[32] Cramer さんはスイスの数学者です。 Cramer の公式は発表の 20 年後、 Pierre Simon Laplace さん (1749–1827) の紹介で世に広まりました ([17])。

[33] Cayley-Hamilton の定理ともいいます。

LDU 定理

A が正則ならば, 適当な置換行列 P を用いて

$$PA = LDU$$

の形に分解できる. また, L, U の対角要素が 1 のとき, 分解は一意に定まる.

この分解を **LDU** 分解 (LDU decomposition) と呼びます。"三角化" とは、行列 A を L や U などの三角行列の積に分解することをいいます。

3×3 行列で簡単な例を作ってみましょう。P は単位行列となるので省略しています。

$$\underset{A}{\begin{pmatrix} 2 & 2 & 2 \\ 2 & 3 & 3 \\ 4 & 5 & 6 \end{pmatrix}} = \underset{L}{\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2 & 1 & 1 \end{pmatrix}} \underset{D}{\begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}} \underset{U}{\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}}$$

このように、任意の正則な行列は、適当な行の入れ換え[34]によって、下三角行列と対角行列と上三角行列の積の形に分解できます。

また、対角行列 D と上三角行列 U の積を改めて U だと考えれば、直ちに次のことがわかります (証明は [49])。

LU 定理

A が正則ならば, 適当な置換行列 P を用いて

$$PA = LU$$

の形に分解できる. また, L の対角要素が 1 のとき, 分解は一意に定まる.

これが有名な **LU** 分解(LU decomposition) です。

LU 分解の形は、先ほどの LDU 分解で DU が一緒になったものです。

$$\underset{A}{\begin{pmatrix} 2 & 2 & 2 \\ 2 & 3 & 3 \\ 4 & 5 & 6 \end{pmatrix}} = \underset{L}{\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2 & 1 & 1 \end{pmatrix}} \underset{U}{\begin{pmatrix} 2 & 2 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}}$$

P は行の順序を入れ換えるだけの置換行列ですので、「適当に行を入れ換えた行列 A を考える」ことで、正則行列 A は

$$A = LU$$

に分解できると考えることもできます。

行列を LU 分解して、それが何になるかと言えば、この形からスラスラと解が求まるのです。

まず、連立 1 次方程式

$$Ly = Pb$$

を y について解きます。ここで L は下三角行列なので、前進代入 (forward substitution)[35]と呼ばれる簡単な計算方法ですぐに y が求まります。次にこの y を右辺として

$$Ux = y$$

を解きます。U は上三角行列なので、今度は後退代入 (backward substitution)と呼ばれる計算方法で x が求まります。

本当に前進代入と後退代入でスラスラと解が求まるのか疑問の方は [45] を御覧下さい。これらは 6 章でもう一度登場します。

さて、この x は何ものかといいますと、先ほど解いた 2 つの方程式から y を消去することで

$$LUx = Pb$$

の解であることがわかります。ところで、A は $PA = LU$ と分解してましたので、x は

$$PAx = Pb$$

の解です。P は転置行列で正則ですので、結局 x は $Ax = b$ の解であることがわかります。

基本的な $LU(LDU)$ 分解の方法は Gauss の消去法ですが、行列の性格に適応させたり、スピードを追求するなどの目的に合わせてたくさんの変形が考案されています。

3.3.1 Gauss の消去法

本来の意味での Gauss の消去法 (*Gaussian elimination*)は、行列 A を成分の左端から結果が上三角行列になるように順番に消していく過程で、b の処理もあわせて行ないます。つまり、LU 分解の過程で同時に前進代入もやってしまいます。

この手順は、"行列の LU 分解が完了したあとで b について前進代入をする" 形に簡単に変更できます[36]。

従って、最近 Gauss の消去法は「LU 分解法」と呼ばれることも多くなりました ([54])。

Gauss の消去法は、LU 分解の手法によって、また行列の種類により、さまざまな変種があります。

内積形式 Gauss の消去法

内積形式 (inner-product form)では、分解処理を行列の左の列から右の列に、さらに列の上から下に順次行ないます。

[34] 置換行列 P を左から引っかけることと同じです。

[35] 前進消去(forward elimination) ともいいます。

[36] この方が、A はそのまま b を変えて何度も計算する場合に便利です。

データがすべて主記憶に入りきらない大規模な LU 分解で威力を発揮します ([49])。

外積形式 Gauss の消去法

外積形式 (outer-product form)は、 $k = 1, \cdots, n$ に対し、 k 列の消去にひき続き、 k 行の分解を行ないます。数値計算の本に書いてあるアルゴリズムはだいたいこの方法です。

この方法を応用すると、ベクトル計算機向きのプログラムが組めることが知られています。例えば、行列をいくつかのブロックに分割した上で、ブロックごとまとめて分解していく方法[37]などです。

外積形式 Gauss 消去法のアルゴリズムとプログラムの注意点は、 6 章で詳しく考えます。

縁どり法

A が対称行列のときに使われることがあります。分解は外積形式と反対方向に行ないます。

行と列をどこかに追加した新しい行列に対して LU 分解する時に便利です ([59])。

対称 Gauss 法

対称 Gauss 法 (symmetric Gaussian method)は、行列が対称な場合、その性質を利用し計算の手間と記憶容量を半分にするように工夫したものです ([50])。

3.3.2 Crout 法

Crout 法は、 Prescot D. Crout さんが 1941 年に発表した Gauss の消去法の変形版です。

手順は、 Gauss の消去法に出てくる中間結果を記録することを省略し、積計算を順に足し合わせていくことで LU 分解を行います。順序は異なりますが、計算は Gauss の消去法と同じになります。

Crout 法は、計算機の性能をうまく引き出すようにプログラムすると Gauss の消去法以上のスピードが出る場合があり、 LU 分解の有力な方法として幅広く使われています ([45], [49], [59])。

3.3.3 Doolittle 法

Doolittle 法は、 M. H. Doolittle さんが 1878 年に発表した論文から名前がつきました。 Black 法ともいいます。 Doolittle 法も Crout 法と同じように積和の形で LU 分解します。

Crout 法との本質的な違いは、 Crout 法が U の対角成分を 1 とするのに対して、 Doolittle 法は L の対角成分を 1 とするところだけです ([12], [34])。

最近は Crout 法の方が有名になってしまい、影がうすくなっています。

3.3.4 Cholesky 法

Cholesky 法[38]は、次の分解定理をもとにしています (証明は [33] 参照)。

Cholesky 分解定理

A が対称かつ正定値ならば, 下三角行列 L が存在し

$$A = LL^T$$

の形に分解できる. さらに, 分解は対角要素を正になるように定めることで一意に定まる.

この分解定理は、 [20] によれば F. R. Cholesky さんが 1916 年に発表した方法です[39]。

Cholesky 法は、この分解定理を利用し、 Gauss の消去法を経由せずに直接 L を積和で計算する手法です。分解ができてしまえば、あとは Gauss の消去法と同様に $Ly = b$ と $L^T x = y$ を解くことで解が得られます。

Cholesky 法の計算は、分解の過程で対角成分の平方根を計算することから、平方根法(*square root method*) ともいいます。分解は L を求めるだけなので、演算回数が一般の Gauss の消去法の半分で済みます。

Cholesky 分解は、 A が正定値でなくても (適当な転置行列を左からひっかけることで) 可能ですが、 L の対角要素が複素数になる場合があります。また、平方根の計算によって、精度が悪くなる場合があります ([51])。

3.3.5 修正 Cholesky 法

修正 **Cholesky** 法 (*modified Cholesky method*)は、別名がいろいろあり、変形 Cholesky 法、改訂 Cholesky

[37] 2 段同時消去法 ([45])、ブロッキング法などがあり、 SSL II の VALU はブロッキング法を用いています ([70])。

[38] 本によっては “Choleski” と書いてあります。

[39] [72] には “Andre-Louis Cholesky” と書いてありました。どちらが正しいのかはよくわかりませんでした。 Cholesky さんの連立 1 次方程式に関する寄与を紹介した Benoit さんの論文 [1](1924 年) によれば、 “Commandant” Cholesky はフランス陸軍地誌部に属していて、第一次世界大戦 (大戦争と書いてありました) 中の 1918 年に戦死されたそうです。また、 *Mathematical Tables and other Aids to Computation* Vol.5, pp.112 (1951) を見ると、どうやら Rainsford[24] さんという人による Benoit さんの論文 (仏語) の英語訳も存在するみたいです。

法、対称 Cholesky 法、 square-root-free Cholesky 法ともいいます。

“みそ” は、 Cholesky 分解で生じる平方根の計算を回避するところにあります。分解は次の定理が保証します。

修正 Cholesky 分解

A が対称かつ正定値ならば,

$$A = LDL^T$$

の形に分解できる. さらに, 分解は対角行列 D の要素を正にすることで一意に定まる.

修正 Cholesky 分解は、分解の形から LDL^T 分解とも呼ばれます。修正 Cholesky 分解も、 Gauss の消去法を経由しません。

なお、上の定理は、 A が対称であれば (適当な置換行列を左から引っかけることで) 成立するのですが、対称正定値行列でないと分解の過程で非常に精度が悪くなることがあります。そこで、普通は A が対称かつ正定値の場合のみに使用されます。

分解ができてしまえば、あとは

$$Ly = b$$

と

$$DL^Tx = y$$

を解くことで簡単に解が得られます。

修正 Cholesky 法も、 Gauss の消去法のように分解の手順でバリエーションが幾つかあります ([59])。

外積形式修正 Cholesky 法

分解列より後の列を常に更新し、計算済みの列は影響を受けません。 A が帯行列、疎行列の場合によく使われます。

内積形式修正 Cholesky 法

外積形式と反対に、計算済みの行データを使って分解列を計算します。小規模行列に用いられます。

縁どり法

左の行から順番に分解行を求める手法です。 A が帯行列、疎行列の場合に使われます。

スカイライン (skyline) 法

envelope 法ともいいます。縁どり法の変形で、疎行列の場合に用います。非対称行列にも拡張がなされています。

3.3.6 Bunch の方法

修正 Cholesky 法のところで、行列 A が対称ならば原理的には LDL^T 分解できるが、精度が悪くなる場合があることを述べました。

Bunch の方法は、 J. R. Bunch さんと L. Kaufman さんが 1976 年に発表した手法 ([4], [5]) で、数値的に安定した行列の分解を与える方法です。

分解は、次の定理に基づいています。

MDM^T 分解

A が対称ならば, 適当な置換行列 P を用いて

$$PAP^T = LDL^T$$

の形に分解できる. ただし, L は対角成分がすべて 1 の下三角行列, D は高々次数 2 の対称なブロックから構成される対称ブロック対角行列.

このような分解は数値的に安定となります。

Bunch さんの方法は、ブロック対角ピボッティング手法 、あるいは “MDM^T 分解” ともいいます。

分解の後は、順番に $Ly_1 = Pb$, $Dy_2 = y_1$, $L^Ty_3 = y_2$, $x = P^Ty_3$ を計算すれば解が求まります。

SSL II, NUMPAC は Bunch さんの方法を用いたサブルーチンをサポートしています ([69], [71])。

3.3.7 越境法

越境法 (*below-the-line method*)は、方程式系 $Ax = b$ を拡張して計算を行う方法です。

A を拡張した行列 M を

$$M = \begin{pmatrix} A & K \\ H & 0 \end{pmatrix}$$

で定義します。この後は 2 つの方法に分かれます。

Crout の越境法

M の n 行 n 列まで Crout 法を適用して A を LDU 分解する方法です。すると、 H と K の形をうまく決めることで、右辺ベクトル b を変化して作った行列 B に対して $A^{-1}B$ が求まります。

が、はっきり言って記憶領域の無駄使いです。

Aitken の越境法

こちらは拡張行列 M の A と H の行に Gauss の消去法を適用します。 H と K の形により、拡張行列 M の右下に結果として A^{-1} や $A^{-1}B$ を作り出すのが目的です。

名前は Alexander Craig Aitken さん (1895–1967) に由来します。興味のある方は [34] を参照下さい。

3.4 対角化に基礎をおく方法

“任意の行列は行と列に有限回の基本操作をほどこすことによって、同値な対角行列に変形できる” という線形代数の基本的な定理にもとづいた方法です。

3.4.1 Gauss-Jordan の消去法

Gauss の消去法に似た方法ですが、対角要素の上の成分も消去することで、最終的に

$$\begin{pmatrix} \hat{a}_{11} & & & 0 \\ & \hat{a}_{22} & & \\ & & \ddots & \\ 0 & & & \hat{a}_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \hat{b}_1 \\ \hat{b}_2 \\ \vdots \\ \hat{b}_n \end{pmatrix}$$

の形まで持っていきます。

結果として、 Gauss の消去法のように後退代入をやる必要がありません ([51])。

Gauss-Jordan[40]の消去法は、掃き出し法 (*sweep-out method*)ともいわれ、昔は盛んに使われていましたが、演算回数が Gauss の消去法の 1.5 倍もかかることから、次第に使われなくなりました[41]。

ところが、行消去の演算が 1 行目から n 行目まで「せーの」で、しかも独立に行われるという性格から、並列計算機向きの解法ではないかと指摘され、最近にわかに注目されつつあります ([60])。

3.4.2 逆行列の積表現

積行列法ともいわれます。 Gauss-Jordan の消去法に関連した手法で、 “これとこれをかけあわせれば A^{-1} になる” という乗数表を作り、精度と記憶場所を節約しようという方法です ([34])。

3.4.3 合同変換による方法

A が対称行列ならば、ある直交行列 R によって

$$D = R^T AR$$

と対角化できる、という性質を用いて A^{-1} を生成する方法です。しかし、一般に A が正定値でないとうまくいきません ([34]) し、連立 1 次方程式の解が欲しい場合ならば他の方法が有利です。

3.5 直交化法

直交化法 (*orthoginal method*) は LU 分解法などに比べて演算回数がかなりかかるため、連立 1 次方程式の数値解法で実際に用いられることはそんなにありません。

ただし、行列の性質が悪い場合や、他の解法の安定性が信用できないときに、精度の比較で引っ張り出されることがあります。

3.5.1 直交ベクトル法

英語で書くと “method of orthogonal vectors” です。 A が対称なとき、 A に直交するベクトルを順次作ることで

$$LAL^T = D$$

の形に分解します。 L は下三角な直交行列です。これより

$$A^{-1} = L^T D^{-1} L$$

が簡単にわかり、従って x が計算できます。この方法は一般の行列にも拡張できます ([34])。

3.5.2 直交行列法

直交行列法 (*method of matrix orthogonalization*) は、固有値の代表的な計算手順である QR 分解から導かれます。

QR 分解

> A が正則ならば, 直交行列 Q と上三角行列 R が存在して
>
> $$A = QR$$
>
> の形に分解できる. また, 分解は R の対角成分をすべて正になるように定めれば一意である.

証明は [51] を御覧下さい。 QR 分解で方程式は

$$QRx = b$$

となるので、まず

$$Qy = b$$

を解きます。 Q は直交行列なので転置すれば逆行列になり、

$$y = Q^T b$$

で求まります。次に R は上三角行列なので、

$$Rx = y$$

[40] “Jordan” は、有名な数学者 Camille Jordan さん (1838–1922) でなく、ドイツの測地学者 Wilhelm Jordan さん (1842–1899) の名前からきています。従ってドイツ語読みをすれば「ヨルダン」が正しいそうです ([72])。

[41] 最近の数値計算の本で紹介されないことすらあります。この世界では速いことが絶対の正義なのです。

を後退代入でスラスラ解けば終りです。

QR 分解を行なう主な算法は

- 修正 Gram-Schmidt 直交化法[42]
- Householder 変換による方法[43]
- Givens 変換による方法[44]

の 3 つです。これらは、どちらかといえば固有値計算や最小自乗法で大きい顔をする手法ですので、これ以上の深入りは避けます。詳しくは [61] を御覧下さい。また、 Householder さんの簡潔にまとまった原論文 [19] もおすすめです。

3.6 分割法

分割法 (*partitioning method*) はブロック分解法ともいい、行列が記憶領域に入りきれない場合などに使います。プログラム演習の題材にできそうな方法ばかりです。詳細は [34] をお読み下さい[45]。

3.6.1 分割計算法

正則な実行列の逆行列を、低い次数の逆行列を用いて計算する方法です。計算時間の節約には全くなりません。

3.6.2 縁どり法

縁どり法 (bordering method)は分割計算法の変形で、 $n-1$ 次の行列 A_{n-1} の逆行列がわかっている場合、 A_{n-1} に 1 つの行と列をつけ加えた (縁どりした) n 次の行列 A_n の逆行列を求める手法です。

3.6.3 エスカレータ法

エスカレータ法 (*escalator method*) は、 J. Morris さんが 1946 年に発表した方法です。

k を行列の次数とすると、低い次数から順番に

$$A_k x_k = b_k \qquad k = 1, 2, \cdots, n$$

として方程式を解いていく方法です。

[42] J. P. Gram さん (1850–1916) と E. Schmidt さん (1876–1959) の名前から来ています。

[43] A. S. Householder さん (1904–?) が考案した方法です。

[44] J. W. Givens Jr. さん (1910–?) が考案した方法です。

[45] 速さや精度はともかく、これらの連立 1 次方程式の数値解法が提案された背景や数学的な証明を丁寧にフォローし、プログラムを組んで性能や問題点を調べるのは、結構面白いと思います。

3.6.4 階数零化法

階数零化法 (*rank annihilation method*) は、 H. S. Wilf さんが 1959 年に発表した方法で、逆行列が既にわかっている B とベクトル u, v を用いて $A = B + uv^T$ と書けるとき、 A の逆行列を

$$A^{-1} = B^{-1} - \frac{(B^{-1}u)(v^T B^{-1})}{1 + v^T B^{-1} u}$$

で求める方法です。逆行列の値が必要な場合は使えるかもしれませんが、連立 1 次方程式の解だけが欲しい場合には、計算時間の関係からあまりお勧めできません。

3.7 三重対角行列専用の方法

偏微分方程式を差分法で解いたりすると、しばしば三重対角行列が現れます。 A が三重対角行列の場合、その性質を存分に生かした専用の手法があります。

3.7.1 Gauss の消去法, Cholesky 法

一般の Gauss の消去法, Cholesky 法を三重対角行列向きに書き直した方法です。

通常の問題が $O(n^3)$ の計算時間を要するのに対し、三重対角の性質を使えば $O(n)$ ですみます。

3.7.2 cyclic reduction 法

もとの方程式に適当な消去処理をすることによって、元数が半分の三重対角方程式に縮小 (reduction) させます。この方程式を更に半分に縮小することを繰り返し (cyclic)、最終的に次数を 1 次まで落してしまって得られる解を逆にたどり、すべての解を求める方法です。

縮小処理はすべて並列に処理できるので、ベクトル計算機、並列計算機にとても適した手法です ([70])。

3.8 疎行列, 帯行列専用の方法

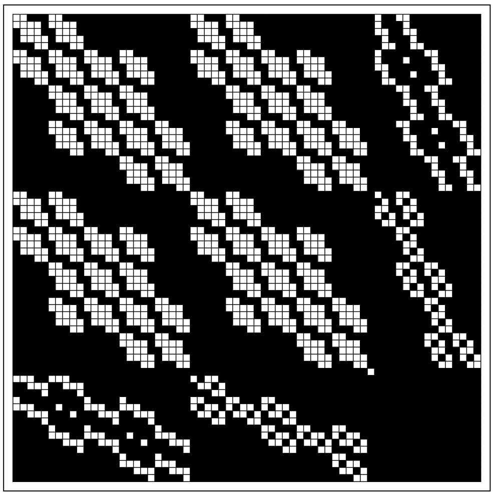
直接法の最大の弱点は、もとの行列 A を何らかの方法で変形するため、もともと 0 だったところがどんどん汚されていく点です。

そのため、大次元の行列の場合には、記憶領域の確保の問題が生じます。

LU 分解による行列の変化

次ページの図は、 2 次元正方領域で定常 Navier-Stokes 方程式の近似解を有限要素法で求めたときの

行列の分布図です[46]。□は成分が 0 でない場所を表します。■は成分が 0 のところです。有限要素法の分割を細かくすると、行列はどんどん “疎” の度合を増します。つまり■が多くなります。



LU 分解前の行列の形

この行列 A に Gauss の消去法による LU 分解をかけます。 L は A の下三角部分に、 U は上三角部分にそれぞれ格納されます。

LU 分解後の行列の形

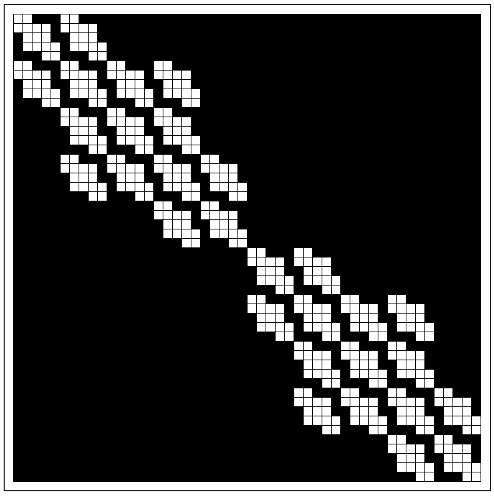
LU 分解の結果、□の部分がほとんどを占めるようになりました。このため、□の部分に対応した記憶場所を計算機内に確保しておく必要があります。

大規模な行列に何も考えずに直接法を適用すると、領域確保の問題で悩むことになります。

しかし、 A が帯行列ならば話が別です。次の図は、先ほどの LU 分解前の図から、対角付近の一部を抜き出した行列です[47]。

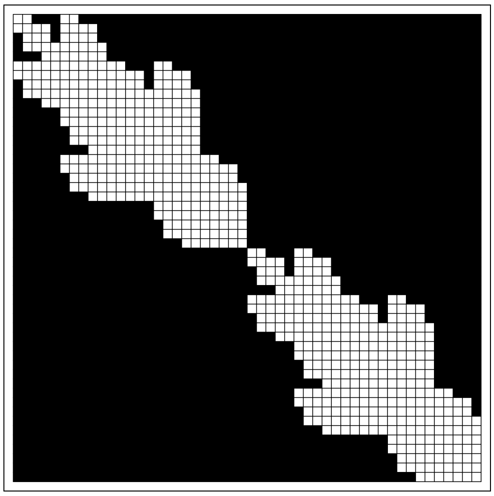
[46]同次境界条件で領域は四角形で等分割しています。近似は、流れ場が区分 2 次、圧力場が区分 1 次です。 Graphics は *Mathematica* の `ListDensityPlot` を使いました。

[47]Laplace 作用素が 2 つ並んでいるみたいなものです。



LU 分解前の行列の形 (帯行列)

この帯行列に Gauss の消去法による LU 分解をかけます。



LU 分解後の行列の形 (帯行列)

帯行列の場合、 Gauss の消去法の過程で、対角線からある程度離れた成分 (帯の外側) は、ずっと 0 のまま変化しないことがわかります。

この性質を生かすと、計算時間と記憶場所を節約する帯行列専用のプログラムが書けます。アルゴリズムは [60] に詳しくのっていますので参照下さい。

A が疎行列の場合も、 wavefront 法やスカイライン法、 DM 分解に基づく方法など、行列の特殊性を生かした解法が開発されています。詳しくは [6], [51], [59], [60] などを参照下さい。

3.9 現在人気の解法

LINPACK, LAPACK, IMSL などの著名な数値計算プログラム・パッケージの多くは、直接法のサブルーチンを主にサポートしています。

行列の形	SSL II	NUMPAC
一般の行列	Gauss の消去法での LU 分解 Crout 法での LU 分解 Householder 変換による QR 分解 特異値分解法による最小二乗最小ノルム解	Gauss の消去法での LU 分解 Doolittle 法での LU 分解 Householder 変換による QR 分解 特異値分解法による最小二乗最小ノルム解
対称	Bunch の方法での MDM^T 分解	Bunch の方法での MDM^T 分解
帯	Gauss の消去法での LU 分解	Gauss の消去法での LU 分解
三重対角	Gauss の消去法での LU 分解 cyclic reduction 法	Gauss の消去法での LU 分解
対称・帯	Bunch の方法での MDM^T 分解	Bunch の方法での MDM^T 分解
正定値・対称	修正 Cholesky 法での LDL^T 分解	Cholesky 法での LL^T 分解 (修正)Cholesky 法での $LL^T(LDL^T)$ 分解
正定値・対称・帯	修正 Cholesky 法での LDL^T 分解	(修正)Cholesky 法での $LL^T(LDL^T)$ 分解
正定値・対称・三重対角	修正 Cholesky 法での LDL^T 分解	Cholesky 法での LL^T 分解

SSL II, NUMPAC が採用している連立 1 次方程式の直接解法 (1995 年現在)

その理由は、直接法が他の手法に比べて優れているからではなくて、ある最低の条件を満たせばサクサクと答を出してくれる “汎用性” にあります。

反復法や共役勾配法は、解こうとする近似モデルに依存して収束が速かったり遅かったりするので、個別撃破[48]が要求される場合が多く、プログラム・パッケージとしてまとめるのは大変です[49]。

上の表は、九州大学大型計算機センターで現在公開している数値計算ライブラリの SSL II, NUMPAC の中から、実係数の連立 1 次方程式に関する直接解法をまとめたものです。

行列の形に対応していろいろな算法が提案されていますが、やはり LU 分解などの “三角化に基礎をおく方法” が圧倒的です。

この中でどれを選ぶかは (行列の形が第一ですが)、精度が極端に悪くなることを考えなければ、

速い、または、記憶領域が節約できる

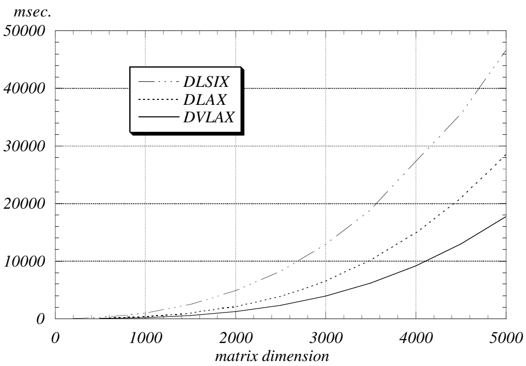
ことが基準になります。

例えば、対称行列専用のサブルーチンは、一般の行列用のサブルーチンに比べ、半分の記憶領域 (行列について) で実行できます。だからといって、実行時間が半分になるとは限りません。

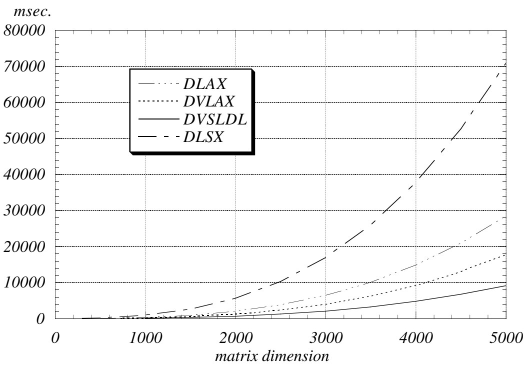
ベクトル計算機 FUJITSU VP2600/10 で SSL II のサブルーチンを用いて実行速度を調べてみます。DLSIX は対称行列専用、 DLAX, DVLAX は一般行列用で、 DVLAX はベクトル計算機向きにチューニングされています。

[48]要するに自分でプログラムを書け、ということです。

[49]もっとも最近は、反復解法専用のプログラム・パッケージも出回るようになりました。



なんと、記憶領域節約型の DLSIX が一番遅くなっています。次に対称正定値の場合を調べましょう。DLSX, DVLSX は対称正定値専用のサブルーチンです。 DVLSX はベクトル計算機向きにチューニングされています。



DVLSX は期待通りの性能を見せましたが、 DLSX はダントツの最下位におわりました。

このことから、記憶領域の節約だけでなく、計算機の性能にあわせたチューニングも大切なことがわかります。

4　反復法の概要

反復法は、直接法と比べむやみに種類が多く、とても分類する気になりません。ここでは、よく知られた反復法に限定して紹介します。

「反復法」と名がついているからには、何かの式を反復しながら収束する近似解を作るんだろうということはすぐに察しがつきます。

反復法でもっともよく使われるのは線形定常反復法 (*linear stationary iterative method*) です。反復列の作成は以下の通りです。

線形定常反復法

適当な初期値 x^0 から出発して，

$$x^k = x^{k-1} + R(b - Ax^{k-1}), \quad k \geq 1 \qquad (9)$$

を気がすむまで繰り返す．ただし R は A^{-1} の近似であることが望ましい．

(9) の "$b - Ax^{k-1}$" は、x^{k-1} が解に近ければ小さくなることが期待される "残差" です。

実際、$R = A^{-1}$ であれば、一発で反復は収束します。もっとも、そんなことが最初からできるなら苦労はしません。

線形定常反復法では、「どんな R を選べばいいのか」が最大のポイントになります。

4.1　反復法の創始者

反復法の歴史を遡ると、おなじみ Carl Friedrich Gauss さん (1777–1855) までたどりつきます。

Gauss さんは 1823 年に弟子の Christian Ludwig Gerling さんにあてた手紙で、連立 1 次方程式を解くために反復法を使うことを推奨しています。

Gauss さんの考えた反復法は、次数が大きくなりすぎて消去法での手計算が難しくなった場合、方程式を 2 つ以上の系に分け、その系を順次解きながら近似解を生成し、望むまで繰り返すといった手法でした ([17])。Gauss さんは手紙の中にこんなことを書いています[50]：

> *You will hardly ever again eliminate directly, at least not when you have more than 2 unknowns. The indirect procedure can be done while one is half asleep, or while thinking about other things.*
>
> Gauss さんから Gerling さんへの手紙 (1823)

Gauss さんの反復手法は、手紙をもらった Gerling さんからドイツの数学者のサークルに広まり、その成果に興味を持った Jacobi さんと、その弟子の Seidel さんに引き継がれます[51]。

なお、反復法についてのさらに詳しい歴史は [2] を参照下さい。

4.2　Jacobi 法

Jacobi 法は、ドイツの数学者 Carl Gustav Jacob Jacobi さん[52](1804–1851) が 1845 年に発表したもので、(9) で $R = D^{-1}$ と選ぶ方法です。ただし、D は A の対角成分を要素とする対角行列です。

(9) を $R = D^{-1}$ として変形すると、Jacobi 法の反復列は

$$x^k = (I - D^{-1}A)x^{k-1} + D^{-1}b$$

となります。

Jacobi 法の収束については、以下のことが知られています ([60])。

Jacobi 法の収束条件

A が対角優位行列 ならば Jacobi 法は収束する．

[50]もちろん、原文はドイツ語です。英語訳は [8] から抜粋しました。

[51]Gauss-Seidel 法の由来はここからきています。もっとも、Forsythe さんによれば、現在 "Gauss-Seidel 法" とよばれる手順について Gauss さんが言及した形跡はどこにもなく ([8])、Sidel さんにいたってはこの方法についてふれたあと「この方法は使わない方がいい」と書いてあるそうです ([10])。

[52]何といっても理科系の学生を悩ませる "Jacobian" で広く知られています。強烈な性格だったらしく、人の反感を買うことが多かったそうです。晩年は財産を失い、健康を損ね、業績も振るわず、不遇のうちに亡くなりました ([73])。

微積分学の勉強の際、Jacobian でつまづく人が多いのはそのためかも知れません。

実際の計算において、 Jacobi 法は収束がかなり遅いため ([61])、収束速度を速める工夫が必要です。

なお、 Jacobi さんの連立 1 次方程式に対する取り組みが [17] に詳細にまとめられていますので、興味のある方はお読み下さい。

4.3 Richardson 法

Lewis Fry Richardson さん (1881–1953) は 1910 年、 (9) の特別な場合として、 R を反復によって変化するスカラー α_{k-1} とおき、この手順を最適化する α_{k-1} の決定方法を提案しました ([32])。

また、適当な実数 α で $R = \alpha A^T$ とおく手法も Richardson 法といいます。

4.4 Gauss-Seidel 法

Gauss-Seidel 法は、 1862 年に Philipp Ludwig von Seidel さん (1821–96) によって発表され、 1874 年にきちんとした形で著作となりました ([2])。

まず、行列 A を分離します。

$$A = L + D + U$$

L : A の下三角部分, ただし対角部分は 0
D : A の対角部分
U : A の上三角部分, ただし対角部分は 0

Gauss-Seidel 法は $R = (L + D)^{-1}$ と選びます。これを (9) に放り込むと、

$$x^k = D^{-1}(b - Lx^k - Ux^{k-1})$$

となります。 L が下三角行列なので、この式は、 x^k の i 番目を計算するとき、前回の x^{k-1} を使う代わりに、既に $i-1$ まで計算されている新しい x^k の要素を使用することを意味しています。

Gauss-Seidel 法の収束について、次のことが知られています ([54])。

Gauss-Seidel 法の収束条件

> A が対角優位行列、または対称かつ正定値行列ならば, Gauss-Seidel 法は収束する.

Gauss-Seidel 法は、 Jacobi 法に比べいくぶん収束性がよくなっていますが、まだまだ改善の余地があります ([32])[53]。

[53] 5.7 節に出てくる前処理技法は反復法にも適用でき、現在も盛んに研究されています。問題は並列化との相性です。

4.5 SOR 法

SOR 法 (*successive over-relaxation method*) は、過剰緩和法、逐次過緩和法ともいいます。

SOR 法は、 S. P. Frankel さんと David M. Young さんがほぼ同時に発表した方法で、 Gauss-Seidel 法の収束を加速させるため、加速係数 $1 < \omega < 2$ を用いて $R = (L + \omega^{-1}D)^{-1}$ とするものです。

これを (9) に放り込むと、 n

$$x^k = (D+\omega L)^{-1}(\omega b+((1-\omega)D-\omega U)x^{k-1}) \quad (10)$$

となります。 $\omega = 1$ のときが Gauss-Seidel 法です。

$\omega > 1$ のとき Gauss-Seidel 法に比べ "過剰" に修正されるため、過剰緩和法の名前がつきました。

(10) を書き直すと、次の反復と同じです。

$$\begin{aligned} y^k &= D^{-1}(b - Lx^k - Ux^{k-1}) \\ x^k &= x^{k-1} + \omega(y^k - x^{k-1}) \end{aligned} \quad (11)$$

SOR 法の収束については、 A. M. Ostrowski さんが 1954 年に示した有名な定理があります ([32])。

Ostrowski の定理

> A が正定値かつ対称で $D + \omega L$ が正則ならば, $0 < \omega < 2$ の範囲で SOR 法は収束する.

最適な ω となる理論値はわかっていますが、実際にどのように選んだら収束が加速されるかは多分にケースバイケースで、これに関して多くの研究がなされています。詳細は [32], [51], [61] を御覧下さい。

SOR 法の最大の利点は、プログラムが簡単に書けることで、現在も流体解析の分野を中心に根強い人気を保っています ([55])。

以下、 SOR 法の仲間を紹介します。

4.5.1 SLOR 法

SOR の反復式 (11) の計算では、 D^{-1} が必要です。しかし、 D は対角行列なので、逆行列は 1 を D の各要素で割ったものとして簡単に求まります。

では、 SOR 法において、この D の部分が三重対角行列になるように $A = L + D + U$ と分解したらどうなるでしょうか。ただし、分解された D は少なくとも正則でないとだめです。

式は (11) と同じです。違うのは、三重対角行列に関する連立 1 次方程式

$$D^{-1}(b - Lx^k - Ux^{k-1})$$

を直接解法で解く必要が生じるところです[54]。

この方法を **SLOR** 法 (*successive line over-relaxation method*[55]) または過剰線緩和法、逐次線過緩和法といいます。

SLOR 法は、直接法と反復法を組み合わせた方法で、収束性の向上、精度の安定などが報告され、1950 年代終りから 1960 年代にかけてアメリカで大いに使われました。

この成功に気を良くし、SLOR 法を更に拡張した[56]方法が **S2SOR** 法 (*two-line successive over-relaxation method*) です。

なお、詳細な解説が [32] にありますので、興味のある方は御覧下さい。

4.5.2 SSOR 法

SOR 法の反復行列は一般に非対称です。反復行列を対称に保つ方法が **SSOR** 法 (*symmetric SOR method*) です。

後で出てくる Chebyshev 加速法と組み合わせて収束性を高めると、なかなかの性能を発揮します。

また、よく似た方法に USSOR 法 (*un-symmetric SOR method*) があります ([51])。

4.5.3 SBOR 法

A を小行列のブロックに分割して、それぞれに SOR 法を適用する方法が **SBOR** 法 (*successive block over-relaxation method*) です。

係数行列が対角成分に集まっている場合に威力を発揮します ([51])。

4.6 ADI 法

ADI 法 (*Alternating Direction Implicit method*) は、D. W. Peaceman さんと H. H. Rachford Jr. さんが 1955 年に発表した方法で、交互方向法ともいいます。

ADI 法は、2 次元の Laplace 方程式、Poisson 方程式[57]、熱伝導方程式の陰解法などの差分法でよく用いられます ([53])。

A_1, A_2 を対称かつ正定値かつ対角要素が正でそれ以外が非正の行列、D を非負の対角行列とします。

もし、

$$A = A_1 + A_2 + D$$

と分解できるとき、適当な行列 E_1, E_2 をとれば、$Ax = b$ は

$$\begin{aligned}(A_1 + D + E_1)x &= b - (A_2 - E_1)x \\ (A_2 + D + E_2)x &= b - (A_1 - E_2)x\end{aligned}$$

のどちらの式とも同値です。

ADI 法は、x^k から一方の式を使い中間ベクトル $\hat{x}^k$ を求め、次にもう一方の式を使い x^{k+1} を作る方法です。

ポイントは、うまい具合に E_1, E_2 を選んで行列 $A_1 + D + E_1$, $A_2 + D + E_2$ を正定値で条件のいい三重対角行列にするところです。

条件の良い三重対角行列を係数行列とする連立 1 次方程式は、直接法でスラスラと解けます。このように、ADI 法[58]は直接解法を一部採り入れた反復解法です。

なお、[32] に 1 章まるまる割いた解説がありますので、詳しくはこちらを参照下さい。

4.7 Aitken の δ^2 法

Alexander Craig Aitken さん (1895–1967) は、数列 $\{x_n\}$ が極限に 1 次収束するとき、$\{x_n\}$ よりもさらに速く収束する数列 $\{y_n\}$ を作り出す方法を提案しました。この変換方法を **Aitken** の δ^2 法といいます。

$\{y_n\}$ の作り方はいろいろありますが、例えば

$$y_i = x_{i+2} - \frac{(x_{i+2} - x_{i+1})^2}{x_{i+2} - 2x_{i+1} + x_i}$$

などで $\{y_n\}$ を求めます。これで反復法の収束を加速させることを狙います ([34])。

4.8 Chebyshev 加速法

Chebyshev 加速法も、反復法の収束を速める有名な手法です。Chebyshev 準反復法ともいいます。

Chebyshev 加速法の基本方針は、"x^k の計算に直前の x^{k-1} しか使わないのはもったいないので、それより前の情報 $x^{k-2}, \cdots, x^0$ も活用しよう" というものです。

情報の使い方としては、反復列の一次結合

$$\hat{x}^k = \sum_{j=0}^{k} \alpha_j x^j$$

[54]厳密には直接法で解く必要はどこにもありません。が、直接法がいちばん速いでしょう。

[55]"線" の意味は、差分化した未知数の格子点の線を単位としてあつかうことに由来しています。

[56]D のバンド幅を大きくした

[57]Siméon Denis Poisson さん (1781–1840) に由来します。

[58]「交互方向法」という名称は、分割した格子を、まず水平方向に解き、次に垂直方向に解くところからきています。

に対し、$\hat{x}^k$ の精度がもっともよくなるような α_j を決定することを考えます。

この α_j は、k 次の Chebyshev[59] 多項式

$$T_k(t) = \begin{cases} \cos(k \arccos t), & |t| \le 1 \\ \cosh(k \arccos t), & |t| > 1 \end{cases}$$

の係数から導かれることが示せます ([61])。

この他、α_j の決め方には **CG** 加速法などもあります。

4.9 モンテカルロ法

モンテカルロ法 (*Monte Carlo method*) は、逆行列の要素または連立 1 次方程式の解の成分を、期待値が望み通りの解になるような統計的変数のランダム・サンプリングを行なうことにより、統計的に推定する方法です。

反復法の出発値としてのごく粗い推定値を得たい場合などに用います ([34])。

4.10 反復改良法

さてここで、(9) の行列 R は「A^{-1} であることが望ましい」という、下手な仕様書みたいな条件を思い出してみましょう。

浮動小数点演算の有界性より、直接法でもとめた A の LU 分解は、ある "誤差" を含んでいます。これを行列の形で

$$A = LU + E$$

と書きます。

LU 分解によって求めた近似解 $\hat{x}$ は、反復法の絶好の初期値といえます。また、LU の逆行列は、「A^{-1} であることが望ましい」という条件に適応した R の絶好の候補といえるでしょう。

そこで、直接法と反復法を組み合わせた反復改良法 (*iterative improvement*) が、初期値を $\hat{x}$、$R = (LU)^{-1}$ として次のように定義できます。

$$x^k = x^{k-1} + (LU)^{-1}(b - Ax^{k-1})$$

$(LU)^{-1}(b - Ax^{k-1})$ の計算は、$c = (b - Ax^{k-1})$ とすると、$LUx = c$ をみたす x を求めればいいので、前進代入・後退代入により求まります。

直接法によって求められた近似解の反復改良は LU 分解に限らず、LDU, LDL^T, LL^T, MDM^T, QR 分解などでも可能です。A の条件がものすごく悪くなければ、よりよい精度に改良されます ([60])。

また、R の決め方は自由ですので、A を LU 分解に都合のいい行列と、それ以外 E に積極的に分けて反復する方法もあります ([59])。

SSL II には、LU 分解で得られた近似解の反復改良サブルーチン DLAXR がサポートされています。Gauss の消去法での解法 DVLAX と組み合わせ、精度がどれくらい改善されるかを調査しました。計算機は VP2600/10、コンパイラは FORTRAN77 EX/VP V12L10 です。

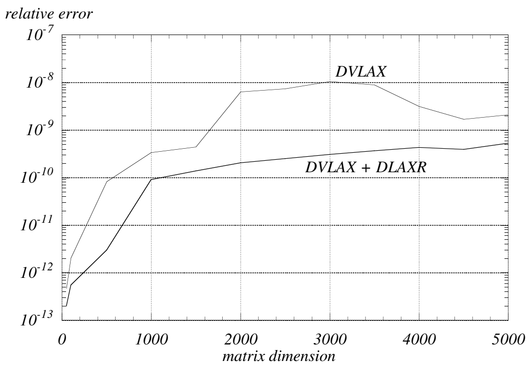
方程式は

$$\begin{cases} a_{ij} = a_{ji} = n + 1 - i \quad (\text{if} \ \ i \ge j) \\ b_i = \displaystyle\sum_{k=1}^{n-i+1} \frac{n-k+1}{n} \end{cases}$$

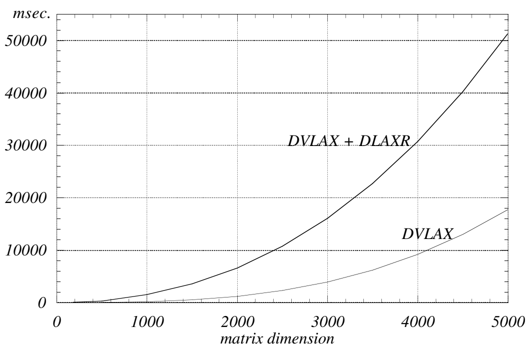
で定義します。解は正確に $x = (1/n, \cdots, 1/n)$ となります。なお、誤差は相対誤差

$$\max_{1 \le i \le n} |nx_i - 1|$$

で評価しました。



反復改良により、約 1 桁精度が改良されています。一方、実行時間は下の通りです。



反復改良を追加すると、約 3 倍の実行時間がかかります。より精度のいい近似解が欲しいならば、反復改良法を使うべきですが、その分コストがかかることを覚悟すべきでしょう。

[59] Pafnutiĭ L'vovich Chebyshev さん (1821–1894) の名前からきた有名な多項式です。

5　共役勾配法の概要

共役勾配法の系統は、略字だらけで大いに混乱しますので、章の終りに一覧表をつけました。

5.1　鮮烈なデビューとその後

共役勾配法[60]は、M. R. Hestenes さんと E. Stiefel さんによって 1952 年に発表されました。

当時は電子計算機のための数値計算技術の確立が急務とされている時期で、共役勾配法は、反復法でありながら有限回のステップで厳密解に到達するという性質と巧妙なアイディアが熱狂的に支持され、「科学史上に残る大発明」として華やかなデビューを飾りました (当時の経緯は [52] に詳しく書いてあります)。

しかし、いざ共役勾配法を実際の問題に適用しようとすると、丸め誤差の影響を受けやすく、何回反復してもいっこうに収束しない例も出てきました。

また、計算機の大容量化にあわせ、ライバルの直接法と SOR 法[61]が理論と技法の両面から頑張ったこともあり、華々しくデビューしたわりには、当時期待したほど普及しませんでした。

1960 年代の数値計算の本を見ると、連立 1 次方程式の解法として Gauss の消去法、Gauss-Jordan 法、Jacobi 法、Gauss-Seidel 法、SOR 法は必ず紹介されていますが、共役勾配法をきちんと扱った本はそれほど多くありません。しばらく不遇の時代が続きます。

共役勾配法は、1960 年代に非線形最適化問題の解法として再登場し、1970 年代には疎行列用の反復解法としての研究が行われました。そして 1980 年代に入ると、“前処理”という新しい手法を付け加えることで、大規模行列に対する反復法として再び脚光をあびるようになりました。

Hestenes さんと Stiefel さんも喜んでいることでしょう[62]。

5.2　基本的な考え方

共役勾配法の考え方は、次の性質が出発点です。

最小化問題

A が正定値かつ対称のとき, $Ax = b$ を解くことは,

$$f(x) = \frac{1}{2}(x, Ax) - (x, b)$$

を最小にする x を求めることと同値.

$(\cdot\,,\cdot)$ はベクトル内積

$$(x, y) = x^T y = \sum_{i=1}^{n} x_i y_i$$

として定義します。

同値性を確認しておきましょう。f を微分すれば

$$\nabla f(x) = \begin{pmatrix} \dfrac{\partial f}{\partial x_1} \\ \vdots \\ \dfrac{\partial f}{\partial x_n} \end{pmatrix} = Ax - b$$

となります[63]。

$f(x)$ の最小点 x^* が見つかったとすると、$\nabla f(x)$ での傾きはなくなるので、x^* での f の微分は 0 となり、つまり $Ax^* - b = 0$ となります。

また、任意のベクトル $h \neq 0$ に対し、A の対称性を使って

$$f(x+h) = f(x) + (h, Ax - b) + \frac{1}{2}(h, Ah)$$

[60]「共役」は「きょうやく」と読みます。もともとは軛 (くびき) をともにして車を引く意味の「共軛」からきています。昔、数学の講義で、とある教授が「これを《きょうえき》と読むのは数学が全くわかっとらん証拠じゃ～」とおっしゃっていたのを記憶しています。数学者の前で発表するときは気をつけましょう。

[61]一般論ですが、「理論」や「手法」などは、科学とか論理の土俵をはみ出して、政治的学閥的立場、個人的好き嫌い、宗教観などで争われることがあります。しかし、動機はともあれ、結果がみんなのためになればそれでいいのです。

[62]よく歌や映画にある筋書きで、売れない歌手の熱狂的なファンだった女の子が、華やかなスポットライトを浴びてどんどんメジャーになっていく彼を悲しそうに見つめながら離れていくというのがあります。1960 年代に共役勾配法を一所懸命に研究した人も、今はそんな気持ちなのでしょうか。

[63]ピンとこない方は、$f(x)$ を全て成分の形に書き下して、一つ一つ偏微分してみて下さい。

となることが簡単にわかります。

ここで、x^* が $Ax^* = b$ をみたすと考えれば、A は正定値でしたので $(h, Ah) > 0$ となって[64]、

$$f(x^* + h) > f(x^*)$$

となります。つまり、x^* から少しでもはずれると、$f(x)$ の値は $f(x^*)$ より大きくなり、このことから x^* が f の最小値を与えることがわかります。

具体的な例をあげましょう。

$$A = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

とすると、A は対称かつ正定値で、$Ax = b$ の解は $x = (1,1)^T$ です。

また、$f(x)$ は

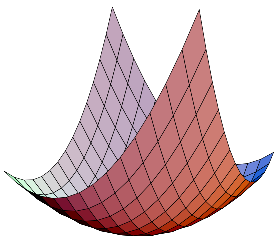
$$f(x) = x_1^2 + x_2^2 - x_1x_2 - x_1 - x_2$$

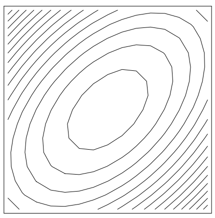
となります。f をごちゃごちゃ変形すると

$$f(x) = \frac{(x_1-1)^2 + (x_2-1)^2 + (x_1-x_2)^2}{2} - 1$$

ですので、最小値は $x_1 = x_2 = 1$ のとき -1 となることがすぐにわかります。

$f(x_1, x_2)$ の $(1,1)^T$ のまわりの形を書くと以下のようになります。





$f(x)$ の形状と等高線図

この図で最小値となる $x = (1,1)^T$ が $Ax = b$ の解です[65]。

5.3 計算の手順

共役勾配法とは、適当な初期値を出発し、順番に探索方向を定め、その方向に沿って最小値を探しに行く手法です。

具体的な手順は次の通りです。

- 初期値 x^0 と探索方向 p^0 を決める。
- $f(x^0 + \alpha p^0)$ が最小になるようなスカラー α を計算し、それを α^0 とする。
- 近似解を $x^1 = x^0 + \alpha^0 p^0$ とする。
- 次の探索方向 p^1 を決める。
- 以下、気が済むまで繰り返す。

最小値が欲しい α は、2 次式の最小化と同じになるので、簡単に求めることができます ([61])。

重要なのは、探索方向となるベクトル p^k を如何に上手に決定するかです。この選び方によって、収束率が決定的に変わってきます。

5.4 最急降下法

最急降下法 (*steepest descent method*) は、探索方向 p^k を x^k における勾配方向

$$p^k = -\nabla f(x^k) = (\frac{\partial f(x^k)}{\partial x^1}, \cdots \frac{\partial f(x^k)}{\partial x^n})^T$$

にとる手法です。**SD** 法ともいいます。

最急降下法は、A の性質が悪い場合、収束がかなり遅くなることから、あまり使われません。

5.5 共役方向法

共役方向法 (*conjugate direction method*) は、さらに上手な p^k の設定方法です。**CD** 法ともいいます。

探索方向の選び方は、これまでの探索方法と “共役” となる方向にとっていきます。“共役” とはこの場合、A に関して直交する方向を指します。ちゃんと書くと、

共役の定義

ベクトル x と y に対し

$$(x, Ay) = (y, Ax) = 0$$

が成り立つとき, x と y は A に関して共役である.

となります。

つまり k 番目までの方向 p^k が決まっているとき、$k+1$ 番目の方向 p^{k+1} は、その全てと共役になるように、すなわち

$$(p^{k+1}, Ap^i) = 0, \qquad i = 0, \cdots, k$$

[64] 正定値の定義より $h \neq 0$ に対し $h^T Ah > 0$ です。

[65] 連立 1 次方程式を同値な最小問題に持ち込むこの方法は、共役勾配法に代表されるような巧妙な理論を生みました。しかし、直接法や反復法に比べて最小化する近似解に「副産物」的な印象を受けるためか、敬遠する人もいます。

で決めます。

具体的な方法は [52], [61] を参照下さい。

探索方向 p^k を共役方向に定めるということは、理論的に大変重要です。それは、次の定理が成り立つからです。

共役方向法の収束定理

A が対称かつ正定値ならば, 共役方向法は高々 n 回の反復で厳密解に到達する.

この、有限回の反復で厳密な解が得られるという理論的保証が、共役方向法の最大の魅力です。

5.6　共役勾配法

真打ち共役勾配法 (*conjugate gradient method*) は、共役方向法の一種で、次のように探索方向 p^k を決定する手順を指します：

最急降下法で選んだ探索ベクトル p^k は

$$-\nabla f(x^k) = b - Ax^k$$

でした。$b - Ax^k$ は、方程式の k 回目の残差と考えられます。この残差ベクトルを

$$r^k = b - Ax^k$$

とおきます。

共役勾配法の探索方向 p^k は、この勾配方向 r^k から Ap^{k-1} の成分を除去し $(p^k, Ap^{k-1}) = 0$ となるように決めます。つまり、

$$p^k = r^k + \beta^{k-1} p^{k-1}$$

$$\beta^{k-1} = -\frac{(r^k, Ap^{k-1})}{(p^{k-1}, Ap^{k-1})}$$

となるように決めます[66]。

このようにして選んだ p^k には共役直交性

$$(p^i, Ap^j) = 0, \qquad i \neq j$$

が、また r^k には直交性

$$(r^i, r^j) = 0, \qquad i \neq j$$

が帰納法を使って示せます ([54])。

これにより n 回反復すれば $r^n = 0$ となる、すなわち x^n が $Ax = b$ の解になることがわかります。その理由は、

$n+1$ 個の 0 でない n 次元ベクトルが互いに直交することはありえない。

からです。

従って、共役勾配法は高々 n 回の反復で厳密解に収束します。

といっても、これは理論上の話です。実際の計算では丸め誤差が混入するため、必ずしも n 回の反復で解が求まるとは限りません。

共役勾配法は共役傾斜法、**CG** 法ともいいます。

5.7　前処理付き共役勾配法

前処理付き共役勾配法 (*preconditioned conjugate gradient method*) は、**PCG** 法ともいいます。

手法は、適当な正則行列 C によって $Ax = b$ を

$$(C^{-1}AC^{-T})(C^T x) = C^{-1}b \qquad (12)$$

と同値変形し[67]、(12) に対し共役勾配法を適用するものです。

(12) の処理を前処理(*preconditioning*)、行列 C を前処理行列(*preconditioner*)[68]といいます。

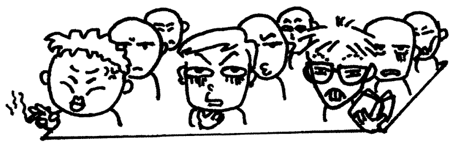
前処理をする理由

共役勾配法の収束性は、解を構成する固有ベクトルに対応した固有値の分布によって大きく変わります。

係数行列の固有値が重複したり、狭い範囲に密集していれば、収束が速くなることが知られています ([56])。つまり、

前処理は、行列の固有値分布を改善し、収束性を向上することを目的

とした手法です。



扱いにくそうな行列

⇓

前処理によって改善された行列

[66] 本当に直交するかどうか計算してみればわかります。

[67] C^{-T} は $(C^T)^{-1} = (C^{-1})^T$ の意味です。

[68] お肌や髪の状態を整えるためにも使うそうな…

前処理付き共役勾配法は、1980 年以降、偏微分方程式の離散近似から導かれる連立 1 次方程式の解法に用いると効率的な近似解が得られることから、盛んに用いられるようになりました。

が、前処理技法の将来には並列計算機が立ち塞がります。どうやら、前処理技法と並列化技法はあまり相性がよくないようです。

5.7.1 ICCG 法

前処理行列の選び方は、個々の問題によっていろいろ考えることができます。

その中でも、とくに一般的な効果が期待できる手法に不完全 **Cholesky** 分解 (*incomplete Cholesky decomposition*) があります。

これは、疎で正値対称な行列 A を、"疎" の性質をあまり失わないように、適当な下三角行列 L と残差行列 E で

$$A = LL^T + E$$

または

$$A = \hat{L}\hat{D}\hat{L}^T + E, \quad L = \hat{L}D^{1/2}$$

と分解し、L を前処理行列 C とする手法です。

まともな Cholesky 分解、修正 Cholesky 分解と違い、あらかじめ指定した成分以外は 0 とし、それ以外は誤差行列 E に押しつけるという身勝手さのため[69]、"不完全 (incomplete)" と呼ばれます[70]。

特に LL^T, LDL^T 分解した行列から (12) を構成する方法を **ICCG** 法 (*incomplete Cholesky conjugate gradient method*) 、または不完全 **Cholesky** 分解付き共役勾配法 といいます。

ICCG 法は 1977 年に J. A. Meijerink さんと H. A. van der Vorst さんによって発表され ([22])、物理・工学などの分野で広く用いられるようになりました。

5.7.2 MICCG 法

このあたりから記号が錯綜してきます。

MICCG 法 (*modified ICCG method*) は、行列 A の対角要素に重みづけをし、修正項を付加して ICCG 法の加速を狙ったものです。

この行列の分解方法を修正不完全 **Cholesky** 分解 (*modified incomplete Cholesky decomposition*) といいます。

[69] 仕事でこんなことをやったら、間違いなく嫌われます。

[70] Cholesky さんの名前は、数値解析の世界で知らない人はいないくらい有名になりましたが、「修正」「変形」「不完全」など、ちょっと気の毒な言葉が頭にくっつきます。

多くの問題で ICCG 法よりも収束が速くなることが報告されています ([59])。

5.8 共役残差法

共役勾配法は A が対称かつ正定値行列であることを前提とした方法でした。

近年、共役勾配法を非対称行列へと拡張する手法が数多く提案され、各分野で大きな成果をあげています。以下、名前だけ簡単に紹介します。

共役残差法 (*conjugate residual method*) は、最小化する関数を残差ベクトルの平方和

$$f(x) = \frac{1}{2}(Ax - b, Ax - b)$$

として、今度は

$$(Ap^{k+1}, Ap^k) = 0$$

となるように探索方向 p^k を決定する方法です。

共役残差法は **CR** 法ともいいます。詳しくは [59], [61] を御覧下さい。

5.8.1 ILUCR 法

共役残差法にも当然前処理を施すことを考えます。前処理付き共役残差法は、頭に "preconditioned" がついて **PCR** 法ともいいます。

今度は A が対称でないので、Cholesky さんの出番はなく、LU 分解の登場です。

非対称行列 A を、"ぎっちり" LU 分解しないで、都合のいい下三角行列 L と上三角行列 U を使って

$$A = LU + E$$

と分解します。E は誤差行列です。

この分解を不完全 **LU** 分解 (*incomplete LU decomposition*) といいます。

ここで、

$$(LU)^{-1}Ax = (LU)^{-1}b$$

に対し共役残差法を適用します[71]。

この手法を **ILUCR** 法 (*incomplete LU conjugate residual method*)、または不完全 LU 分解付き共役残差法と呼びます。

[71] A が対称の場合は対称性を保つために $C^{-1}A(C^{-1})^T$ とする必要がありましたが、この場合 A はもともと非対称なので、左から $(LU)^{-1}$ をかけるだけです。

5.8.2 MILUCR 法

MILUCR 法 (*modified ILUCR method*) は、共役勾配法のときの MICCG 法と同じく、行列 A の対角要素を重みづけることで修正項を付加し、ILUCR 法の加速を狙ったものです。

この分解を修正不完全 **LU** 分解 (*modified incomplete LU decomposition*) といいます。

5.9 GMRES 法

行列 A が対称であっても必ずしも正定値でない場合、Lanczos 法と共役勾配法との関係をうまく利用した方法に **MINRES** 法 (*minimal residual method*) という方法があります[72]。

この方法を非対称行列に拡張した方法が **GMRES** 法 (*generalized minimal residual method*) です。具体的な手法は、最初に提案された論文 [26] を御覧下さい[73]。

5.10 双対共役勾配法

双対共役勾配法 (*biconjugate gradient method*) は、対称でない行列 A に対し、対称行列用の共役勾配法を修正して適用するもので、**BCG** 法または **BiCG** 法ともいいます。

原理は、

$$(r^i, \hat{r}^j) = 0, \quad (Ap^i, \hat{p}^j) = 0 \qquad i \neq j$$

$$(r^i, \hat{r}^i) \neq 0, \quad (Ap^i, \hat{p}^i) \neq 0 \qquad i = j$$

となるベクトル列 $\{p^k\}$, $\{r^k\}$ と $\{\hat{p}^k\}$, $\{\hat{r}^k\}$ を反復過程で作ることにあります。

なお、詳細は [59] を御覧下さい。

双対共役勾配法の変種に自乗共役勾配法 (conjugate gradient squared method) があります。自乗共役勾配法は **CGS** 法ともいいます。

BCG 法と CGS 法は本質的には同じものですが、BCG 法よりも CGS 法の方が収束が速いことが報告されています ([55])。

この二つに前処理を施すと、先頭に “preconditioned” がついて、**PBCG** 法、**PCGS** 法になります。

[72]もう少し具体的に言うと、A の固有値問題を解く方法でよく知られている Lanczos 法 ([72]) は、$v_1 = r_0/||r_0||$ としたとき Krylov 空間 $K_k = \mathrm{span}\{v_1, Av_1, \cdots, A^{k-1}v_1\}$ 上の Galerkin 法であり、この方法は $Ax = b$ を解く共役勾配法に対応している ([26]) という関係です。

[73]その他 FGMRES 法, GMRESR 法などがあるそうですが、ちゃんと調べてません。

5.10.1 ILUBCG 法

ILUBCG 法 (*incomplete LU BCG method*) は、不完全 *LU* 分解で前処理を施した双対共役勾配法です。

同様に自乗共役勾配法に対する **ILUCGS** 法 (*incomplete LU CGS method*) もあります。

5.10.2 MILUBCG 法

修正不完全 LU 分解によって ILUBCG 法の加速を狙ったのが **MILUBCG** 法 (*modified ILUBCG method*) です。

ILCGS 法の改良版の **MILUCGS** 法 (*modified ILUCGS method*) もあります。

5.11 QMR 法

双対共役勾配法 (BCG 法, BiCG 法) は演算の途中でゼロ割りが発生して計算が破綻 (breakdown) したり、数値的に不安定となる可能性があります。QMR 法 (*quasi-minimal residual method*)は、残差ノルムの最小化により双対共役勾配法の不安定性を克服し、高い収束性を狙った手法です。詳細は [15] を御覧下さい。

5.12 BiCGStab 法

自乗共役勾配法 (CGS 法) の収束の不規則性を改善する目的で提案された手法に **BiCGStab** 法があります。

“BiCGStab 法” は無理に展開すれば “biconjugate gradient stabilized method” でしょうが、最初に提案した H. A. van der Vorst さんによれば “CGS 法によく似ていて、安定性 (stability) が得られ、しかも BiCG 法とも関係するので、BiCGStab と名付けた” そうです。

BiCGStab 法にも前処理が適用できて、van der Vorst さんは **BiCGStab-P** と命名しています。詳細は [31] をお読み下さい。

以上、共役勾配法に属する手法を用語の意味だけ概観しました。ここで紹介したものは、雑誌や論文でよく見かけるもののほんの一部です[74]。

特に非対称行列に対する手法は、現在盛んに研究が行なわれており、新しい手法が次々に提案されています。

[74]残念ながら、共役勾配法系統で現在 (1995 年 11 月) センターで公開しているサブルーチンライブラリはありません。

“M” は “modified” の M，“IC” は “imcomplete Cholesky” の “IC”，“ILU” は “imcomplete LU” の “ILU”，“P” は “preconditioned” の “P” だと覚えておけば、共役勾配法関連のややこしい略語が整理できます。

略語	英語名	日本名
SD 法	steepest descent method	最急降下法
CD 法	conjugate direction method	共役方向法
CG 法	conjugate gradient method	共役勾配法
PCG 法	preconditioned conjugate gradient method	前処理付き共役勾配法
ICCG 法	imcomplete Cholesky conjugate gradient method	不完全 Cholesky 分解付き共役勾配法
MICCG 法	modified imcomplete Cholesky conjugate gradient method	修正不完全 Cholesky 分解付き共役勾配法
CR 法	conjugate residual method	共役残差法
PCR 法	preconditioned conjugate residual method	前処理付き共役残差法
ILUCR 法	incomplete LU conjugate residual method	不完全 LU 分解付き共役残差法
MILUCR 法	modified incomplete LU conjugate residual method	修正不完全 LU 分解付き共役残差法
BCG 法	biconjugate gradient method	双対共役勾配法
PBCG 法	preconditioned biconjugate gradient method	前処理付き双対共役勾配法
ILUBCG 法	incomplete LU biconjugate gradient method	不完全 LU 分解付き双対共役勾配法
MILUBCG 法	modified incomplete LU biconjugate gradient method	修正不完全 LU 分解付き双対共役勾配法
CGS 法	conjugate gradient squared method	自乗共役勾配法
PCGS 法	preconditioned conjugate gradient squared method	前処理付き自乗共役勾配法
ILUCGS 法	incomplete LU conjugate gradient squared method	不完全 LU 分解付き自乗共役勾配法
MILUCGS 法	modified incomplete LU conjugate gradient squared method	修正不完全 LU 分解付き自乗共役勾配法
QMR 法	quasi-minimal residual method	準最小残差法 *
MINRES 法	minimal residual method	最小残差法 *
GMRES 法	generalized minimal residual method	一般化最小残差法 *
BiCGStab 法	biconjugate gradient stabilized method*	双対共役勾配安定化法 *

【注意】* は一般的な呼び方がわからなかったので適当に名付けたものです。

上に、やたらと出てきた略語の一覧をのせます。

5.13 共役勾配法の復活

1952 年に登場した共役勾配法は、当初の熱狂的なブームが去ると、一時は全然顧みられない時期がありました。

たとえば、偏微分方程式についての古典的名著といわれる [11] では、次のように扱われています。

> Hestenes と Stiefel (1952) の共役勾配法は有限回反復の族の 1 つで、構造は反復的でそれにもかかわらず有限回のステップで終了する。比較的大容量の記憶装置を必要とするのと、各反復段階が比較的複雑な構造のためにいままでのところ偏差分方程式には広く採用されなかった。
>
> G. E. Forsythe and W. R. Wasow (1960)
> 『偏微分方程式の差分法による近似解法』

ただし、かなりの数の数値実験より、共役勾配法は“うまくいけば”SOR 法や Chebyshev 加速法と比較して格段に速く収束することが知られていました。そのかわり、うまくいかない場合は、丸め誤差の影響を受けやすい性格が災いして、何回反復しても少しも先に進まないという事態に陥ることでも有名でした[75]。

収束が遅い場合の対策として、反復が停滞してしまったら一旦反復を打ち切って再スタートする方法や、初期値をうまく改良する方法などが工夫されましたが、なかなか決め手を欠きました。原因は、丸め誤差に対する繊細さにありました。

> 共役勾配法は丸め誤差に弱い。それは「n 回反復したのに残差が 0 にならない」という形で検出される。また、共役勾配法の計算を、単精度と倍精度で行なって比較してみると、単精度の場合には丸め誤差のために、収束 (事実上の) がかなり遅くなっていることがわかる。少ない桁数で計算すれば丸め誤差が入るのは当然であるが、共役勾配法は他の解法に比較して、丸め誤差の影響が大きいようである。
>
> 戸川 隼人 (1975)
> 『共役勾配法』

[75] 倍精度演算が桁違いに遅かった時代は、やはり単精度演算が多く使われました。それが共役勾配法の評判を落とした原因でもあります。理論的には高々 n 回の反復で収束する “はず” なのに全然収束しなければ、当然イライラするでしょう。

結局のところ、共役勾配法の持つ収束性のバラツキを克服するためには、もとの方程式を適当に変数変換する、つまり何らかの行列を引っかけることで、性根を叩き直すしかないという結論に達し、いかに叩き直すかの研究が行なわれました。

Meijerink さんと van der Vorst さんが 1977 年に発表した前処理の手法 ([22]) は、共役勾配法の収束性を劇的に改善することが明らかとなり、その後の理論的な研究の発展と前処理技法との組合せによって、共役勾配法は大型疎行列を係数とする連立 1 次方程式の近似解法として、鮮やかな復活を遂げます。

そして現在、前処理付き共役勾配法は次のような評価を得るまでになりました ([57])。

> 前処理の導入により、 CG 法はガウス消去法などの直接解法をメモリ使用量、計算速度の面で凌いだ。以来、 CG 法系の行列解法は各方面で研究され、正定値が保証されない、弱い非対称行列を解くことが可能な解法まで開発されている。現在ガウス消去法の優れている点は、よほどの悪条件行列でなければ解ける点である。一方、 CG 系の計算方法はパイプライン型スーパーコンピュータにも適しており、今後ますます使用されることであろう。
>
> 羽根 邦夫 (1987)
> 『共役勾配法の計算方法を利用した
> 半導体デバイスシミュレータ』

5.14 反復解法の汎用性

ただし、上の評価は、“特定の偏微分方程式を特定の手法で離散化して得られた大規模疎行列に対して” という付帯条件をつけるべきでしょう。現在のところ、共役勾配法がすべての直接法と反復法に対しあらゆる面で優れているとはいえません。

結局のところ、設定した数学モデルと近似モデルから導かれる連立 1 次方程式の性格によって、適切な手法を選択するしかありません。

その場合、反復法と共役勾配法の難点[76]は、特にスーパーコンピュータの性能を極限まで引き出すには、専門家が作ったライブラリが充実している直接法と違い、何らかの形でチューニングしなければならないという点です。理由は以下の通りです ([13])。

> うまく働くとされている反復法が成功する理由は、ほとんどの場合、それが解こうとしている実際の問題と密接に結び付いた方法になっているからである。したがって、反復解法のサブルーチンをライブラリの中で探しても無駄である。反復法自体は数学的に単純であるが、行列 A の構造は問題によって複雑になり、また、問題に特有の形をしていることが多い。
>
> G. E. Forsythe, M. A. Malcolm
> and C. B. Moler (1977)
> 『計算機のための数値計算法』

5.15 INSPEC を用いた実験

最後に一つだけ、共役勾配法の復活を示すデータをあげます。

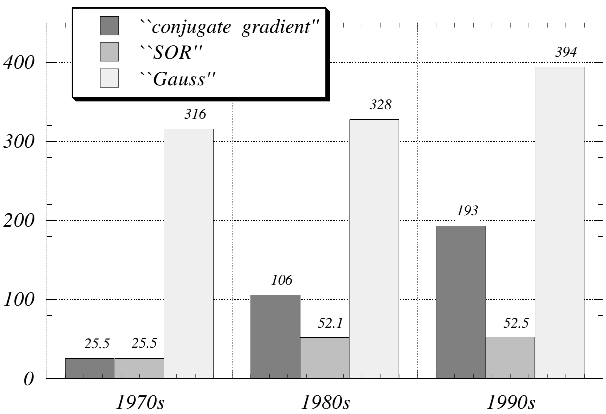
九州大学大型計算機センターでは、 1969 年以降の物理、電気・電子工学、計算機科学、制御工学、情報工学関係の文献のデータベース INSPEC をサービスしています ([66])。

そのデータベースに、以下のキーワードで検索をかけました (1995 年 10 月現在)。

- “conjugate gradient” “linear equation”
- “SOR” “linear equation”
- “Gauss” “linear equation”

実際には “equation” は “equations” を含みます。また “Gauss” は “Gauss” と名前がつくすべてのものが対象です。従って、 “Gaussian” や “Gauss-Seidel” や “Gauss-Jordan” も含んでいます。

このキーワードでヒットした件数を年代別にまとめました。なお、年代別の総データ数が異なるため、 100 万件に対して何件の割合かという値に変換しています。



INSPEC のヒット件数の推移 (100 万件あたり)

[76] ということは論文のネタになりやすいところ

1970 年代は “SOR” と “conjugate gradient” は同じ数でした。 1980 年代には共役勾配法に言及した論文の数が急速に増加しています。 1990 年代になると、 SOR 関係は 80 年代と大差ないですが、共役勾配法関係はさらに数が増えています。

1979 年当時のセンターオープン室

6　Gauss の消去法のアルゴリズム

ここまで、連立 1 次方程式の解法によく使われる方法を直接法、反復法、共役勾配法の順にざっとみてきました。

当然、これらの中でどれが最も優れた解法なのか知りたくなります。しかし、いざ比較するときは、行列 A の条件 (次数や形状など) や、計算機の性能、欲しい精度など、いろいろな条件を加味しなくてはならないので、誰もが納得するような比較をするのはそんなに簡単ではありません。

逆に、これだけたくさんの手法が淘汰されずに残っている[77]という事実は、連立 1 次方程式の決定的な解法が未だ提案されていないことの証明でもあります。

今後は、超並列計算機・ベクトル計算機・ベクトル並列計算機などの、次世代スーパーコンピュータと親和性のある解法が生き残ることでしょう ([43], [45], [56])。

さてこれから先は、最も有名、かつ、行列データを全て主記憶に格納できた場合に最も汎用性があるといわれている **Gauss** の消去法について考えたいと思います。

Crout 法も Cholesky 法も、結果的には LU 分解に帰着されるので、 Gauss の消去法[78]の仲間として考えることにします。

なお、この章は、以降で使用する Gauss の消去法のアルゴリズムと用語のための解説です。本題は最後の章なので、ご存知の方は読み飛ばして結構です。

6.1　Gauss の消去法

連立 1 次方程式を

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i = 1, \cdots, n \tag{13}$$

とします。

行列 A は一般の非対称な密行列で、もちろん正則だとします。

まず、 (13) で $a_{11} \neq 0$ と仮定します。第 1 式に $m_i^{(1)} = a_{i1}/a_{11}$ をかけ、 x_1 の係数である a_{21} から a_{n1} までの成分を $i \geq 2, j \geq 2$ について

$$\begin{aligned} a_{ij}^{(1)} &= a_{ij} - m_i^{(1)} a_{1j}, \\ b_i^{(1)} &= b_i - m_i^{(1)} b_1 \end{aligned}$$

の過程で消去します。

すると方程式は、 (13) と同値な

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & \vdots & \cdots & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(1)} \\ \vdots \\ b_n^{(1)} \end{pmatrix}$$

の形になります。

一般の形で書くと、第 k 段目の消去は演算は、 $a_{kk}^{(k-1)} \neq 0$ と仮定するとき、 $i \geq k+1, j \geq k+1$ に対し

$$\begin{aligned} m_i^{(k)} &= a_{ik}^{(k-1)}/a_{kk}^{(k-1)} \\ a_{ij}^{(k)} &= a_{ij}^{(k-1)} - m_i^{(k)} a_{kj}^{(k-1)} \\ b_i^{(k)} &= b_i^{(k-1)} - m_i^{(k)} b_k^{(k-1)} \end{aligned}$$

で計算します。

消去の段階における $a_{kk}^{(k-1)}$ をピボット($pivot$)または枢軸といいます。

この消去を第 1 式から第 $n-1$ 式まで行なうと、最終的に次の式になります。

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ & & \cdots & \vdots \\ 0 & & & a_{nn}^{(n-1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(1)} \\ \vdots \\ b_n^{(n-1)} \end{pmatrix}$$

この方程式は (13) と同値です。

従って、 $a_{nn}^{(n-1)} \neq 0$ ならば、ただちに n 番目の未知数 x_n が求まり、これを直前の式に放り込むことで x_{n-1} がわかります。

[77]「速い」ことが正義だとすれば、既に淘汰された手法もかなりあります。

[78]Gauss-Jordan の掃き出し法は、 LU 分解法でないので、仲間に入れてあげません。

以下、順番に $k = n-1, \cdots, 1$ に対し

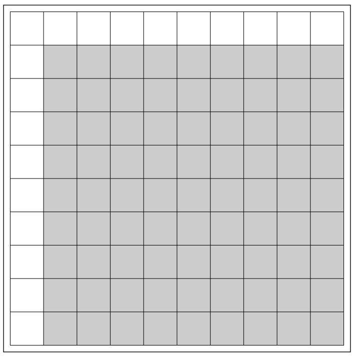
$$x_k = \left(b_k^{(k-1)} - \sum_{j=k+1}^{n} a_{kj}^{(k-1)} x_j \right) / a_{kk}^{(k-1)}$$

で解が求まります。

以上が、基本的な Gauss の消去法の手順です。

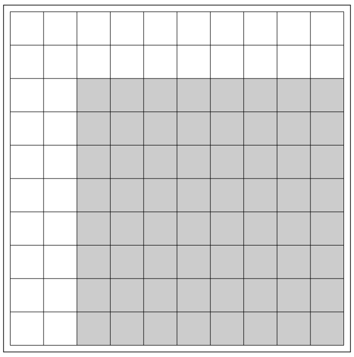
6.2 消去過程

Gauss の消去法による行列 A の変化を図で見ましょう。



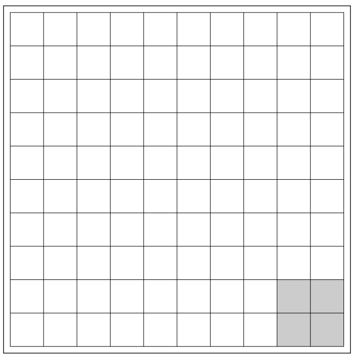
2 段目の消去過程

白い部分は既に消去が完了した部分です。残りのグレーの部分は参照・代入があります。



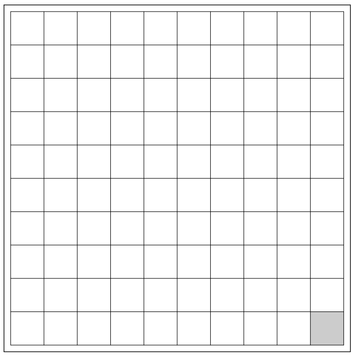
3 段目の消去過程

段が進むにつれて、参照される小行列が縮小していきます。



$n-1$ 段目の消去過程

これが最後の消去過程です。



消去完了

消去過程が終了すると右下に $a_{nn}^{(n-1)}$ が出現します。

6.3 LU 分解との関係

先ほどの消去過程では、右辺 b の値も A と一緒に変形しました。しかし数値計算では、同じ A に対して何度も違う b を使って連立 1 次方程式を解くことがよくあります。その場合、先のアルゴリズムをそのままプログラムとして書くと、b にあわせて何度も同じ A を変形することになって、効率が悪くなります。

そのため、まず行列 A を LU 分解し、三角行列 L と U を用いて b を変形するプログラムに書き直します。このようにすると、異なる b が与えられた場合、以前の LU 分解で蓄積した情報を駆使してスムーズに解を計算することができます[79]。

実は、LU 分解は Gauss の消去法の計算で既に完了しています[80]。

消去過程で出てきた $m_i^{(k)}$ を、消去における乗数 ($multipliers$) といいます。LU 分解の下三角行列 L は、消去における乗数を順番に並べた行列

$$\begin{pmatrix} 1 & & & & \\ m_2^{(1)} & 1 & & 0 & \\ m_3^{(1)} & m_3^{(2)} & 1 & & \\ \vdots & \vdots & & \ddots & \\ m_n^{(1)} & m_n^{(2)} & \cdots & m_n^{(n-1)} & 1 \end{pmatrix}$$

となることが証明できます ([50], [60])。

一方、上三角行列 U は、消去法によって最終的に変形される行列です。従って、Gauss の消去法によって L と U の値がすべてわかります。ということは、LU 分解ができるわけです。

[79] 世の中に出回っている著名な Gauss の消去法関係のライブラリはほとんどこのようになっています。

[80] そのため、Gauss の消去法と LU 分解法は最近同じものとして扱われます。

プログラムを組む場合は、記憶領域の節約のため、最初に A を記憶した場所に L と U を上書きします ([69])。

LU 分解が完了すれば、あとは作業用のベクトル z を用意し、

$$Lz = b$$

を z について解いたあと、

$$Ux = z$$

を x について解けば、解が得られます。

前者を前進代入 (*forward substitution*)、後者を後退代入 (*backward substitution*) と呼びます。

具体的な計算方法は、次の通りです。

前進代入計算

$i = 1, \cdots, n$ の順に、次を計算する。

$$z_i = b_i - \sum_{k=1}^{i-1} l_{ik} z_k$$

後退代入計算

$i = n, \cdots, 1$ の順に、次を計算する。

$$x_i = \left(z_i - \sum_{k=i+1}^{n} u_{ik} x_k \right) / u_{ii}$$

6.4 ピボット選択

前節のアルゴリズムでは、ピボット $a_{kk}^{(k-1)}$ が 0 でないことを仮定していました。しかし、ピボットが 0 となる場合も十分ありえます。また、ピボットの絶対値が小さい場合も、浮動小数点演算でオーバーフローが起き、次の段階に進めないことも考えられます。

その場合、何らかの方法で 0 でない、ついでに、数値的に最良のピボットを探す工夫が必要です。

数値解析の分野で多くの創造的な仕事をされた James Hardy Wilkinson さん (1919–1986) は、 1961 年に発表した論文 [35] で 2 つのピボット捜索法を提案しています。

1 つ目は、 k 段目の消去を行なう前に、 k 列の第 k 行から n 行までを調べ、その中で絶対値最大となるような

$$\max_{k \le i \le n} |a_{ik}^{(k-1)}|$$

がピボットとなるように行を入れ換えてから、次の段階の消去を行なう方法です。

この方法を部分ピボット選択法 (*partial pivoting strategy*)、または、部分軸選択法 といいます。

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & \boxed{a_{22}^{(1)}} & \cdots & a_{2n}^{(1)} \\ \vdots & \vdots & \cdots & \vdots \\ 0 & \boxed{a_{n2}^{(1)}} & \cdots & a_{nn}^{(1)} \end{pmatrix}$$

部分ピボット選択の対象となる成分 (2 段目)

行を入れ換えるということは、 A に左から適当な置換行列 P を引っかけるということです。

2 つ目の方法は、残っている方程式の係数行列 (前節のグレーの部分) すべての成分を調べて、その中で絶対値最大の

$$\max_{k \le i,j \le n} |a_{ij}^{(k-1)}|$$

がピボットとなるように選ぶ方法です。

この方法を完全ピボット選択法 (*complete pivoting strategy*) 、または、完全軸選択法 といいます。

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & \vdots & \cdots & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{pmatrix}$$

完全ピボット選択の対象となる成分 (2 段目)

完全ピボット選択法のためには、行の入れ換えにプラスして列の入れ換えも必要です。これは、 A に適当な置換行列 P_1 を左から、また、置換行列 P_2 を右から引っかけることを意味します。

この二つのピボット選択法の比較は、最終章の中心の話題です[81]。

2 章で紹介した LU 定理は、 A が正則なとき、このような分解が可能なことを保証しています。

6.5 ピボット選択の利点

ピボット選択を行う最大の理由は、ピボット $a_{kk}^{(k-1)}$ が 0 になり、 Gauss の消去法の処理が破綻するためのを防ぐためです[82]。

しかし、 Wilkinson さんの提案したピボット選択法を採用した場合、たとえ $a_{kk}^{(k-1)} \neq 0$ であっても、行や列の入れ換えが生じる可能性があります。

その場合、この操作は無駄なのでしょうか？

[81] ただし、 Wilkinson さんは、最大の要素をピボットに選ぶ方法が最良の方法であるとは言っていません。しかし、現在のところ、これに代わりうる実用的なピボット選択法は提案されていません ([34])。

[82] 実数を 0 でわる行為は、実数体に対する挑戦です。

そうではありません。ピボット選択法は、次節で述べるような、大変都合のいい行列であることがあらかじめわかっている場合を除けば、是非組み込むべき手法です。

その理由は次の通りです ([72])。

- ピボットの絶対値が小さいと、乗数 $m_i^{(k)}$ の計算で桁あふれを起こす危険がある。
- $|a_{kk}^{(k-1)}|$ の値が小さいのは、前の消去の段階で桁落ちをした可能性が大きい。
- $|m_i^{(k)}|$ を小さく、つまり $|a_{kk}^{(k-1)}|$ を大きく選択するほうが、 $a_{ij}^{(k-1)}$ の情報を生かすことができる。

Gauss の消去法における理論的な丸め誤差解析によれば、ピボット選択によって丸め誤差の増大が著しく抑えられることがわかっています ([12])。

6.6 ピボット選択が必要ない行列

一般の行列に Gauss の消去法を適用する場合は、前節にあげた理由よりピボットの部分選択または完全選択を行う必要があります。

ただし、行列 A が以下のどれかの場合は、ピボット選択をしなくも LU 分解可能で、丸め誤差の影響も少ないことがわかっています ([12], [74])。

- 正定値対称行列
- 対角優位行列
- M 行列

行列 A が正定値であるかどうかを判定するには、定義通り固有値を計算して、符号を調べればいいのですが、次数が大きくなるとなかなかそうもいきません。

その場合、行列を導いた “意味” を考えるとわかることがあります。

例えば、有限要素法などではある基底で張られた関数 u_h のノルム式が

$$||\mathrm{grad}\, u_h||^2_{L^2(\Omega)} = z^T A z$$

などと 2 次形式の形で書き表せる場合があります。

もしノルムと基底の性質から、 $z \neq 0$ の場合 2 次形式が正ということがわかるなら、 A が正定値行列であることが判定できます。

また、対称かつ対角優位で対角要素が正の場合、正定値であることが知られています[83]。

6.7 スケーリング

ピボット選択によって、 Gauss の消去法が破綻することなく、しかも丸め誤差の増大が抑えられることがわかりました。

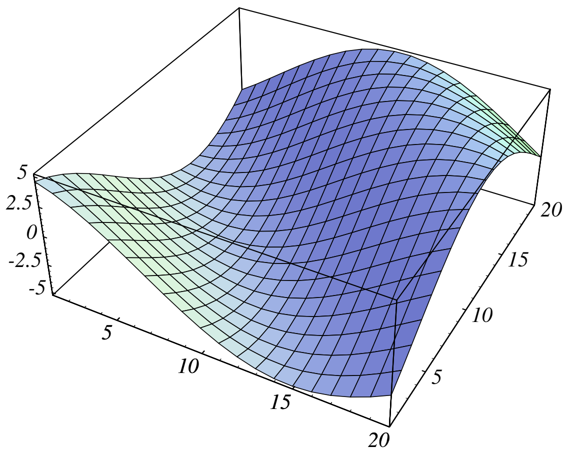
しかし、これには「各行列成分の絶対値の大きさがだいたい同じ程度であれば」という前提条件がつきます。

実際、各方程式に 0 でない定数をかけても、数学的な意味は不変です。しかし、ピボット選択の結果は大きくちがってきます。

一般に、要素の大きさが極端に違うことは好ましくありません。ピボット選択を用いても見当違いの答がでる例も簡単に作れます ([61])。

そこで、消去法に入る前に各要素の絶対値の大きさがだいたい同じ程度になるように調整し、ピボット選択に意味を持たせる必要があります。

この操作を行列のスケーリング (*scaling*) または正規化(*normalization*)、または均等化 (*equilibrium*) といいます。



スケーリング前の行列

現在のところスケーリングの決定版は存在しません[84]が、よく使われるものは次の方法です ([52])。

6.7.1 部分ピボット選択の場合

$i = 1, \cdots, n$ に対し、 i 行の成分の絶対値最大

$$\alpha_i = \max_{1 \le j \le n} |a_{ij}|$$

[83] 厳密には「既約」という条件も必要です。

[84] [13] にスケーリングによって失敗する人工的な例があります。また、[7] も参照下さい。

でもって、

$$a_{ij}^* = a_{ij}/\alpha_i, \quad j = 1, \cdots, n$$
$$b_i^* = b_i/\alpha_i$$

と操作します。式の内容は変わりません。

この変形の結果、すべての係数が絶対値 1 以下となり、少なくとも 1 つは絶対値 1 の係数がどこかにある行列になります。

6.7.2 完全ピボット選択の場合

完全ピボット選択の場合は、式だけでなく未知数の、すなわち列のバランスもとる必要があります。

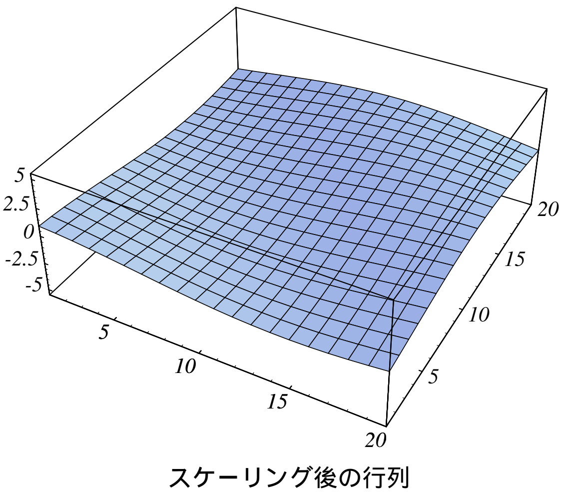
そのためには、各行をその行の絶対値最大の要素で割ったのちに、各列をその列の絶対値最大の要素

$$\beta_j = \max_{1 \le i \le n} |a_{ij}^*|$$

でもって

$$a_{ij}^{**} = a_{ij}^*/\beta_j, \quad i = 1, \cdots, n$$
$$x_j^* = \beta_j x_j$$

と操作します。 β_i の情報はあとで使うので記録しておく必要があります。



スケーリング後の行列

このようなスケーリングによって、新しい行列のどの成分も絶対値 1 以下となり、しかもどの行にも、また、どの列にも少なくとも 1 つの絶対値 1 の要素が含まれる結果となります。

6.8 プログラム例

以上のことから、行列が対称なのか正定値なのか優対角なのか性格が判然としない連立 1 次方程式に対応した Gauss の消去法のプログラムを書く場合、

- ピボット選択を必ずする
- 行列のスケーリングもあわせて行う

と、一応の精度が期待できます。

以下、 Gauss の消去法のプログラムの例をあげます。部分ピボット選択のプログラムは数値計算の本によく載っていますので ([45], [49] の解説がお勧めです)、ここでは省略し、あまりお目にかかることがない完全ピボット選択法の Gauss の消去法プログラムを紹介します。

言語は Fortran で書きましたが、 FORTRAN77 と Fortran90 の中間みたいになってしまいました[85]。なお、簡単なチューニングを次章で解説します。

サブルーチン名は “Gaussian Elimination with Complete Pivoting” の頭文字をとって、さらに、“Ver.0” の意味をつけて “GECP0” としておきます。

```
! ================================================
!     完全ピボット選択 Gauss の消去法プログラム
!      <<注意>> チューニング前なので遅いです！
! ================================================
      SUBROUTINE GECP0(A,NX,N,B,EPS,IW,P,Q,VP,
     &                 VS,VW,ICON)
      DOUBLE PRECISION A(NX,1),B(1),VW(1),VS(1),
     &                 VP(1),D,EPS
      INTEGER N,NX,I,J,K,L,C,P(1),Q(1),IW,ICON
      ICON=0
      IF(N.LT.1.OR.N.GT.NX.OR.EPS.LE.0.0D0)
     &   GO TO 8000
      IF(IW.EQ.2) GO TO 6000
! ----------------- 各式の正規化 ----------------
      DO 10 I=1,N
       D=0.0D0
       DO 20 J=1,N
        IF(ABS(A(I,J)).GT.D) D=ABS(A(I,J))
20     CONTINUE
       IF(D.EQ.0.0D0) GO TO 7000
       DO 30 J=1,N
        A(I,J)=A(I,J)/D
30     CONTINUE
       VP(I)=D
10    CONTINUE
! ----------------- 未知数の正規化 --------------
      DO 40 J=1,N
       D=0.0D0
       DO 50 I=1,N
        IF(ABS(A(I,J)).GT.D) D=ABS(A(I,J))
50     CONTINUE
       IF(D.EQ.0.0D0) GO TO 7000
       DO 60 I=1,N
        A(I,J)=A(I,J)/D
60     CONTINUE
       VS(J)=D
40    CONTINUE
      DO 110 K=1,N
       P(K)=K
       Q(K)=K
110   CONTINUE
```

[85] コメントの “!” は、最近の Fortran コンパイラならほとんど解釈してくれると思いますが、 “C” と同じです。

```
! ------------------ Pivot の捜索 ---------------
      DO 120 K=1,N
       D=0.0D0
       DO 130 I=K,N
       DO 130 J=K,N
        IF(ABS(A(P(I),Q(J))).GT.D) THEN
         D=ABS(A(P(I),Q(J)))
         L=I
         C=J
        END IF
130    CONTINUE
       IF(D.EQ.0.0D0) GO TO 7000
! ----------- index のつけ変え -------------------
       IF(K.NE.L) THEN
        J=P(K)
        P(K)=P(L)
        P(L)=J
       END IF
       IF(K.NE.C) THEN
        J=Q(K)
        Q(K)=Q(C)
        Q(C)=J
       END IF
! ----------- LU Decomposition -----------------
       IF(ABS(A(P(K),Q(K))).LT.EPS) GO TO 7000
       A(P(K),Q(K))=1.0D0/A(P(K),Q(K))
       DO 140 I=K+1,N
        A(P(I),Q(K))=A(P(I),Q(K))*A(P(K),Q(K))
        DO 150 J=K+1,N
         A(P(I),Q(J))=A(P(I),Q(J))
     &               -A(P(I),Q(K))*A(P(K),Q(J))
150     CONTINUE
140    CONTINUE
120   CONTINUE
! ----------- Forward Substitution -------------
6000  DO 160 I=1,N
       VW(I)=B(P(I))/VP(P(I))
       DO 170 J=1,I-1
        VW(I)=VW(I)-A(P(I),Q(J))*VW(J)
170    CONTINUE
160   CONTINUE
! ---------- Backward Substitution -------------
      DO 180 I=N,1,-1
       B(I)=VW(I)
       DO 190 J=I+1,N
        B(I)=B(I)-A(P(I),Q(J))*B(J)
190    CONTINUE
       B(I)=B(I)*A(P(I),Q(I))
180   CONTINUE
! ---------- 解の index つけ換え -----------------
      DO 200 I=1,N
       VW(I)=B(I)
200   CONTINUE
      DO 210 I=1,N
       B(Q(I))=VW(I)/VS(Q(I))
210   CONTINUE
      IC=0
      RETURN
! --------- ERROR が出た場合の処理 --------------
7000  ICON=2000
      RETURN
8000  ICON=3000
      RETURN
      END
```

なお、アルゴリズムは [60] を参考にしました。

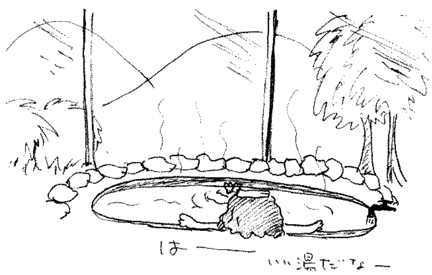
GECP0 では、ピボット選択の過程で生じる方程式と変数の変換情報を記録する整数のインデックスベクトル

$$p = (p(1), p(2), \cdots, p(n))$$
$$q = (q(1), q(2), \cdots, q(n))$$

を用意し、この $p(k)$ と $q(k)$ の値を交換することで正しい結果を得るようにプログラムを組んでいます。

LU 分解のプログラムでは、α_i の情報をそのまま部分ピボット選択に利用する方法を採用する場合が多いのですが (具体的なプログラムは [49] 参照)、LU 分解のコストに比べスケーリングのコストは大したことないことと、プログラムを見やすくするため、陽に行列をスケーリングしています。

次の章では、サブルーチン GECP0 をできるだけ楽をしてスピードアップする方法を考えます。



ちょっと休憩

7　GECP のスピードアップ作戦

この章では、完全ピボット選択付き Gauss の消去法プログラム "GECP0" を、スカラー計算機、およびベクトル計算機でとりあえずのスピードが出るようにチューニングします。

計算機の特性にあわせたチューニングのためには、計算機内部の作りに立ち入った知識が多少必要になります。しかし、最近のプログラム言語の最適化技術は優秀なので、ある程度の筋道を示すと、あとは勝手に仕事をしてくれます[86]

ここでは、富士通株式会社のコンパイラ部隊が総力を結集して作り上げた Fortran コンパイラの "FORTRAN77 EX/VP" を用いて、 GECP0 のスピードアップ作戦を展開します。

7.1　実行時間の調査

まず [45] の指摘を紹介します。

> 経験的事実として、 CPU 時間が全体の中で支配的なプログラムでは、 CPU 時間の大部分 (90% 以上) はソースプログラムの数 % 以下の部分で集中して消費される場合が多い。したがって、プログラムの改良を効率よく行なうためには、計算時間の長いプログラムの部分を検出することが重要である。
>
> 島崎 眞昭『スーパーコンピュータとプログラミング』

プログラム全体で考えると、連立 1 次方程式や固有値計算などが「CPU 時間の大部分」を消費することはよく知られています。

つまり、 GECP0 の計算が遅いと、全体の計算時間が遅くなってしまいます。その意味で GECP0 のチューニングは非常に大事になります。

第一にやるべきことは、 [45] の指摘通り、

各処理過程で最もコストがかかる部分を特定する

ことです。

GECP0 は 4 つの部分に分けることができます。

1. 正規化 (スケーリング) 処理
2. ピボット選択と行・列の交換
3. LU 分解
4. 前進代入 / 後退代入、および解の並べ換え

すぐれたプログラミング言語には、実行時の振舞いを解析するソフトウェアが付属しています[87]。

しかし、ここではもっと具体的に、プログラム内に CPU 時間を計測する関数を埋め込んで、 4 つの処理の経過時間を測定します。

測定は 1995 年 10 月 6 日におこないました。計算機は FUJITSU　VP2600/10、コンパイラは FORTRAN77 EX/VP です。最適化オプションは`OPT(E)`として、スカラーモードとベクトルモードで方程式の次数を変えてデータを集めました[88]。

以下は、 GECP0 における各処理過程の全体の実行時間に占める割合です。まずはスカラーユニットのみを使った実行結果です。

各処理に要するコスト (スカラーモード)

処理過程	$n=100$	$n=1000$
行列の正規化	2.64%	0.23%
ピボット選択処理	49.53%	38.25%
LU 分解	46.39%	61.40%
前進消去 / 後退代入	1.44%	0.12%

データからすぐわかるように、ピボット選択と LU 分解にかかるコストが他を圧倒しています。

各処理に要するコスト (ベクトルモード)

処理過程	$n=100$	$n=1000$
行列の正規化	0.66%	0.00%
ピボット選択処理	20.57%	0.80%
LU 分解	78.03%	98.20%
前進消去 / 後退代入	0.73%	0.00%

対してベクトルモードでは、 LU 分解に大半のコスト ($n=1000$ で 98% 以上) が集中しています。

これは極めて重要なことです。なぜこのような違いが生じるのでしょうか？

[86] こういった優秀なのが研究室にたくさんいればいいのですが、現実はそうもいきません。

[87] FORTRAN77 EX/VP には ANALYZER というツールがあります。

[88] このあたりの用語がわからない方は、 [63] をごらん下さい。

ベクトル化できない原因

$n = 1000$ での実行時間を見ると、

	スカラー	ベクトル
ピボット選択処理	149 秒	1.9 秒
LU 分解	239 秒	237 秒

です。ベクトルモードでは、ピボット選択処理の実行時間が 149 秒から 1.9 秒と大幅に短縮できています。対して LU 分解の方は全く改善されていません。これは、 LU 分解の部分が全くベクトル化できてないことを意味します。ベクトル計算機にとって、この事態は極めて遺憾なことです。

従って、ベクトルモードの処理において LU 分解に大量のコストがかかる理由がわかります。つまり、他の処理がそこそこベクトル化できているのに対して、 LU 分解の部分だけ全くベクトル化できていないため、相対的なコストが増大したわけです。

では、この部分の改良を考えましょう。

7.2 行列の値の交換

7.2.1 親切なコンパイラ

GECP0 のプログラムでピボット選択によって生じる行と列の交換は、行の情報を記憶するインデックスベクトルの値を交換することですませています。

このようにすれば、実際に行列の値を変換をする手間が省け、計算時間の節約になる、と思うのが普通です。インデックスベクトルを添字に用いる方法は George E. Forsythe さんと Cleve B. Moler さんの書いた有名な数値計算の教科書 [12] などでも推奨されています[89]。が、このインデックスベクトルを使ったプログラムは、ベクトル計算機に適さないことが知られています。

原因は、プログラムを組む人のベクトルコンパイラに対する説明不足にあります[90]。例えば、

```
      DO 140 I=K+1,N
       A(P(I),Q(K))=A(P(I),Q(K))*A(P(K),Q(K))
140   CONTINUE
```

の部分をコンパイラが見た場合、添字の P(I) と Q(I) の値は実行するまでわからないと判断します。

実際、ピボット選択が済むまで P(I) とQ(I) の値はわかりません。しかし、プログラムを書く人は、

> P(I) と Q(I) は 1 から n までのどれかの値をとり、同じ値となることは絶対にない

ことを知っています。よって、I を K+1 から N まで「せーの」でベクトル処理して欲しいと期待するのは当然です。

ところが、コンパイラはそんなことはつゆ知りませんので

> 「もしかしたら P(I) は 1, 2, 2, 1,... などのように、重なりがある回帰データかも知れないな。もしそうなら、ベクトル処理によって A が変な値に書きかわってしまうかもしれないぞ。これはまずい。スカラーユニットで順番に処理しようっと。」

と考え、この部分をスカラーモードで実行します。これは、コンパイラにとって当然の選択です。

その結果、本来はベクトルユニットで実行されるべき DO ループが非常に遅くなってしまいます[91]

では、どうすべきか？ 対策は次のどちらかです。

《対策 1》 ベクトル化を推進する最適化指示行をソースプログラムに埋め込み、配列に回帰参照がないことをコンパイラに指示する。

《対策 2》 おしゃれなインデックスベクトルに頼るのをやめ、ピボット選択時に本当に行と列の値を交換することでコンパイラに自分の意図を思い知らせる。

7.2.2 最適化指示行の挿入

《対策 1》は、 DO ループの前に「この配列には回帰参照はないよ」という意味の行を 1 つ追加します。 FORTRAN77 EX/VP の場合、次のようになります。

```
 *VOCL LOOP,NOVREC(A)   ! この行を追加
      DO 100 I=K+1,N
       A(P(I),Q(K))=A(P(I),Q(K))*A(P(K),Q(K))
100   CONTINUE
```

NOVREC は “no vector recurrence” の略です。詳しくは [68] を御覧下さい。

ただしこの方法は、ベクトル計算機とコンパイラの種類によって異なりますので、あくまでも個別撃破の作戦となります[92]。

[89]もっとも、これは部分ピボット選択のプログラムに関することで、この場合 A(P(I),J) に対しての話です。完全ピボット選択の場合 A(P(I),Q(J)) なので状況は異なります。

[90]あるいはコンパイラの認識不足ともいえるのですが、それは酷というものです。

[91][63] の実験でわかるように、ベクトル計算機からベクトルユニットをとれば、汎用機どころかパソコンにも負ける可能性があります。

[92]また、この例でのベクトル化の効果は《対策 2》と比べて半分以下です。

7.2.3 本当に行と列の値を交換する

汎用性を追求するならば、対策 2 です。さっそくプログラムを修正します。

```
      DO 120 K=1,N
!-------------- Pivot の捜索 ------------------ !
       D=0.0D0
       DO 130 I=K,N
       DO 130 J=K,N
        IF(ABS(A(I,J)).GT.D) THEN   !  変更 1
         D=ABS(A(I,J))              !  変更 2
         L=I
         C=J
        END IF
130    CONTINUE
!------------- index のつけ変え ---------------- !
       IF(K.NE.L) THEN
        J=P(K)
        P(K)=P(L)
        P(L)=J
        DO 131 J=1,N                ! 追加 1
         D=A(K,J)                   ! 追加 2
         A(K,J)=A(L,J)              ! 追加 3
         A(L,J)=D                   ! 追加 4
131     CONTINUE                    ! 追加 5
       END IF
       IF(K.NE.C) THEN
        J=Q(K)
        Q(K)=Q(C)
        Q(C)=J
        DO 132 I=1,N                ! 追加 6
         D=A(I,K)                   ! 追加 7
         A(I,K)=A(I,C)              ! 追加 8
         A(I,C)=D                   ! 追加 9
132     CONTINUE                    ! 追加 10
       END IF
```

以上はピボット選択の部分です。

“追加” の部分で行と列の入れ換えを行なうようにしました。なお、入れ換えの情報 P, Q は、前進 / 後退代入で使うのでそのまま残します。

```
! -------- LU Decomposition ---------------- !
      A(K,K)=1.0D0/A(K,K)
      DO 140 I=K+1,N
       A(I,K)=A(I,K)*A(K,K)
       DO 150 J=K+1,N
        A(I,J)=A(I,J)-A(I,K)*A(K,J)
150    CONTINUE
140   CONTINUE
```

以上は LU 分解の部分です。インデックスベクトル P と Q が追放され、すっきりしました。

さっそく実行時間を計測します。さきほどと同じ $n = 1000$ でのデータです。

	スカラー	ベクトル
ピボット選択処理	112 秒	0.88 秒
LU 分解	224 秒	0.45 秒

うまくベクトル化できました。コンパイラは今度は LU 分解をすべてベクトルユニットで処理するコードをはきだしたため、スカラーモードに比べて何と 498 倍のスピードがでました。こうでなくちゃいけません。

また、行と列を交換したプログラムの方が、オリジナルのインデックスベクトルを使ったプログラムよりスカラーモードでも速くなっています。行と列の交換部分を新規に追加するため、その分計算時間が余計にかかるのでは、と心配されたピボット選択処理部分も、37 秒速くなりました。その理由は、

- ピボット選択の実行時間の大部分は、最大ピボット探索の DO ループが消費する
- その DO ループ内で A(P(I),Q(J)) が A(I,J) とスッキリしたため、実行効率があがった

ためです。

7.3 添字の動き

続いては、Fortran 言語の特徴を利用します。次の文をお読み下さい ([60])。

> 一般に、FORTRAN では、2 次元以上の配列の配列要素は、実際のメモリ上では、第 1 の添字の順に連続して並び、次に第 2 の添字が一つ増えて再び第 1 の添字の順に連続して並ぶようにコンパイルされることになっている。（中略）したがって、たとえば 2 次元配列の添字のそれぞれを二重の DO ループで動かすような場合には内側の DO ループで第 1 の添字が動き、外側の DO ループで第 2 の添字が動くようにプログラムを書くべきである。
>
> 森 正武『数値解析法』

これに従ってプログラムを修正します。まず、ピボット捜索の部分の

```
       DO 130 I=K,N
       DO 130 J=K,N
        IF(ABS(A(I,J)).GT.D) THEN
         D=ABS(A(I,J))
         L=I
         C=J
       END IF
130    CONTINUE
```

の I と J を入れ換え

```
       DO 130 J=K,N      !    <--- 修正
       DO 130 I=K,N      !    <--- 修正
        IF(ABS(A(I,J)).GT.D) THEN
         D=ABS(A(I,J))
         L=I
         C=J
        END IF
130    CONTINUE
```

とします。次に、LU 分解の部分

```
      DO 140 I=K+1,N
       A(I,K)=A(I,K)*A(K,K)
       DO 150 J=K+1,N
        A(I,J)=A(I,J)-A(I,K)*A(K,J)
150    CONTINUE
140   CONTINUE
```

で二重ループの内側で第 1 添字の I が動くように

```
      DO 140 I=K+1,N
       A(I,K)=A(I,K)*A(K,K)    !  <-- ループ分割
140   CONTINUE                 !  <-- 修正
      DO 145 J=K+1,N           !  <-- 修正
       DO 150 I=K+1,N          !  <-- 修正
        A(I,J)=A(I,J)-A(I,K)*A(K,J)
150    CONTINUE
145   CONTINUE                 !  <-- 修正
```

と修正します。その他、前進 / 後退代入計算も修正することができますが、全体に占めるコストが微々たるものなので、何もしないことにします。

実行結果を見ましょう。

	スカラー	ベクトル
ピボット選択処理	38 秒	0.87 秒
LU 分解	29 秒	0.45 秒

ベクトルモードで実行時間が変わらないのは、既に自動ベクトルコンパイラによってチューニングが完了しているためです ([45], [49])。

一方、スカラー処理では、たったこれだけの修正なのに、ピボット選択で約 3 倍、LU 分解で 7.7 倍もの性能がでました。以上 2 つの改良方法は、ベクトル計算機だけでなく、汎用機やワークステーション、パーソナルコンピュータなどにも共通する技法です。

7.4 GECP プログラム

プログラムのチューニングの余地はまだまだあります。例えば、前進 / 後退代入とスケーリングの部分はまったく手つかずです。また "ループアンローリング" という有名な手法をとり入れるのも効果的でしょう。さらに、記憶領域の節約の面では、引数で使うスケーリング情報を格納するベクトルは、宣言を不要にすることができます。

しかし、最低限のチューニングをするだけでも、(もとのプログラムに比べて) 格段の性能アップができましたので、一応の完成にします。

出来上がりの完全ピボット選択サブルーチン名を "GECP" とします。

以下はプログラムリストです。

```
! ========================================== !
! ------------------------------------------ !
!
      SUBROUTINE GECP(A,NX,N,B,EPS,ISW,P,Q,VP,
     &                VS,VW,ICON)
! ------------------------------------------ !
! <<概要>>
!   実行列の連立 1 次方程式 Ax=b を complete pivoting
!   を用いた Gauss の消去法で解く倍精度用サブルーチン
! <<引数>>
!      A : 係数行列．演算後，内容は保存されない．
!     NX : 配列 A の整合寸法
!      N : 係数行列 A の次数 (N>1)
!      B : 定数ベクトル B．解ベクトル x として出力
!    EPS : 係数行列の特異性の判定定数
!    ISW : ISW=1 のとき -- 最初から処理を行う
!          ISW=2 のとき -- LU 分解のデータはそのまま
!                          B の値だけを換えて実行
!      P : pivoting の列の入れ換え情報ベクトル
!      Q : pivoting の行の入れ換え情報ベクトル
!     VP : 式のスケーリング情報ベクトル
!     VS : 方程式のスケーリング情報ベクトル
!     VW : 作業用実ベクトル
!   ICON : condition code
!                 0 -- 正常終了
!              2000 -- 係数行列のある行か列がゼロ
!                      または pivot 判定が EPS 以下
!              3000 -- 引数が間違っている
! ------------------------------------------ !
      IMPLICIT NONE
      DOUBLE PRECISION A(NX,1),B(1),VW(1),VS(1),
     &                 VP(1),D,EPS
      INTEGER N,NX,I,J,K,L,C,P(1),Q(1),ISW
!
! ------------------------------------------ !
!            引数のチェック
! ------------------------------------------ !
      ICON=0
      IF(N.LT.1.OR.N.GT.NX.OR.EPS.LE.0.0D0)
     &   GO TO 8000
      IF(ISW.EQ.2) GO TO 6000
!
! ========================================== !
!          スケーリング (正規化処理)
! ------------------------------------------ !
!  * 処理内容 *
!    Pivoting の前段階として，各係数の大きさがほぼ
!    等しくなるように，個々の式をその式の絶対値最大の
!    係数で割り調整する．
!    さらに各未知数間のバランスをとるため，係数行列の
!    各要素をその列の絶対値最大の係数で割る．
!  * 注意 *
!    式のスケーリング情報を VP に保存．また，
!    未知数の情報を VS に保存する
! ========================================== !
!
! ------------- 各式の正規化 ------------------ !
      DO 10 I=1,N
       D=0.0D0
        DO 20 J=1,N
         IF(ABS(A(I,J)).GT.D) D=ABS(A(I,J))
20      CONTINUE
        IF(D.EQ.0.0D0) GO TO 7000 ! 行列が非正則
        DO 30 J=1,N
         A(I,J)=A(I,J)/D
30      CONTINUE
       VP(I)=D
10    CONTINUE
```

```
! ------------- 未知数の正規化 ---------------- !
       DO 40 J=1,N
        D=0.0D0
         DO 50 I=1,N
          IF(ABS(A(I,J)).GT.D) D=ABS(A(I,J))
 50      CONTINUE
         IF(D.EQ.0.0D0) GO TO 7000 ! 行列が非正則
         DO 60 I=1,N
          A(I,J)=A(I,J)/D
 60      CONTINUE
        VS(J)=D
 40    CONTINUE
!
! ========================================== !
!          Complete Pivoting
! ------------------------------------------ !
!  ＊処理内容＊
!   LU 分解の過程の小行列で絶対値最大の要素を特定
!   対応する行と列の番号をベクトル P, Q に保存
! ========================================== !
      DO 110 K=1,N
       P(K)=K         ! 初期値の設定
       Q(K)=K
 110  CONTINUE
! ------------- pivot の捜索 ----------------- !
      DO 120 K=1,N
       D=0.0D0
        DO 130 J=K,N
        DO 130 I=K,N
         IF(ABS(A(I,J)).GT.D) THEN
          D=ABS(A(I,J))
          L=I                ! pivot となる行番号
          C=J                ! pivot となる列番号
         END IF
 130    CONTINUE
       IF(D.EQ.0.0D0) GO TO 7000 ! 行列が非正則
! ----------- index のつけ変え ---------------- !
       IF(K.NE.L) THEN
        DO 131 J=1,N
         D=A(K,J)
         A(K,J)=A(L,J)       ! 行の入れ換え
         A(L,J)=D
 131    CONTINUE
        J=P(K)
        P(K)=P(L)            ! 行情報の入れ換え
        P(L)=J
       END IF
!
       IF(K.NE.C) THEN
        DO 132 I=1,N
         D=A(I,K)
         A(I,K)=A(I,C)       ! 列の入れ換え
         A(I,C)=D
 132    CONTINUE
        J=Q(K)
        Q(K)=Q(C)            ! 列情報の入れ換え
        Q(C)=J
       END IF
```

```
! ========================================== !
!             LU Decomposition
! ------------------------------------------ !
       IF(ABS(A(K,K)).LT.EPS) GO TO 7000
       A(K,K)=1.0D0/A(K,K)
        DO 140 I=K+1,N
         A(I,K)=A(I,K)*A(K,K)
 140    CONTINUE
        DO 145 J=K+1,N
          DO 150 I=K+1,N
           A(I,J)=A(I,J)-A(I,K)*A(K,J)
 150      CONTINUE
 145    CONTINUE
 120  CONTINUE
!
! ========================================== !
!             Forward Substitution
! ------------------------------------------ !
 6000 DO 160 I=1,N      ! ISW=2 の場合，ここから
       VW(I)=B(P(I))/VP(P(I))
        DO 170 J=1,I-1
         VW(I)=VW(I)-A(I,J)*VW(J)
 170    CONTINUE
 160  CONTINUE
!
! ========================================== !
!             Backward Substitution
! ------------------------------------------ !
      DO 180 I=N,1,-1
       B(I)=VW(I)
        DO 190 J=I+1,N
         B(I)=B(I)-A(I,J)*B(J)
 190    CONTINUE
       B(I)=B(I)*A(I,I)
 180  CONTINUE
!
! ========================================== !
!             解の index つけ換え
! ------------------------------------------ !
      DO 200 I=1,N
       VW(I)=B(I)
 200  CONTINUE
!
      DO 210 I=1,N
       B(Q(I))=VW(I)/VS(Q(I))
 210  CONTINUE
      ICON=0          ! ここまで来れば正常終了
      RETURN
!
! ========================================== !
!          ERROR が出た場合の処理
! ------------------------------------------ !
 7000 ICON=2000
      RETURN
 8000 ICON=3000
      RETURN
      END
```

7.5 実行時間の比較

完成したプログラム “GECP” で各処理がどれくらいのコストを占めるかをベクトル・スカラーモード別に表にします。

各処理に要するコスト (スカラーモード)

処理過程	$n = 100$	$n = 1000$
行列の正規化	4.30%	1.33%
ピボット選択処理	56.74%	55.88%
LU 分解	37.14%	42.29%
前進消去 / 後退代入	1.82%	0.52%

相変わらずピボット処理と LU 分解の部分にコストが集中しています。うち、ピボット操作の実行時間が半分以上を占めています。

各処理に要するコスト (ベクトルモード)

処理過程	$n = 100$	$n = 1000$
行列の正規化	2.61%	0.63%
ピボット選択処理	64.50%	65.40%
LU 分解	30.94%	33.59%
前進消去 / 後退代入	1.94%	0.38%

ベクトルモードでは、ピボット処理にかかる割合がさらに増加して 60% をこえています。さらに詳しく調べると、ピボット選択処理の 95% 以上が

```
      DO 130 J=K,N
      DO 130 I=K,N
       IF(ABS(A(I,J)).GT.D) THEN
        D=ABS(A(I,J))
        L=I
        C=J
       END IF
130   CONTINUE
```

の部分に集中していることがわかります。コンパイラの情報を見ると、この部分は既にベクトル化されています。従って、数倍の order はあったとしても、数十倍、数百倍の order での改善は難しいと思われます[93]。

この部分のコストは、部分ピボット選択ではほとんど無視できる値です。つまり、どんなに頑張って GECP をチューニングしても、絶対に部分ピボット選択法より速くなることはありません[94]。演算回数を比較しても、計算時間が 2 倍程度なら素晴らしい速さだといえます。

[93]はっきり言って、この部分のチューニングをどうやっていいのかよくわかりません。ループアンローリングくらいでしょうか？

[94]部分ピボット選択のプログラムも同様にチューニングした場合の比較です。

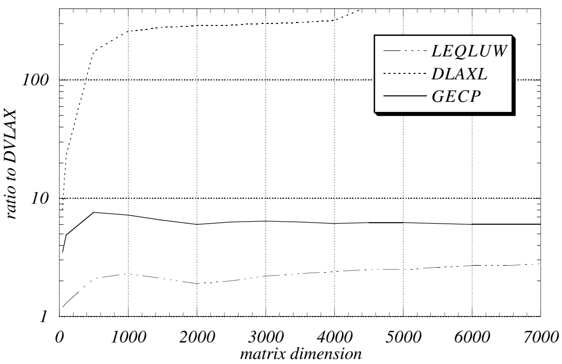
では、 GECP サブルーチンの処理速度を他の連立 1 次方程式のサブルーチンと比較します。計算はベクトル計算機 VP2600/10 で行ないました。連立 1 次方程式のサブルーチンは次の 4 つです。

サブルーチン名	解法
DVLAX	Gauss の消去法 (部分ピボット選択)
LEQLUW	Crout 法 (部分ピボット選択)
DLAXL	Householder 変換による QR 分解
GECP	Gauss の消去法 (完全ピボット選択)

DVLAX と DLAXL は SSL II のサブルーチンです。 DLAXL は、もともと最小自乗法用なのですが、行と列の数を揃えると連立 1 次方程式の解法としても使えます。 LEQLUW は NUMPAC のサブルーチンです。

この中で VP2600/10 の性能を最高に引き出すサブルーチンは DVLAX です。 [63] での調査によれば、 VP2600/10 のピーク性能に迫る数値を出しているので、他はとてもたちうち出来ません。

よって、 DVLAX の処理時間を 1 として、各サブルーチンがその何倍の処理時間になるか、次数を変えながら測定しました。



サブルーチンの処理性能

QR 分解の DLAXL はもともと演算数が LU 分解に比べて多いことと、ベクトル計算機にうまく対応していないこともあって、 $N = 4000$ で DVLAX の 300 倍以上の実行時間がかかってしまいました。 LEQLUW は $N = 2000$ までほぼ 2 倍の処理速度でしたが、 N が増えるとだんだん性能が低下して、 $N = 7000$ では 2.8 倍になりました。 GECP は $N = 500$ のときに最大 7.6 倍となりましたが、 N が大きくなるとじわじわと差を詰め、 $N = 7000$ では 6 倍の実行時間となりました。

LEQLUW と GECP の比較では、目標の “2 倍” から見るとまあまあの結果です。ただし DVLAX のレベルから見れば、まだまだと言えるでしょう。

8　Gauss の消去法は安定か？

最後の章は、電子計算機が世の中に登場して以来、ずっと議論の種になっている古典的命題

> 部分ピボット選択の Gauss の消去法は、はたして安定な数値解法か？

をとりあげたいと思います。

この問題は、理論的には既に解決済みです。ただし、経験的・実用的には “もやもや” としたものが残ります。

8.1　消去法の歴史

Gauss さんは、 1809 年に出版された『天体運動論』という本の中で消去法を紹介し ([50])、この方法は広く世に知られるところとなりました。

しかし、 Gauss さんが消去法のアルゴリズムの発明者というわけではないようです。

18 世紀後半にフランスで活躍した数学者 Joseph Louis Lagrange さん (1736–1813)[95]は、 Gauss さんが『天体運動論』を出す数十年前に、斉次な系を解くための消去法のアイデアを既にほのめかしていました ([17])。

また、次の事実もあるそうです。

> 遠く中国では『九章算術』の第 8 章「方程」に、ガウスの消去法とまったく同じアルゴリズムの記述がある。『九章算術』は紀元前 2 世紀の頃に編纂されたというから、すでに 2000 年も前から知られていたわけである。
>
> 戸川 隼人『数値計算』

みなさんも、高校や大学で問題を解いた[96]ことがあると思いますが、消去法や掃き出し法は、理論が明解な上、四則演算さえ知っていれば、あとは機械的な行列計算で答が出てきます。

そのため、紙と鉛筆で連立 1 次方程式を解く場合の有力な手法として、 Gauss の消去法は揺るぎない地位を獲得しました。

1940 年代になると、電子計算機が本格的な実用段階に入り、それにともなって、丸め誤差の研究もさかんになります。当然、連立 1 次方程式の簡単な解法である Gauss の消去法も格好の研究対象となりました。

最初になされた研究では、丸め誤差が Gauss の消去法にあたえる影響はきわめて悲観的でした。 [30] から引用します。

> 電子計算機時代の幕開けの頃、 Gauss の消去法は連立 1 次方程式の解法として役に立たないのでは、と懸念された。 Hotelling は 1943 年の論文で、 $A^TAx = b$ の形をした $n \times n$ 方程式の解において誤差は 4^{n-1} 程度になるであろうと予言した。彼はこう指摘する。「78 行の行列では、小数点第 1 位の近似精度を保証するのですら、 46 桁が必要となるだろう。」
>
> L. N. Trefethen and R. S. Schreiber
> “Average-Case Stability of Gaussian Elimination”

最後の “桁数云々” の部分は、行列の次数を $n = 78$ として 10 進数の計算機を用意したとき、誤差が 4^{n-1} ですから、

$$10^x = 4^{77}$$

の解として、桁数が 46 以上絶対に必要なことを意味します。

当初の研究での結論は、 100 個程度の未知数を持つ方程式系を解く場合でも、不可能なほどの多くの有効桁を保って演算しない限り、必要な精度はとても得られまい、というものでした。

そのため、一時期 Gauss の消去法は計算機を用いた数値計算の舞台から姿を消し、しばらく反復法の天下になります[97]。

[95] 変分法、解析力学の創始者です。また、代数の世界でも偉大な足跡を残しました。メートル法の制定における委員長でもあった人です。

[96] 正確には解こうと努力した

[97]「半分眠りながらでも解けるよ」と弟子への手紙で反復法を勧めていた Gauss さんも、生きていれば喜ばれたでしょう。消去法に “Gauss の” という言葉がくっついていることを当人が知っていたかどうかよく解りません。

浮動小数点演算と固定小数点演算[98]における誤差解析を理論的に発展させ、 Gauss の消去法を数値計算の表舞台に復活させせた人たちは、 Alan Mathison Turing さん (1912–1954)、 John von Neumann さん (1903–1957) と Herman Heine Goldstine さん (1903–?)、そしてとりわけ James Hardy Wilkinson さん (1919–1986) です ([36])。

8.2 後退誤差解析

Wilkinson さんの誤差評価手法は、後退誤差解析 (*backward error analysis*)といわれる手法です。読んで字のごとく、解をたどって定式化を行ない、誤差を見積もる方法です。

後退誤差解析の方法と成果は [35], [36], [37] に詳細にまとめられています[99]。これに対して、前進誤差解析 (forward error analysis)もあります。こちらは [27], [28] を御覧下さい。

後退誤差解析より得られた知見の中から、 Gauss の消去法に関する誤差評価を紹介します。まずは、用語の説明からはじめます。

8.2.1 マシンイプシロン

マシンイプシロン (*machine epsilon*)は、浮動小数点数の相対誤差の上限をあらわす値で、計算機と浮動小数点体系によって決定されます ([60], [54])[100]。

マシンイプシロンを ε_M とすれば、

$$\varepsilon_M = c\beta^{1-t} \tag{14}$$

です。 β は基数、 t は仮数部の桁数、 c は丸めの方法によって決まる定数で、四捨五入の場合 1/2、切捨ての場合 1 となります。

現在よく使われている数値表現での ε_M の値は以下の通りです ([55], [67])。

	β	t	ε_M
IBM 形式 単精度	16	6	9.54×10^{-7}
倍精度	16	14	2.22×10^{-16}
4 倍精度	16	28	3.10×10^{-33}
IEEE 形式 単精度	2	24	5.96×10^{-8}
倍精度	2	53	1.11×10^{-16}

《注意》 ε_M は 10 進数に換算した近似値

ε_M は、反復法の収束判定などによく利用される基本的な数値です。

[98]現在はあまり使われなくなりました。詳しくは [36] を御覧下さい。

[99]今はもう古典の部類に属してるといえるかも知れません。

[100]「計算機イプシロン」ともいいます。

8.2.2 条件数

誤差を測るためには、何らかの形の「距離」の概念が必要です。その尺度をノルム(*norm*) といいます。

ベクトルのノルムを

$$||x||_\infty = \max_i |x_i|$$

また、行列のノルムを

$$||A||_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|$$

で定義します[101]。

ノルムの性質をより、直ちに次のことが言えます。

条件数

A が ΔA だけ変化したとき x が Δx だけ変わったとすると

$$\frac{||\Delta x||_\infty}{||x+\Delta x||_\infty} \leq ||A||_\infty ||A^{-1}||_\infty \frac{||\Delta A||_\infty}{||A||_\infty}$$

をみたす. また, b が Δb だけ変化したときは

$$\frac{||\Delta x||_\infty}{||x||_\infty} \leq ||A||_\infty ||A^{-1}||_\infty \frac{||\Delta b||_\infty}{||b||_\infty}$$

となる.

この二つの不等式は、 x の変化が A や b の変化の何倍に拡大されるかをあらわしています。

この拡大を表す値を

$$\kappa_\infty(A) = ||A||_\infty ||A^{-1}||_\infty \tag{15}$$

で書き、 $\kappa_\infty(A)$ を行列 A の条件数 (*condition number*) といいます。

一般に、条件数が大きくなると、係数のわずかな変化で解の変化に大きな影響がでます ([58])。

条件数がものすごく大きくなるような行列は、相対誤差が拡大する可能性が高く、不安定性を持ちます。このような行列を悪条件 (*ill-conditioned*) な行列といいます。

実際の計算で $||A^{-1}||_\infty$ の値を少ない計算量で正確に見積もるのはなかなか難しい問題で、感度解析とよばれる分野の中心問題です。詳しくは [55] を御覧下さい。

[101]有限次元空間の場合、“ノルムの定義” をみたすノルムはすべて同じと考えることができます。ノルムには他にも $||\cdot||_1$ や $||\cdot||_2$ などがあります (詳しくは [32], [60] 参照)。

8.2.3 誤差評価

以上の数値を用いて誤差評価を行ないます。

$\hat{x}$ を浮動小数点演算による Gauss の消去法を用いて求めた近似解、また ρ を

$$\rho = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max i, j |a_{ij}|} \tag{16}$$

で定義します。このとき Wilkinson さんはいくつかの仮定のもと、次の有名な定理を証明しました ([37])。

Wilkinson の定理

連立 1 次方程式を Gauss の消去法で求めたとき、その相対誤差は次で評価される。

$$\frac{||\hat{x} - x||_\infty}{||x||_\infty} \leq 4n^2 \kappa_\infty(A) \rho\, \varepsilon \tag{17}$$

条件数 $\kappa_\infty(A)$ は行列 A が決まれば固定されます。また、 ε も計算機と精度により決まります。

$4n^2$ は次数 n によって決まり、浮動小数点演算においては不可避の値です。しかし、倍精度演算をすれば、マシンイプシロンの頑張りでなんとか小さくなりそうです[102]

結局、 Wilkinson さんの与えた定理より次のことがわかります。

Gauss の消去法は、条件数と ρ がめちゃくちゃ大きくならない限り安定である。

ρ は、三角行列化における不安定因子(*growth factor*) 、または増加因子と呼ばれます ([38])。

8.2.4 悪条件の場合の対策

条件数 $\kappa_\infty(A)$ がどれくらいの大きさになれば "悪条件" と言うか、基準はいろいろです。少なくとも n に関して指数関数的な勢いで、たとえば 2^n とか e^n とかで増加してしまうと、もうお手上げです。

[55] から引用します。

良条件の問題は、誤差が増大しない安定な方法を用いれば精度よく解くことができる。悪条件の問題を精度よく解くためには、いくら解法を工夫してもだめである。唯一の対策は計算桁数を多くすることである。

名取 亮『数値解析とその応用』

[102]逆にこの定理は、単精度演算で $n = 10000$ や $n = 100000$ などの計算をすることの無謀さを語っています。ご注意下さい。

8.2.5 Hilbert 行列

悪条件となる行列の有名な例として、 **Hilbert** 行列があります。 Hilbert 行列は、 David Hilbert さん[103](1862–1943) が 1894 年に発表した論文より名付けられたもので、

$$a_{ij} = \frac{1}{i + j - 1} \tag{18}$$

となる行列のことです。

有名な行列のわりには、どうやって導かれたのか数値解析の本にはあまり書いてないので、経緯を [12] から簡単に説明します。

問題は、区間 $\Omega = [0, 1]$ で定義される連続関数 $f(x)$ を $n-1$ 次までの多項式で

$$f^n(x) = \sum_{i=1}^{n} c_i x^{i-1}$$

の形で近似したときの L^2 ノルムの誤差

$$E(c) = ||f^n(x) - f(x)||_{L^2(\Omega)}$$

を最小にするように係数 c_i を決定するものです。なお、 L^2 ノルムとは

$$||g(x)||_{L^2(\Omega)}^2 = \int_0^1 g(x)^2\, dx$$

です。 $E(c)$ の微分は 0 となるので、 c_i でひとつずつ微分して変形すれば、

$$\sum_{j=1}^{n} \left(\int_0^1 x^{i+j-2}\, dx \right) c_j = \int_0^1 f(x) x^{i-1}\, dx$$

を得ます。

これは、 n 個の未知数に対する連立 1 次方程式になっています。行列 A の成分は

$$a_{ij} = \int_0^1 x^{i+j-2}\, dx = \frac{1}{i + j - 1}$$

となって、 (18) が出てきます。

Hilbert 行列は、対称かつ正定値行列なのですが、その条件数は指数関数的に増大します (詳しい値が知りたい方は [12], [18] を参照下さい)。

従って、ほんのわずかな誤差の混入が著しく解の精度を損ないます。対策としては、計算桁数を多くとるしかありません。

Hilbert 行列は、そのあまりの条件数の悪さ、即ち解きにくさから、 1940 ~ 50 年代にかけて盛んに研究されました[104]

[103]20 世紀前半の最も偉大な数学者の一人です。 1900 年パリの国際数学者会議における『23 の数学の問題』は特に有名です。

[104]$n = 10$ までの逆行列を与えるだけの論文というのもあります。

ただし、 Hilbert 行列の構造を研究するのも大事なのですが、 George E. Forsythe さんの次の指摘も大事です ([12])。

Hilbert 行列の条件の悪さは、この行列の導入に用いた近似の問題にさかのぼることができる。区間 $0 \leq x \leq 1$ の上での関数 x^i $(i = 0, \cdots, n-1)$ はほとんど 1 次従属である。このことは Hilbert 行列の行ベクトルがほとんど 1 次従属であること、すなわちこの行列がほとんど特異であることを意味する。（中略）要するに、この近似の問題は、 Hilbert 行列のような行列を導く形をしている以上、問題の “たて方が悪い”。

G. E. Forsythe and C. B. Moler
『計算機のための線形計算の基礎』

8.3 不安定因子の上限

Gauss の消去法における Wilkinson さんの定理 (17) のもう一つの主役は、不安定因子 ρ です。

条件数が純粋に行列の性格、つまり問題の設定そのものに起因するのに比べ、不安定因子は行列の性格にプラスして、算法の方法によって大きく変わります。

Wilkinson さんは、誤差定理を証明するだけでなく、さらに [35] において、 ρ の理論的な上限値を与えました[105]。以下、方程式はスケーリングされていると仮定します。

ρ の上限 (Wilkinson)

部分ピボット選択を用いたとき

$$\rho \leq 2^{n-1} \tag{19}$$

完全ピボット選択を用いたとき

$$\begin{aligned} \rho &\leq n^{1/2}(2^1 3^{1/2} 4^{1/3} \cdots n^{1/(n-1)})^{1/2} \\ &\leq 1.8 n^{1/4 \log n} \end{aligned} \tag{20}$$

対称正定値行列のとき (ピボット選択なし)

$$\rho \leq 1$$

対角優位行列のとき (ピボット選択なし)

$$\rho \leq 2$$

Hessenberg 行列に部分ピボット選択を用いたとき

$$\rho \leq n$$

[105] 1961 年のこの論文の中で初めて “complete pivoting” と “partial pivoting” の用語が用いられました。

これによって、 A が悪条件でない場合、対称正定値行列[106]と優対角行列はピボット選択をしなくても安定、また Hessenberg 行列でも部分ピボット選択によってほぼ安定といえます。

また、 Wilkinson さんによれば、完全ピボット選択の上限 (20) は極めて「ゆるい」限界であって、証明した本人も「ひどい過大評価だ」と認めています。実数行列において今まで知られている最大の ρ は n です。任意の正則行列に対して $\rho \leq n$ が成り立つのではないか、という Wilkinson さんの有名な予想があり、特定の n では証明ができていますが、一般の証明はまだのようです ([23], [30])。

ここでは、その予想を信用することにします。

完全ピボット選択の Gauss の消去法は、条件数が指数関数的に増大しない限り安定

問題は、部分ピボット選択の上限 2^{n-1} です。 (19) と (20) を比べて見ましょう。

n	10	20	50
complete pivoting	6.1	15	75
partial pivoting	512	$524,288$	5.6×10^{14}

部分ピボット選択の不安定因子の上限は、ものすごい勢いで増加します。これは、ちょっとまずいことになりそうです。

8.4 Wilkinson さんの例題

部分ピボット選択を用いた Gauss の消去法における不安定因子の上限 2^{n-1} を証明した Wilkinson さんは、同じ論文で正確な上限を与える行列を披露しています ([35])。

n 次行列を A_n とします。行列の作り方は以下の通り、いたって簡単です。

$k = 1, 2, \cdots, n-1$ に対し、 A_n の第 k 列を

- 上から $k-1$ 番目までが 0
- そのあとに $1, 1, -1, 1, -1, 1, \cdots$ が続く
- 最後の列だけは $-1, 1, -1, \cdots$

として作成します。 A の条件数はそれほど大きくありません ([35])。従って、解の安定性は不安定因子 ρ の大きさにかかってきます。

[106] 非対称正定値でも $\rho \leq 1$ が示せますが、その場合、 L の成分が大きくなる可能性があります。

$n = 5$ のときは

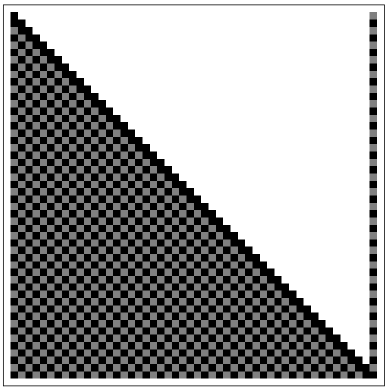
$$\begin{pmatrix} 1 & 0 & 0 & 0 & -1 \\ 1 & 1 & 0 & 0 & 1 \\ -1 & 1 & 1 & 0 & -1 \\ 1 & -1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 & -1 \end{pmatrix}$$

です。これを Gauss の消去法、または Crout 法で LU 分解します。部分ピボット選択での行の変換は起きません[107]。下三角行列 L は、もとの A_5 の下三角成分と完全に一致します。一方 U は、

$$\begin{pmatrix} 1 & 0 & 0 & 0 & -1 \\ & 1 & 0 & 0 & 2 \\ & & 1 & 0 & -4 \\ & 0 & & 1 & 8 \\ & & & & -16 \end{pmatrix}$$

となり、右下の成分の絶対値が $2^4 = 2^{n-1}$ となっています。

行列の形を A_{50} で描いてみます。白い部分が 0、黒が 1、グレーが -1 をとります。



以下同様に、一般の A_n を部分ピボット選択で LU 分解した場合、上三角行列 U の右下成分が

$$(-1)^n 2^{n-1}$$

となることが確認できます。

このことより、n が大きくなると、部分ピボット選択の Gauss 消去法は非常に不安定になり、U のわずかな違いが見当違いの LU の値を生む危険性をはらんでいます。

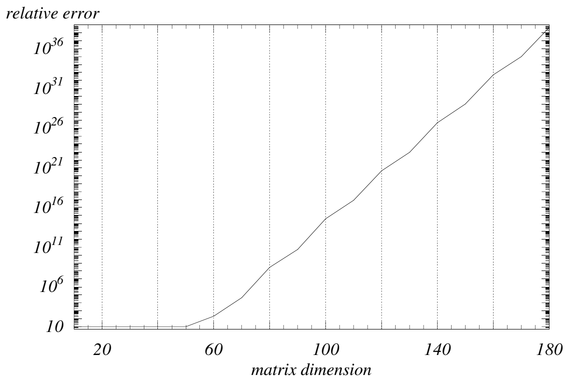
なぜ危険かの理論的な説明は [33], [35], [38] に任せることにして、さっそく数値実験をします。

Wilkinson さんの定義に従い、行列 A_n を作ります。b は、解が $x = (1, 1, \cdots, 1)^T$ となるように

$$b_i = \sum_{j=1}^{n} a_{ij}$$

で定義します。

n を 10 から 180 まで 10 きざみで動かし、誤差を測定しました。計算は倍精度でおこなっています。計算機は FUJITSU M-1800/20、FORTRAN77 EX コンパイラです。スケーリングつき部分ピボット選択の Gauss の消去法サブルーチンとして、SSL II 拡張機能の DVLAX を使いました ([70])。



不安定因子が $\rho = 2^{n-1}$ の場合の解の振舞い

$n = 60$ になる前から真の解から外れ、見当違いな方向に飛んでいってしまいました。

この原因を調べてみます。解が飛んで行ってしまう 55 から 60 までの a_{nn} の値を書き出します。

n	a_{nn}
55	18014398509481984
56	36028797018963968
57	72057594037927936
58	14411518807585587$\underline{0}$
59	28823037615171174$\underline{0}$
60	57646075230342349$\underline{0}$

IBM の浮動小数点の精度は、10 進数に直すと約 17 桁です。n を 58 まで大きくすると、仮数部の精度以上の情報が必要になります。浮動小数点演算では、情報をため込む領域ははじめから決まっているので、計算機は仕方なく下線部の情報をカットします。

その結果、$a_{nn} - 1/2$ などの計算は、意味をなくし、その演算を繰り返すことによって、必要な情報が次々に落ちてしまい、結果的に答がどんどん正解から離れていきます。

この反例より、次の悲観的な結論が導かれます。

部分ピボット選択の Gauss の消去法は、条件数がよい場合でも不安定となる場合がある。

[107] ここがミソです。完全ピボット選択の場合、ピボットが変化して、不安定因子は常に 2 以下に抑えられます。

8.5 なぜ部分ピボット選択法が使われるのか？

Wilkinson さんの後退誤差解析の結論を見れば、ピボットの部分選択法に関して、1943 年の Hotelling さんの予想の “4^{n-1}” が “2^{n-1}” になっただけのように見えます。

では、理論的にすぐれた手法[108]である完全ピボット選択法や、さらにすぐれた手法である QR 分解法[109]が、なぜ部分ピボット選択に代わって台頭しなかったのでしょうか。

それには、“実際には” という、数値計算における膨大な経験則が関わってきます。

いくつかの本と論文から、Gauss の消去法に関するコメントを年代順に拾っていきます。

> 疑いもなく密な行列を持つ連立 1 次方程式を解くための最もよく知られた方法は Gauss の消去法およびその種々の変型である。N 個の未知数に関する N 個の密な方程式は $N^3/3$ 程度の回数の乗算とそれに通常あらい時間評価をする際には無視される他の算術演算で解くことができる。
> von Neumann と Goldstein による完全な誤差解析からディジタル型計算機で連立 1 次方程式の固定小数点方式の解における積み重ねられた丸めの誤差に対するくわしい誤差限界が与えられる。
>
> G. E. Forsythe and W. R. Wasow (1960)
> 『偏微分方程式の差分法による近似解法』

von Neumann さんと Goldstein さんは、1947 年に発表した論文の中で、正定値対称行列の場合に、丸め誤差の影響は、それまで考えられていたほど大きくないことを示しました ([38])。

> （完全ピボット選択法を説明した後）消去の進行を最後まで円滑に進行させるには、このようにするのが最も望ましいことである。人間の眼は状況を 2 次元で認識するので、上のような絶対値最大なものの捜索は人手による計算ではむしろ簡単である。機械計算では 1 次元の過程でこれを走査していかなければならない面倒があるが、しかし機械の高速性を考慮すれば大したことではないので、ぜひこの手つづきを経由するのが望ましい。
>
> 宇野 利雄 (1963)
> 『計算機のための数値計算』

おそらく、電卓や数式処理ソフトの普及が、紙と鉛筆を使って自力で計算する能力を減退させているのは間違いないでしょう。

> プログラム 5.2 は Gauss の消去法を使って連立 1 次方程式を解く。このプログラムの算法が単純であることは教育の目的には理想的である。一般に優れた Gauss-Jordan 法の算法はプログラム 9.7 で示す。
>
> J. M. McCormick and M. G. Salvadori (1964)
> 『FORTRAN による数値計算プログラミング』

この本 ([21]) では、Gauss-Jordan 法に部分ピボット選択を施すプログラムをこの後で紹介してます。

Gauss-Jordan 法が、Gauss の消去法に比べて「一般に優れている」理由は、どうやら逆行列の計算が簡単にできるためらしいです。

> A が対角優位でなく、正定値対称でも Hessenberg でもなく、U の増大について何も情報をもっていない場合には、完全ピボット選択を行なう必要がある。
>
> B. Wendroff (1966)
> 『理論数値解析』

理論的な立場からは、この結論になるでしょう。
肝心の Wilkinson さんの意見はどうでしょうか。

> ($\rho = 2^{n-1}$ となるような) どちらかというと特殊な行列も存在する。しかし、実際には、そのように大きくなるのは非常にまれである。任意の要素が 8 の大きさになることも、ごく珍しいことであり、行列がまったく悪条件であるときは、次々と $a_{ij}^{(k)}$ は減少する場合のほうがずっとありそうなことである。さらに、その成長がもっとゆるやかな因子で制限されていることが “事前的に” わかっているようないくつかの重要な類がある。
>
> J. H. Wilkinson (1963)
> 『基本的演算における丸め誤差解析』

“いくつかの重要な類” とは、先ほどの定理であげたように、A が対称正定値や対角優位や Hessenberg 行列な場合のことです。

つまり、$\rho = 2^{n-1}$ となるようなものは “例外” であり、人工的に意地悪して作らないかぎり、不安定因子が増大することはない、と Wilkinson さんは主張します。

さらに、完全ピボット選択法の問題点も指摘しています。

---

[108] 誤差が少なくて安定しているという意味。

[109] A を Householder 法で QR 分解した場合、不安定因子は $\rho \leq n^{1/2}$ になります ([38])。

> こうして理論的に優れていることが示されても、完全ピボット選択法の利用が有効であるかどうかは疑わしい。とくに、行列の各成分の数値の記憶に磁気テープを使うときには、実際の管理上の問題が完全ピボット選択法においては、重要なことになってくるのである。完全ピボット選択法の r 段階において、最大成分を定めるためには、$(n-r+1)^2$ 回の減算をしなければならないが、浮動小数点方式の演算では、減算の所要時間は、乗算に比較して無視できるというわけにはいかない。
>
> J. H. Wilkinson (1967)
> 『性質の悪い一次方程式の解法』

ピボット選択のために必要な比較の回数を調べると

部分選択	$n(n+1)/2$ 回
完全選択	$n(n+1)(2n+1)/6$ 回

です。LU 分解にかかる演算回数 $(2/3)n^3$ 回 ([41]) と比較して、部分選択の場合は無視できる回数ですが、完全選択の場合、ほぼ同じ回数がピボットを選ぶ作業で必要になります。また、実際にこの点が完全選択法の最大のネックになることが、前章の数値例を見ることで納得できます。

つまり、ピボットの完全選択法の方が、数少ない (と Wilkinson さんは言っています) 特別な場合によりよい精度の結果を得ることができることは理論的にわかっているのですが、“経験的に” 部分選択法でも安定な結果が得られているので、計算時間の節約のため、部分選択で十分だと Wilkinson さんはおっしゃっているわけです。

この、“経験的に安定” と言い切れるだけの数値実験の積み重ねが、ピボットの部分選択付き Gauss の消去法を「最も有名かつ信頼できる (密行列に関する) 連立 1 次方程式の数値解法」の地位に押し上げることになります。

1970 年代に入ると、数値計算の専門家は次のような結論を下します ([13])。

> 行列の計算にとり組んできた人々が過去 15 ないし 20 年間に知った唯一のしかもきわめて重要なことは、次の事実であろう。ピボットの部分的選択を行う Gauss 消去法では残差が小さいことが保証される。
>
> G. E. Forsythe, M. A. Malcolm and C. B. Moler (1977)『計算機のための数値計算法』

1980 年代に入っても、ピボットの部分選択付き Gauss の消去法に与えられた地位はますます堅固なものになります。[73] を引用します。

> 計算に誤差がないならば、有限回の 4 則演算で解が得られるのが直接法である。この中で、ピボットの選択を伴う Gauss 消去法が計算の手間と精度の上で最も優れた方法とされている。一時期、その精度について悲観的見方があったが、誤差の後退解析によって、この方法の信頼性が保証された。（中略）完全選択法を用いた場合の方が $a_{ij}^{(k)}$ の増大率について、より小さい上界が得られているが、経験上、部分選択法で十分である。
>
> 『岩波数学辞典』(1985)

そうはいっても、根本的な疑問が残ります。Trefethen さんの言葉を借りましょう ([30])。

> Wilkinson による Gauss の消去法についての標準的誤差解析は、ただちに安定性の議論を消去過程で成分のサイズがどのくらい増大するのかという問題に帰着させる。つまり、$PA = LU$ の作成段階で、U の成分が A のそれと比べて極端に大きく成長することがありうるか、ということである。原理的には答えは yes である。簡単な例により 2^{n-1} 倍もの大きさにすることができる。しかし現実には決して起きない。この経験的な結論はあまりによく確立してしまったため、次の注目すべき状況を忘れがちである。すなわち、なぜ (ピボットの部分選択付き)Gauss の消去法が安定であるのか、誰も知らないのである！
>
> L. N. Trefethen (1985)
> “Three Mysteries of Gaussian Elimination”

8.6 random matrix での実験

L. N. Trefethen さんは、この問題を 5 年間あたためた結果、自分の問いかけに対して自分で答を出そうとしました。R. S. Schreiber さんとの共著で 1990 年に発表した論文 [30][110]において、Trefethen さんは、次数が 1024 以下の行列に対し、適当な乱数を発生させて作った random matrix で Gauss の消去法を片っぱしから解き、不安定因子の平均値を算定しました。

それによれば、平均値は

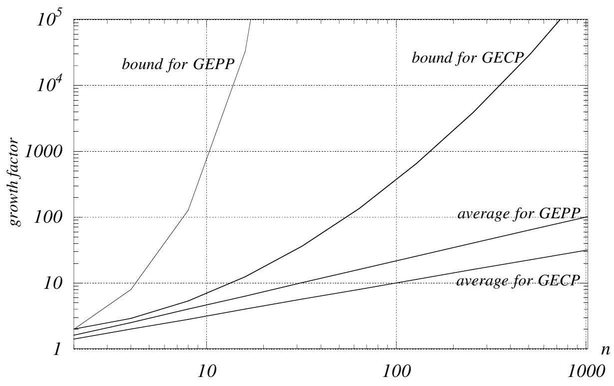
部分ピボット選択のとき	$\rho \approx n^{2/3}$
完全ピボット選択のとき	$\rho \approx n^{1/2}$

となりました[111]。

これをグラフにします。

[110] 副題は “Dedicated to the memory of Jim Wilkinson.” です。

[111] random matrix がどれくらい有効性を持つのかは、意見が分かれます。random matrix を用いたテストは大変重要だが、十分ではない、という意見もあります ([14])。



不安定因子の増大の様子

上の二つの曲線は、 Wilkinson さんが証明した不安定因子 ρ の上限 (19), (20) です。下の二つは Trefethen さんが random matrix を用いて計算した不安定因子の平均値です。

L. N. Trefethen さんと R. S. Schreiber の実験が実際に数値計算で使用される一般の行列をうまく反映していると信じるならば、このグラフによって不安定因子の増加は最悪の理論値よりも相当低いことになります。

不安定因子が指数関数的に増加する行列は、平均値から遠く離れた極めて特異なものであり、もし行列の条件数がめちゃくちゃに大きくなければ、部分ピボット選択付き Gauss の消去法は安定であると結論づけられます。

この頼もしい結果も取り込んで、現在数値計算をする人の共通の認識は次のようになっています。

> 部分的ピボッティングのコストは、計算全体のコストに比して無視できるくらい軽微なものであるが、完全ピボッティングのコストは大きく、計算全体そのものと同程度である。一方、実用上は部分的ピボッティングで十分であることがわかっているので、通常は部分的ピボッティングを採用する。部分的ピボッティングが有効に働かないで誤差の増大を起こすような例題も作れるが、それは病理学的な例で、実際上、めったに出会わないものである。
>
> 津田 孝夫 (1988)
> 『数値処理プログラミング』

8.7 病理学的な例

「めったに出会わない」ということは、たまには出会うということです。その意味で Trefethen さんが「現実には決して起きない」と言うのは行き過ぎです。

最近、 Wilkinson さんが作った人工的な行列ではなく、“実際のモデル” から導かれたそれほど悪条件でない行列に対して、不安定因子が指数関数的に増加する例がいくつか報告されてましたので、簡単な紹介と行列の作り方をお教えします。

8.7.1 Wright さんの例

1 つめは、 S. J. Wright さんが 1993 年に [39] で発表した例です。 $\mathbb{R}^n$ の 2 点境界値問題

$$y' = M(t)y + q(t), \quad y(a) = y(b) = \beta$$

を考えます。求めたいのは $y(t) \in \mathbb{R}^n$ です。 $M(t)$ は t についての $n \times n$ の関数行列、 $q(t)$ はベクトルです。この問題を標準的な shooting method[112] を適用し離散化します。 Wright さんは特別な場合として $n = 2$ の

$$M(t) = \begin{pmatrix} -1/6 & 1 \\ 1 & -1/6 \end{pmatrix}, \quad a = 0, \quad b = 60$$

を考えました。係数行列 A は次のようになります。

$$A = \begin{pmatrix} I & & & & I \\ -e^{Mh} & I & & & 0 \\ & -e^{Mh} & I & & \vdots \\ & & \ddots & \ddots & 0 \\ & & & -e^{Mh} & I \end{pmatrix}$$

I は 2 次の単位行列、 h は区間 $[a, b]$ の分割幅、

$$-e^{Mh} = I + Mh + O(h^2) \approx \begin{pmatrix} h/6 - 1 & -h \\ -h & h/6 - 1 \end{pmatrix}$$

です。 A は h が小さいなら、それほど条件数は悪くなりません。行列の作り方は次のように極めて簡単です。

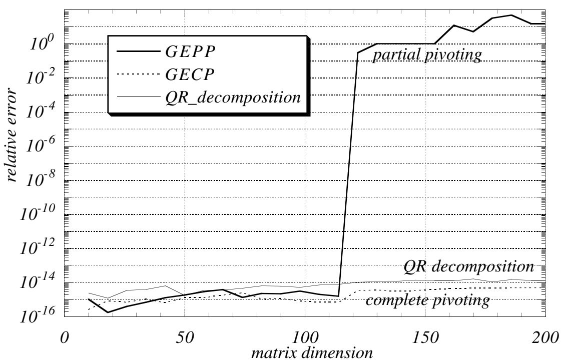
```
      read(5,*) n        ! 分割数
      h = 60.0D0/n       ! 分割幅
      m=2*n+2            ! 行列の次数
      do 10 i=1,m
       do 10 j=1,m
        A(i,j) = 0.0D0   ! 初期化
 10   continue
      do 30 i=1,m
       A(i,i) = 1.0D0    ! 対角成分
 30   continue
      do 40 i=1,n
       A(2*i+1,2*i-1) = h/6-1.0D0
       A(2*i+2,2*i-1) = -h
       A(2*i+1,2*i)   = -h
       A(2*i+2,2*i)   = h/6-1.0D0
 40   continue
      A(1,m-1) = 1.0D0   ! 残り
      A(2,m)   = 1.0D0
```

[112] 手法は [39] の参考文献を御覧下さい。

ところが、この行列 A を係数行列にする連立 1 次方程式を部分ピボット選択付き Gauss の消去法で LU 分解すると、不安定因子が指数関数的に増大してしまいます[113]。 Wilkinson さんの行列と同じく、解が $x = (1, 1, \cdots, 1)^T$ となるように b を決めて数値実験します。計算機は FUJITSU M-1800/20 です。比較のサブルーチンは

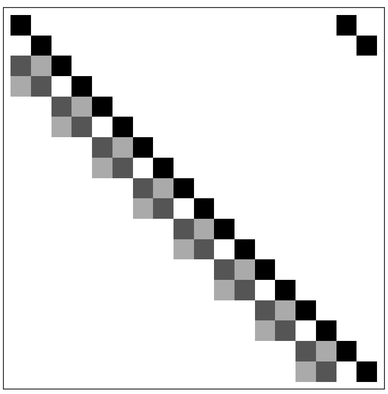
- GECP(完全ピボット選択)
- GEPP(部分ピボット選択)
- QR 分解

で行いました[114]。



QR 分解と完全ピボット選択は、期待通りの精度を示しています。一方、部分ピボット選択では、行列の次数が 120 くらいで見当違いな解に発散します。原因は、 Wilkinson さんの例題と同じで、不安定因子の指数関数的な増加のためです。

参考のために、行列 A の形を graphics にします。□が 0、■が 1、残りはグレーで書きます。



Wright さんの作った行列の形

8.7.2 Foster さんの例

実際の数学モデルから導かれる連立 1 次方程式において部分ピボット選択付き Gauss の消去法が失敗する 2 つ目の例として、 L. V. Foster さんが 1994 年に [14] に発表した論文[115]を紹介します。

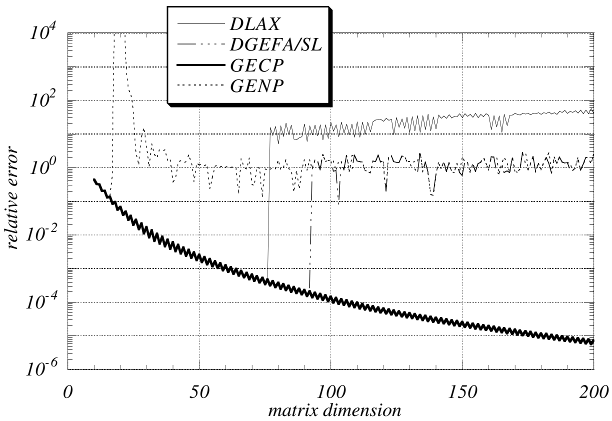
問題は 2 階の線形 Volterra[116]型積分方程式から導かれる

$$x(s) - \int_0^s k(s,t)x(t)\,dt + \beta(s)x(L) = G(s)$$

を一般的な求積法で近似するものです。求めたいのは区間 $[0, L]$ 上の $x(s)$、その他の関数は既知です。 Foster さんは、既知の関数を連続問題の解が具体的にわかるように選び、その離散近似を行いました。積分の近似は Newton-Cotes[117]公式と Simpson[118]の公式を組み合わせて用います。

この近似から導かれる行列 A は悪条件ではありません。しかし、部分ピボット選択による LU 分解をすると、ある n 以上ではピボット選択をしなくなり、そのため不安定因子が急激に増大します。

Foster さんは素晴らしいことに、 MATLAB という行列演算ソフトを使ったプログラムをネットワーク上で公開していました[119]。そこで、プログラムをとり寄せて Fortran にプログラムを組み直し、数値実験しました。



これは [14] の最初の例題です。誤差は、連続問題の真の値と近似解の相対誤差です。 DLAX は部分ピボット選択付きの Crout 法の LU 分解、 DGEFA/SL は LINPACK のサブルーチンです。また、 GENP はスケーリングもピボット選択もしない Gauss の消去法の解法、 GECP はおなじみスケーリングと完全ピボット選択を行なったサブルーチンです。

[113] $n = 200$ 程度で 2.59×10^{21} くらいに増大します。

[114] GEPP は SSL II の DVLAX、QR 分解は同じく SSL II の Householder 変換の DLAXL を用いました。

[115] "Gaussian Elimination with Partial Pivoting Can Fail in Practice" という、そのものズバリの表題です。

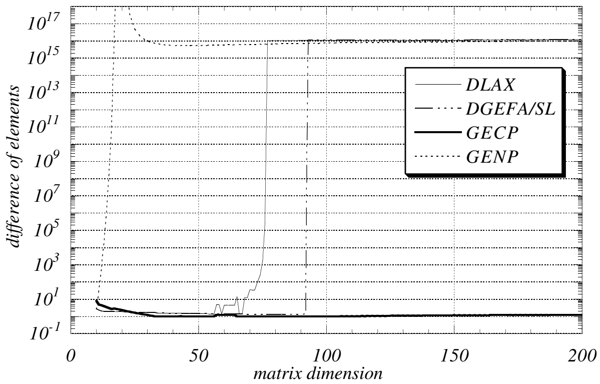
[116] Vito Volterra(1860–1940) さんはイタリアの数学者です。

[117] Roger Cotes さん (1682–1716) の名前が半分入っています。

[118] Thomas Simpson さん (1710–1761) から来ています。

[119] アクセス先は論文に書いてあります。

GECP が順調に収束していくのに対し、他のサブルーチンは、ある n から急に力つきて飛んで行ってしまう様子がよくわかります。



こちらは、不安定因子の値をプロットしたものです。完全ピボット選択以外のプログラムが真の解から突然離れていく原因が、不安定因子の急激な増大による丸め誤差の累積によることがわかります[120]。

[14] にはもう一つ例題があり、上と同様な挙動を示します。こっちは A がすぐに作れます。Fortran で書けば、

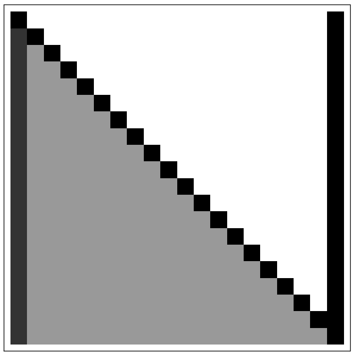
```
      read(5,*) n        !  行列の次数
      k = 1.0D0          !  定数 1
      C = 6.0D0          !  定数 2
      L =40.0D0          !  区間幅
      h = L/dble(n-1)    !  分割幅
      do 20 i=1,n
       do 20 j=1,n
         A(i,j) = 0.0D0 ! 初期化
 20   continue
      do 30 i=3,n
       do 30 j=2,i-1
        A(i,j) = -k*h
 30   continue
      do 40 i=2,n
       A(i,1) = -k*h/2.0D0
       A(i,i) = -k*h/2.0D0 + 1.0D0
       A(i-1,n) = -1.0D0/C
 40   continue
      A(1,1) = 1.0D0
      A(n,n) = 1.0D0 - 1.0D0/C - k*h/2
```

で部分ピボット選択付きの Gauss の消去法が倍精度演算で失敗する例ができます。

この行列の形状を次のページに描きます。□が 0、グレーが $-kh/2$ の部分です。

Wilkinson さん、Wright さん、Foster さんの行列の形を見比べると、なんとなく似ていることに気づきます。このような形の行列を乱数で発生させるのはかなり難しいでしょう[121]。

逆に、行列の形から部分ピボット選択の Gauss の消去法が失敗することを事前に判断する基準ができるならば、これは素晴らしいことです[122]。



Foster さんの作った行列の形

Foster さんは、論文の最後の章の「なぜ部分ピボット選択が失敗する実際の例がこれまで報告されなかったのか？」という項で、次のように述べています ([14])。

> 我々は (失敗する例がこれまで報告されなかった) 本質的な理由は、このような例が極めて “まれ” であるからだと期待する。問題の設定と解法へのアプローチは慎重に選択されたものである。しかしながら、数値線形計算のソフトウェアパッケージのほとんどが不安定因子について何の情報も提供しておらず、そして不安定因子の増大が、時々ではあるが、実際に発生している可能性があることを指摘すべきであろう。
>
> L. V. Foster (1994)
> “Gaussian Elimination with Partial Pivoting Can Fail in Practice”

“速さ” の名のもとに疎外されている完全ピボット選択付き Gauss の消去法が “精度” の名のもとに頼りにされるのは、このような場面です。

おそらく、ほとんどの専門家が指摘する通り、部分ピボット選択が実際の数値計算において致命的な不安定因子の増大を起こすことは “まれ” なのでしょう[123]。しかし、プログラムのバグ取りも兼ねて数値計算の信頼性をあげるために複数の解法を試してみることは、大変効果があります[124]。そんなとき、完全ピボット選択付き Gauss の消去法は (2 倍以上の実行時間がかかりますが) うってつけの解法だといえます。そして、もしかしたら皆さんを “致命的な失敗” から救ってくれるかも知れません。

[120] これだけ不安定因子が増大すると、反復改良をかけても、行列が “特異” だと判断されてしまいます。

[121] 条件を限定した random matrix を作れば、違う結果が出るかもしれません。

[122] ただし、“この形以外は安定である” というのを証明するのは大変だと思います。

[123] なんせ見つけただけで論文になるのですから。

[124] ついでに精度や定数もいろいろ変えてみましょう。

参考文献と索引

あとがき

この原稿は、もともと連立 1 次方程式の数値解法の一覧表として (自分のために) 書き始めした。現在の研究のいきがかりから、連立 1 次方程式の解の厳密な評価 (浮動小数点演算を使う以上、ある幅を持った区間の中に解を包み込むことになります) が必要になったからです。ついでに最大固有値の厳密な評価もいるのですが、こっちは手つかずです。

解の評価のためには精度良い近似値が必要です。近似解を得る手段は “どんな方法を用いても” 構いません。そこで、世間にはどんな解法があるのだろうと思い、数値解析の教科書などを見ながらメモしていった結果がこの記事となりました。

もともとが “精度がいちばん、速度は二の次” という妙な視点で見ていったため、 6 章以下、特に 8 章は部分ピボット選択付き Gauss の消去法が失敗する可能性ばかりを探ってしまいました。

しかし、 Gauss の消去法の歴史や丸め誤差についての議論など、ご存知なかった方にはなかなか面白かったのではないでしょうか。また、 Wilkinson さんや Forsythe さんなどの名前が出てきて懐かしく思われた方もいらっしゃると思います。

さて、肝心の厳密な解の評価ですが、こちらは “区間解析” と “不動点定理” という数学的な側面と、アルゴリズムを実現する精度保証付き数値演算ソフトの使い方という “利用の手引” の側面があって、今回は解説する時間 (および能力) がありませんでした。近いうちに、この原稿の続編として紹介できれば思います。

参考文献について

以下の参考文献の中には、『古典的名著』の誉れを受けながら絶版になっている本も少なくありません。一応、孫引きは避け、原典を極力見るようにしましたので、参考文献のほとんどは九州大学中央図書館または大型計算機センター図書室にあるか、または書店で注文すれば手に入ります (1995 年 11 月現在)。文献に簡単な紹介をつけました。何かの参考になれば幸いです。

参考文献

[1] Benoit, Commandant : Note sur une méthode de résolution des équations normales provenant de l'application de la méthode des moindres carrés a un systéme d'équations linéaires en nombre inférieur a celui des inconnues —Application de la méthode a la résolution d'un systéme defini d'équations Linéaires (Precédé du Commandant Cholesky), *Bullètin Géodésique*, No.2, pp.67–77 (1924).

— “Commandant” Cholesky の業績を紹介した長いタイトルの論文です。おそらくここから “Cholesky 分解” の名前が普及したものと思われます。東京大学に所蔵されています。

[2] Bodewig, E. : *Matrix Calculus*, North-Holland Publishing Company-Amsterdam (1956).

— 反復法の歴史が詳しく書いてあります。

[3] Booth, A. D. : *Numerical Methods*, Butterworths, London (1957).

— Hamilton-Cayley の定理に基礎をおく A^{-1} の求め方が載っています。

[4] Bunch, James R., Kaufman, Linda and Parlett, Beresford N. : Decomposition of a Symmetric Matrix : *Numerische Mathematik*, Vol.27, pp.95–109 (1976).

[5] Bunch, James R. and Kaufman, Linda : Some Stable Methods for Calculation Inertia and Solving Symmetric Linear Systems, *Mathematics of Computation*, Vol.31, pp.163–179 (1977).

— [4], [5] では、対称行列に対する安定した数値解法 (MDM^T 分解) を提案しています。具体的なアルゴリズムおよび ALGOL プログラムがついています。

[6] Duff, I.S., Erisman, A. M. and Reid, J. K. : *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford (1986).

— 疎行列の形状の graphics が付録に載っています。行列の形状の図はこれを真似ました。

[7] Dyksen, Wayne. R. and Rice, John R. : The Importance of Scaling for the Hermite Bicubic Collocation Equations, *SIAM Journal on Scientific and Statistical Computing*, Vol.7, No.5, pp.707–719 (1986).

— 数値解析におけるスケーリングの必要性を豊富なデータを駆使して訴えています。

[8] Forsythe, George E. : Gauss to Gerling on Relaxation, *Mathematical Tables and other Aids to Computation* (現在 *Mathematics of Computation*), Vol.5, pp.255–258 (1951).

— Gauss さんが弟子の Gerling さんにあてて書いた手紙の英訳です。

[9] Forsythe, George E. : Tentative Classification of Methods and Bibliography on Solving Systems of Linear Equations, in *Simultaneous Linear Equations and the Determination of Eigenvalues* Edited by L. J. Paige and Oliga Taussky, National Bureau of Standards Applied Mathematics Series 29 (1953).

— 連立 1 次方程式の解法についての総合分類および当時の最新の参考文献の一覧です。あきれるくらい完璧に分類されています。

[10] Forsythe, George E. : Solving Linear Algebraic Equations Can be Interesting, *Bulletin of the American Mathematical Society*, Vol.59 (1953).

— こちらは [9] から興味ある解法をピックアップして、適切な解説を加えた論文です。

[11] Forsythe, George E. and Wasow, Wolfgang R., : *Finite-Difference Methods for Partial Differential Equations*, John Wiley & Sons, New York (1960).
[邦訳] 藤野 精一 訳 : 偏微分方程式の差分法による近似解法 (上),(下), 吉岡書店 (1968).

— 偏微分方程式の数値解法についての古典的名著として知られています。訳者の藤野先生 (我々は "大御大" とお呼びしています) には、修士の頃に御指導いただきました。

[12] Forsythe, George E., Moler, Cleve B., : *Computer Solution of Linear Algebraic Systems*, Prentice-Hall, (1967).
[邦訳] 渋谷 政昭, 田辺 国士 : 計算機のための線形計算の基礎 — 連立 1 次方程式のプログラミング —, 培風館 (1969).

— 連立 1 次方程式の数値解法に関する古典的名著といわれています。序文で「この本を J. H. Wilkinson に捧げたい」と書いてあります。

[13] Forsythe, George E., Malcolm, Michael A. and Moler Cleve B. : *Computer Methods for Mathematical Computations*, Prentice-Hall (1977).
[邦訳] 森 正武 訳 : 計算機のための数値計算法, 日本コンピュータ協会編 コンピュータ・サイエンス研究所シリーズ, 科学技術出版社 (1978).

— 完成したプログラムに重点をおいた解説がされています。 Forsythe さんの遺作です。

[14] Foster, Leslie V. : Gaussian Elimination with Partial Pivoting Can Fail in Practice, *SIAM Journal on Matrix Analysis and Applications*, Vol.15, No.4, pp.1354–1362 (1994).

— 部分ピボット選択付き Gauss の消去法が不安定となる例を実際の問題から導いて紹介しています。

[15] Freund, Roland W., Nachtigal, Nöel M. : An Implementation of the QMR Method Based on Coupled Two-Term Recurrences, *SIAM Journal on Scientific Computing*, Vol.15, No.2, pp.313–337 (1994).

— QMR 法を提案した本人が解説しています。

[16] Girault, Vivette and Raviart, Pierre-Arnaud : *Finite Element Methods for Navier-Stokes equations, Theory and Algorithms*, Springer Series in Computational Mathematics 5, Springer-Verlag (1986).

— Navier-Stokes 方程式に対する有限要素法の解法を解説した、その方面では有名な本です。

[17] Goldstine, Herman H. : *A History of Numerical Analysis from the 16th through the 19th Century*, Studies in the History of Mathematics and Physical Sciences 2, Springer (1977).

— 16 世紀から 19 世紀にかけての数値解析の歴史が書いてあります。

[18] Gregory, R. T. and Karney, D. L. : *A collection of matrices for testing computational algorithms*, John Wiley & Sons, New York (1969).

— 連立 1 次方程式、逆行列、固有値問題などのテスト行列を集めた本です。

[19] Householder, Alston S. : Unitary Triangularization of a nonsymmetric Matrix, *Journal of the Association for Computing Machinery*, Vol.5, pp.339–342 (1958).

— 後に Householder 変換と呼ばれる手法が最初に載った論文です。

[20] Macon, Nathaniel : *Numerical Analysis*, John Wiley & Sons, New York (1963).

— LL^T 分解が "geodetic surveyor" の Cholesky さんによって 1916 年に示された、と書いてある本です。

[21] McCormick, John M. and Salvadori, Mario G. : *Numerical Methods in FORTRAN*, Prentice-Hall (1964).
[邦訳] 清水 留三郎 訳 : FORTRAN による数値計算プログラミング, サイエンスライブラリ 情報電算機 1, サイエンス社 (1970).

— Fortran90 の廃止予定事項に入ってしまった部分がかなりあります。

[22] Meijerink, J. A. and van der Vorst, H. A. : An Iterative Solution Method for Linear Systems of which the Coefficient Matrix is a Symmetric M-Matrix, *Mathematics of Computation*, Vol.31, pp.148–162 (1977).

— 不完全 LU 分解、不完全 Cholesky と共役勾配法の相性のよさを指摘して共役勾配法を復活させた画期的な論文です。ちなみに Bunch さんの論文がこのすぐあとに続きます。

[23] Puschmann, Heinrich and Cortés, Joaquín : The Coordinex Problem and Its Relation to the Conjecture of Wilkinson, *Numerische Mathematik*, Vol.42, pp.291–297 (1983).

— 完全ピボット選択法についての Wilkinson さんの推測を扱った論文です。

[24] Rainsford, Hume F. : *Survey Adjustments and Least Squares*, Constable & Company (1957).

— pp.74 に、「"Commandant Cholesky" の提案した方法が最初に公なった 1924 年の論文は私が英語に訳した」と書いてある本です。そこ以外は読んでないので、内容は知りません。

[25] Rump, Siegfried M. : *Verification methods for dense and sparse system of equations*, Berichte des Forschungsschwerpunktes Informations- und Kommunikationstechnik, TUHH (1993).

— 精度保証付き数値計算を用いた連立方程式の解法を扱っています。

[26] Saad, Youcef, Schultz Martin H. : GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems, *SIAM Journal on Scientific and Statistical Computing*, Vol.7, No.3, pp.856–869 (1986).

— GMRES 法の原論文です。

[27] Stummel, Friedrich : Forward Error Analysis of Gaussian Elimination Part I : Error and Residual Estimates, *Numerische Mathematik*, Vol.46, pp.365–395 (1985).

[28] Stummel, Friedrich : Forward Error Analysis of Gaussian Elimination Part II : Stability Theorems, *Numerische Mathematik*, Vol.46, pp.397–415 (1985).

— [27], [28] は前進誤差解析を扱っています。 Part I, Part II に分かれています。

[29] Trefethen, Lloyd N. : Three Mysteries of Gaussian Elimination, *ACM SIGNUM Newsletter*, Vol.20, No.4, pp.2–5 (1985).

— 計算量、直交変換法との関係、安定性について論じています。

[30] Trefethen, Lloyd N., Schreiber, Robert S. : Average-Case Stability of Gaussian Elimination, *SIAM Journal on Matrix Analysis and Applications*, Vol.11, No.3, pp.335–360 (1990).

— random matrix を用いて不安定因子を計算することで Gauss の消去法の平均的な安定性を主張する論文です。

[31] van der Vorst, H. A. : Bi-CGStab: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems, *SIAM Journal on Scientific and Statistical Computing*, Vol.13, No.2, pp.631–644 (1992).

— BiCGStab 法の原論文です。

[32] Varga, Richard S. : *Matrix Iterative Analysis*, Prentice-Hall, (1962).
［邦訳］ 渋谷 政昭, 棚町 芳弘, 金子 正久, 野田 隆 訳 : 計算機による大型行列の反復解法, サイエンスライブラリ 情報電算機 10, サイエンス社 (1972).

— 連立 1 次方程式の反復解法のアルゴリズムと数学的背景を詳しく論じた古典的名著です。

[33] Wendroff, B., : *Theoretical Numerical Analysis*, Academic Press, New York (1966).
［邦訳］ 戸川 隼人 訳 : 理論数値解析 — 計算機による数値計算の基礎 —, サイエンスライブラリ 情報電算機 19, サイエンス社 (1973).

— 数値解析における理論的な側面を平易に解説した本です。

[34] Westlake, Joan R. : *A Handbook of Numerical Matrix Inversion and Solution of Linear Equations*, John Wiley & Sons, New York (1968).
［邦訳］ 戸川 隼人 訳 : コンピュータのための線形計算ハンドブック, 培風館 (1972).

— 連立 1 次方程式の数値解法をきめ細かく分類したハンドブックです。

[35] Wilkinson, J. H., : Error Analysis of Direct Methods of Matrix Inversion, *Journal of the Association for Computing Machinery*, Vol.8, pp.281–330 (1961).

— Gauss の消去法によって発生する浮動小数点演算の誤差を、後退誤差解析によって評価した古典です。

[36] Wilkinson, J. H., : *Rounding Errors in Algebraic Processes*, Prentice Hall (1963).
［邦訳］一松 信, 四条 忠雄 訳 : 基本的演算における丸め誤差解析, 培風館 (1974).

— 誤差解析の基礎理論を解説した本です。

[37] Wilkinson, J. H., : *The Algebraic Eigenvalue Problem*, Oxford University Press, London (1965).

— それまでの Wilkinson さんの成果を結集して書かれた 600 ページを越える大作です。一時期、数値解析の論文では必ずこの本が参考文献に載せられていました。

[38] Wilkinson, J. H., : 性質の悪い一次方程式の解法, in *Mathematical Methods for Digital Computers II*, Edited by Anthony Ralson, Herbert S. Wilf, John Wiley & Sons, New York (1967).
［邦訳］片岡 信二 監訳 : 電子計算機のための数学的方法 II, 鹿島出版会 (1975).

— Wilkinson さんの方法のダイジェスト版です。

[39] Wright, Stephen J. : A Collection of Problems for which Gaussian Elimination with Partial Pivoting is Unstable, *SIAM Journal on Scientific Computing*, Vol.14, No.1, pp.231–238 (1993).

— 部分ピボット選択付き Gauss の消去法が失敗する例をあげた論文です。

[40] Zeidler, E. : *Nonlinear Functional Analysis and Its Applications*, Springer-Verlag, Berlin (1986).

— 関数解析の世界では有名な本です。たまに証明が間違っていたりします。

....

以下は国内の数値計算に関する文献です。

[41] 伊理 正夫, 藤野 和建 : 数値計算の常識, 共立出版 (1985).

[42] 宇野 利雄 : 計算機のための数値計算, 応用数学力学講座 14, 朝倉書店 (1963).

[43] 唐木 幸比古 : スーパーコンピュータと行列計算, 情報処理, Vol.28, No.11, pp.1441–1451 (1987).

[44] 柴垣 和三雄, 関数解析と数値計算の基礎, 森北出版 (1985).

[45] 島崎 眞昭 : スーパーコンピュータとプログラミング, 計算機科学 / ソフトウェア技術講座 9, 共立出版 (1989).

[46] 数値流体力学編集委員会 編 : 非圧縮性流体解析, 数値流体力学シリーズ 1, 東京大学出版会 (1995).

[47] 杉原 正顕, 室田 一雄 : 数値計算法の数理, 岩波書店 (1994).

[48] 谷口 博 : 機械工学における電子計算機の応用, サイエンスライブラリ情報電算機 34, サイエンス社 (1976).

— Cramer の公式の Fortran プログラムが載っています。

[49] 津田 孝夫 : 数値処理プログラミング, 岩波講座ソフトウェア科学 9, 岩波書店 (1988).

[50] 戸川 隼人 : 数値計算, 情報処理入門コース 7, 岩波書店 (1991).

[51] 戸川 隼人 : マトリクスの数値計算, オーム社 (1971).

[52] 戸川 隼人 : 共役勾配法, シリーズ新しい応用の数学 17, 新曜社 (1977).

[53] 戸川 隼人 : 微分方程式の数値解法 — 有限要素法と差分法 —, オーム社 (1973).

[54] 名取 亮 : 線形計算, すうがくぶっくす 12, 朝倉書店 (1993).

[55] 名取 亮 : 数値解析とその応用, コンピュータ数学シリーズ 15, コロナ社 (1990).

[56] 名取 亮, 野寺 隆 : 大規模行列計算における反復解法, 情報処理, Vol.28, No.11, pp.1452–1459 (1987).

[57] 名取 亮, 野寺 隆 編: スーパーコンピュータと大型数値計算, bit 臨時増刊, 共立出版 (1987).

[58] 一松 信, 戸川 隼人 編 : 数値計算における誤差, bit 臨時増刊, 共立出版 (1975).

[59] 村田 健郎, 小国 力, 唐木 幸比古 : スーパーコンピュータ — 科学技術計算への適用 —, 丸善 (1985).

[60] 森 正武 : 数値解析法, 朝倉現代物理学講座 7, 朝倉書店 (1984).

[61] 森 正武, 杉原 正顕, 室田 一雄 : 線形計算, 岩波講座応用数学 8, 岩波書店 (1994).

[62] 渡部 善隆 : 数値計算と誤差の話, 九州大学大型計算機センター広報, Vol.27, No.5, 556–580 (1994).

[63] 渡部 善隆 : 数値計算と速度の話, 九州大学大型計算機センター広報, Vol.28, No.3, 207–231 (1995).

[64] 線型代数ア・ラ・カルト, 数学セミナー・リーディングス, 数学セミナー増刊, 日本評論社 (1990).

[65] 精度保証付き数値計算とその応用, 情報処理 Vol.31, No.9 (1990).

[66] 情報検索システム AIR —AIR を用いた文献情報の検索 —, 九州大学大型計算機センター (1995).

[67] OS IV/MSP FORTRAN77 EX 使用手引書 V12 用, 79SP-5031 , 富士通株式会社 (1991).

[68] OS IV/MSP FORTRAN77 EX/VP 使用手引書 V12 用, 79SP-5041 , 富士通株式会社 (1991).

[69] SSL II 使用手引書 (科学用サブルーチンライブラリ), 99SP–4020, 富士通株式会社 (1987).

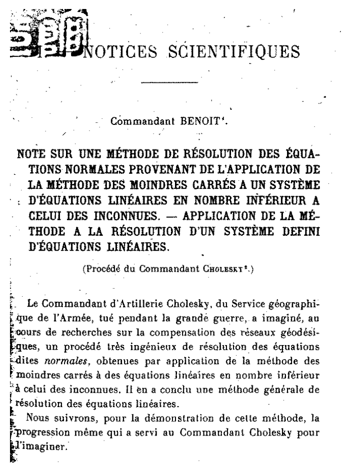
[70] SSL II 拡張機能使用手引書 (科学用サブルーチンライブラリ), 99SP–4070, 富士通株式会社 (1991) .

[71] NUMPAC 利用手引書, 富士通株式会社 (1994) .

[72] 現代数理学事典, 大阪書籍 (1991).

[73] 岩波数学辞典 (第 3 版), 岩波書店 (1985).

[74] アルゴリズム辞典, 共立出版 (1994).



NOTICES SCIENTIFIQUES

Commandant BENOIT[1].

NOTE SUR UNE MÉTHODE DE RÉSOLUTION DES ÉQUATIONS NORMALES PROVENANT DE L'APPLICATION DE LA MÉTHODE DES MOINDRES CARRÉS A UN SYSTÈME D'ÉQUATIONS LINÉAIRES EN NOMBRE INFÉRIEUR A CELUI DES INCONNUES. — APPLICATION DE LA MÉTHODE A LA RÉSOLUTION D'UN SYSTÈME DEFINI D'ÉQUATIONS LINÉAIRES.

(Procédé du Commandant CHOLESKY[2].)

Le Commandant d'Artillerie Cholesky, du Service géographique de l'Armée, tué pendant la grande guerre, a imaginé, au cours de recherches sur la compensation des réseaux géodésiques, un procédé très ingénieux de résolution des équations dites *normales*, obtenues par application de la méthode des moindres carrés à des équations linéaires en nombre inférieur à celui des inconnues. Il en a conclu une méthode générale de résolution des équations linéaires.

Nous suivrons, pour la démonstration de cette méthode, la progression même qui a servi au Commandant Cholesky pour l'imaginer.

Cholesky 法を紹介した 1924 年の論文 ([1])