連立 1 次方程式の Fortran ソルバー GECP

渡部 善隆 *

一般実行列に対する連立 1 次方程式の数値解を完全ピボット選択付き Gauss の消去法により求める Fortran プログラム GECP (Gaussian Elimination with Complete Pivoting) を公開します．プログラムは Fortran 90 規格および 4 倍精度変数に対応している Fortran コンパイラで翻訳可能です．単精度・倍精度・ 4 倍精度のサブルーチンプログラムとともに，少ない引数で手軽に利用できるインターフェイス LINEAR_SOLVE も利用できます．

1 GECP の概要

GECP は "Gaussian Elimination with Complete Pivoting" の略で，一般実行列に対する連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の数値解を完全ピボット選択付き Gauss の消去法により求める Fortran プログラムです．プログラムは 1995 年に執筆した連立 1 次方程式に関する広報記事 [3] に掲載したソースを以降の Fortran 規格にあわせて変更したものです．

以下に GECP の特徴をあげます．

- Fortran 90 以降の規格に準拠した Fortran コンパイラで動作します．
- 単精度版・倍精度版・ 4 倍精度版[1]が利用できます．
- アルゴリズムとして数値的に安定した手法である完全ピボット選択付き Gauss の消去法を採用しているため，他の連立 1 次方程式の解法との比較にも用いることができます．
- オプションの指定により逆反復法による A の最小特異値の近似計算を行ないます．最小特異値を用いて近似解の誤差評価を行なうことができます．
- オプションの指定により $A^T\boldsymbol{x} = \boldsymbol{b}$ の形の方程式を解くことができます．
- 個別の精度 (単精度・倍精度・ 4 倍精度) のサブルーチンを呼び出す他に，組込み関数のように総称名として使用できるインターフェイス LINEAR_SOLVE を利用することができます．
- LINEAR_SOLVE のソースプログラムは既存のサブルーチンライブラリのインターフェイスを作成する雛型としても使えます．

*九州大学情報基盤センター研究部　E-mail: watanabe@cc.kyushu-u.ac.jp

[1] 4 倍精度は正式な Fortran 規格ではありません．しかし，多くの Fortran コンパイラがサポートしています．

2 GECP の利用方法

2.1 ダウンロード

http://www.cc.kyushu-u.ac.jp/RD/watanabe/

からソースプログラムを入手することができます．記載された注意事項に同意した上でダウンロードをお願いします．

2.2 ファイルの構成

GECP プログラムは単精度版・倍精度版・4 倍精度版のサブルーチンプログラムと各サブルーチンを手軽に利用することのできるインターフェイスで構成されています．

以下の手順ではすべてのプログラムが記述されたファイル “GECP.f90” を利用するものとします．なお個別のプログラムをダウンロードすることもできます．

2.3 拡張子

ダウンロードしたソースプログラムの拡張子は “.f90” です．Fortran コンパイラによっては “.f95” や “.f” などへの修正が必要になる場合があります．

2.4 モジュールの宣言

モジュールの宣言は PROGRAM 文に続けて “USE GECP” と記述します．大文字 / 小文字どちらでも構いません．

```
PROGRAM TEST
  USE GECP       ! モジュールの引用
  IMPLICIT NONE
  INTEGER,PARAMETER :: N=8
         :
```

2.5 実行可能ファイルの作成方法

GECP プログラムから翻訳処理によってオブジェクトファイルを作成し主プログラムと結合させる例を紹介します．通常の副プログラムの使用方法と同様に主プログラムと同じファイルにまとめたりアーカイブファイルを作成して管理することも可能です．

以下の実行可能ファイルの作成手順は GECP プログラム “GECP.f90” と主プログラム “main.f90” が同じディレクトリ / フォルダにある場合です[2]．

[2] 実行可能ファイルの作成方法は他にも幾つか考えられます．例えば，主プログラムと GECP ソースをまとめて翻訳する方法などです．ここでは，一度 “GECP.f90” を翻訳しオブジェクトファイルを作成しておくことにより，主プログラムを修正したとしても GECP のソースを再翻訳する必要がないという観点からの利用例を紹介しました．

• UNIX システムでの利用例

HP, Sun, Compaq の Fortran コンパイラの多くは特別なオプションを必要としません．

```
% f90 -c GECP.f90         ← GECP プログラムの翻訳
% f90 main.f90 GECP.o     ←実行可能ファイルの作成
```

作成されるオブジェクトファイル名は “GECP.o”，モジュール名は “gecp.mod” または “GECP.mod” となります．

富士通のコンパイラでインターフェイス LINEAR_SOLVE を利用する場合には，モジュールを翻訳する翻訳時オプション -Am が必要です．

```
% frt -c -Am GECP.f90       ← GECP プログラムの翻訳
% frt -Am main.f90 GECP.o   ←実行可能ファイルの作成
```

作成されるオブジェクトファイル名は “GECP.o”，モジュール名は “gecp.mod” となります．

• Windows 版 Fortran の利用例

手元にある富士通の Windows 版 Fortran & C Academic Package V3.0 での実行可能ファイルの作成例は次のようになります．

```
% frt /c /Am GECP.f90       ← GECP プログラムの翻訳
% frt /Am main.f90 GECP.o   ←実行可能ファイルの作成
```

/c はオブジェクトファイルを作成するまでの処理を行なうオプション，/Am はモジュールを翻訳することを指定するオプションです．作成されるオブジェクトファイル名は “GECP.obj”，モジュール名は “gecp.mod” となります．

2.6 LINEAR_SOLVE

LINEAR_SOLVE は GECP を手軽に利用するための関数です．

◆機能

連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の近似解を求める．

◆書式

```
LINEAR_SOLVE(A,B,[,EPS=eps][,SV=sv][,DELTA=delta])
```

◆引数

A	行列 A．実数型 2 次元配列．行および列の大きさが B の大きさ以上であること．
B	ベクトル $\boldsymbol{b}$．実数型 1 次元配列．B の大きさが方程式の次数となる．
EPS=eps	行列の非正則性判定定数．省略可能．選択されたピボットの絶対値と A の要素の絶対値最大との相対比が eps 以下になった場合，行列は数値的に非正則と判断し処理を打ち切る．省略した場合，マシンエプシロンが採用される．
SV=sv	A の最小特異値を計算し sv に返却する．省略可能．
DELTA=$delta$	SV=sv が指定された場合の逆反復法の収束判定条件を指定．省略可能．省略した場合，単精度で 1.0E-5，倍精度で 1.0E-10，4 倍精度で 1.0E-20 が設定される．

◆戻り値

$\boldsymbol{x}$の近似解．B と同じ大きさの実数型 1 次元配列．

◆注意

引数の精度は同じでなければならない．例えば A, B が倍精度で eps が単精度の場合には主プログラム翻訳時にエラーとなる．

● LINEAR_SOLVE の使用例 1

10×10 の単精度実行列 A とベクトル b を Fortran の組込み関数 RANDOM_NUMBER で作成し，連立 1 次方程式の解を x に格納します．

```
program test                                   ! プログラム文
   use gecp                                    ! モジュールの宣言
   real,dimension(10,10) :: A                  ! A の宣言
   real,dimension(10)    :: b,x                ! b,x の宣言
   call random_number(A)                       ! A の生成
   call random_number(b)                       ! b の生成
   x=linear_solve(A,b)                         ! Ax=b を解く
   write(*,*) x                                ! x の出力
end program test                               ! プログラムの終了
```

ポイントは 2 行目のモジュールの宣言 “use gecp” と 6 行目の LINEAR_SOLVE の呼び出しです．Fortran は大文字 / 小文字を区別しないため，ソースは大文字で記述しても構いません．

● LINEAR_SOLVE の使用例 2

次のプログラムは http://www.cc.kyushu-u.ac.jp/RD/watanabe/ からダウンロード可能な倍精度のテストプログラムを一部修正したものです．

テスト行列は $A^{-1} = A$ となります[3]．ベクトル b は Fortran の組込み関数 RANDOM_NUMBER で作成し，$A^{-1}\boldsymbol{b} = A\boldsymbol{b}$ を真の解として sol に格納しています．

プログラム文の直後にモジュール "GECP" を使用することを宣言します．LINEAR_SOLVE は ★1, ★2 で呼び出しています．

★1 は連立 1 次方程式を解き結果を x1 に返却する処理です．★2 は連立 1 次方程式を解き結果を x2 に返却する処理とともに最小特異値を変数 s に返却します．最小固有値を計算するオプションを指定する場合 "sv=" または "SV=" の記述が必要です．値を返却する変数名 (例では s) は任意です．

LINEAR_SOLVE では A と b の値は保存され，書き換えられることはありません．

```
program linear_solve_test
   use GECP                                                   ! モジュールの宣言
   implicit none

   integer                                       :: n
   real(kind(1D0)),dimension(:,:),allocatable :: A
   real(kind(1D0)),dimension(:),allocatable   :: x1,x2,b,sol
   integer                                       :: i,j
   real(kind(1D0))                               :: Pi,s

   n=1000                                                     ! 次数の決定（調整可）
   allocate(A(n,n),x1(n),x2(n),b(n),sol(n))                   ! 配列の宣言

   Pi=4*atan(1.0D0)                                           !--+
   A=0                                                        !  |
   do j=1,n                                                   !  |
     do i=1,n                                                 !  +-- 行列 A の生成
       A(i,j)=sqrt(2.0D0/(n+1))*sin((i*j*Pi)/(n+1))           !  |
     end do                                                   !  |
   end do                                                     !--+

   call random_number(b)                                      ! 右辺 b の生成
   sol=matmul(A,b)                                            ! 真の解の設定

   x1=linear_solve(A,b)                                       ! Ax=b を解く ★1
   x2=linear_solve(A,b,sv=s)                                  ! Ax=b を解く ★2
                                                              ! 最小特異値も計算

   write(6,*)                                                 ! 誤差，特異値の出力
   write(6,*) 'dimension=',N
   write(6,*) '               maximum of the solution=',maxval(x1)
   write(6,*) '        relative error of linear solve=',&
                     maxval(abs(x1-sol))/maxval(abs(sol))
   write(6,*) ' singular value=',s

   deallocate(A,x1,x2,b,sol)

end program linear_solve_test
```

[3] 浮動小数点演算数が実数の有限桁の近似になることから，《厳密な意味では》一致しません．もちろん，十分よい近似です．

2.7 GECP_S，GECP_D，GECP_Q

◆機能

連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の近似解を求める．

◆書式

```
GECP_S(A,NX,N,B,EPS,ISW,P,Q,VP,VS,VW,ICON)
GECP_D(A,NX,N,B,EPS,ISW,P,Q,VP,VS,VW,ICON)
GECP_Q(A,NX,N,B,EPS,ISW,P,Q,VP,VS,VW,ICON)
```

(ただし A, B, EPS, VP, VS, VW は対応する実数型で宣言すること)

◆引数

A	係数行列 A．大きさ NX × NX の実数型 2 次元配列． LU 分解された下三角行列 L と上三角行列 U が返却される (注意事項参照)．
NX	A の整合寸法．整数型．N 以上．
N	方程式の次数．整数型．
B	ベクトル $\boldsymbol{b}$．大きさ N の実数型 1 次元配列．近似解 $\boldsymbol{x}$ が返却される．
EPS	行列の非正則性判定定数．実数型．正規化後のピボット選択の際，決定されたピボットの絶対値が EPS 以下の場合，行列は数値的に特異だと見なし処理を打ち切る．0 以下が指定された場合はマシンエプシロンが採用される．
ISW	処理方法を指定する．整数型． 0 … LU 分解のみを行なう． 1 … $A\boldsymbol{x} = \boldsymbol{b}$ を解く． 2 … LU 分解を省略して $A\boldsymbol{x} = \boldsymbol{b}$ を解く． 3 … $A^T\boldsymbol{x} = \boldsymbol{b}$ を解く． 4 … LU 分解を省略して $A^T\boldsymbol{x} = \boldsymbol{b}$ を解く． その他 … 1 が指定されたとみなす．
P	行に関するピボット選択情報が保存される．大きさ N の整数型 1 次元配列．
Q	列に関するピボット選択情報が保存される．大きさ N の整数型 1 次元配列．
VP	変数の正規化に関する情報が保存される．大きさ N の実数型 1 次元配列．
VS	方程式の正規化に関する情報が保存される．大きさ N の実数型 1 次元配列．
VW	作業用ベクトル．大きさ N の実数型 1 次元配列．
ICON	計算が正常に行なわれたかどうかを判定する．整数型． 0 … 正常終了． 2000 … A のある行または列の要素がすべて零であったかまたはピボットの絶対値が EPS 以下になった．係数行列は非正則の可能性が強い． 3000 … パラメータの入力ミス．

◆注意事項

1. ISW の値が $2, 4$ 以外の場合，入力された係数行列 A の値は保存されず LU 分解後の行列に置き換わる．

2. LU 分解されて返却される係数行列 A には，

   - 上三角成分に U の要素
   - 対角成分は U の対角成分の逆数
   - 下三角成分に L の要素

   が格納されている．下三角行列 L の対角成分はすべて 1 であり，これらは既知の値として保存されない．LU 分解された行列を GECP プログラム以外で利用する場合には上記格納方法に注意する必要がある．

3. 引数の実数型はサブルーチンに合わせて宣言すること．例えば A, B が倍精度で 4 倍精度の GECP_Q を呼び出した場合には結果は保証されない[4]．

●使用例

10×10 の倍精度実行列 A とベクトル b を Fortran の組込み関数 RANDOM_NUMBER で作成し，連立 1 次方程式の解を GECP_D により求めます．モジュールの宣言は不要です．

```
PROGRAM TEST2                                           ! プログラム文
   IMPLICIT NONE                                        ! 暗黙の型宣言の抑止
   INTEGER,PARAMETER     :: N=10                        ! パラメータの設定
   REAL(KIND(1D0)),DIMENSION(N,N) :: A                  ! A の宣言
   REAL(KIND(1D0)),DIMENSION(N)   :: B,VP,VS,VW         ! ベクトルの宣言
   REAL(KIND(1D0))                :: EPS                ! 特異性判定数の宣言
   INTEGER,DIMENSION(N)           :: P,Q                ! 整数型ベクトルの宣言
   INTEGER                        :: ISW,ICON           ! 整数型変数の宣言

   CALL RANDOM_NUMBER(A)                                ! A の生成
   CALL RANDOM_NUMBER(B)                                ! B の生成

   EPS=0                                                ! 特異性判定数の設定
   ISW=1                                                ! 処理種別を設定
   CALL GECP_D(A,N,N,B,EPS,ISW,P,Q,VP,VS,VW,ICON)       ! GECP_D の呼び出し

   WRITE(*,*) ICON                                      ! 処理状況の確認

   WRITE(*,*) B                                         ! 解の出力
END PROGRAM TEST2                                       ! プログラムの終了
```

[4] 翻訳時にエラーとならず，最悪の場合は「それらしい結果」が返ってくることもあります．サブルーチンの引数の型の記述の確認はデバックの基本です．

3 GECP で用いている Fortran の機能

ここでは LINEAR_SOLVE に使用している Fortran の機能を簡単に紹介します．

3.1 動的配列割り当て

LINEAR_SOLVE ではプログラムの実行中に配列の大きさを決める動的配列割り当て機能を用いています．配列は ALLOCATABLE 属性を使って宣言し，ALLOCATE 文により配列の大きさを決定します．不要になった配列は DEALLOCATE 文によって解放します．この機能を用いることにより，GECP_S, GECP_D, GECP_Q での作業用配列を宣言する手間を省いています．

以下は動的配列 vp, vs, vw, ip, iq を宣言し，何らかの方法で決定された n によって配列の大きさを割り当て，サブルーチン GECP_D に作業用配列として渡すプログラムの抜粋です．

```
real(kind(1D0)),dimension(:),allocatable :: vp,vs,vw ! 配列の宣言
        integer,dimension(:),allocatable :: ip,iq
        :
allocate(vp(n),vs(n),vw(n),ip(n),iq(n))              ! 大きさの決定
        :
call GECP_D(Z,k1,n,x,epsz,isw,ip,iq,vp,vs,vw,icon)   ! 作業用配列の引き渡し
        :
deallocate(vp,vs,vw,ip,iq)                           ! 配列の解放
        :
```

3.2 省略可能引数

引数が省略可能であるものを OPTIONAL 属性によって宣言することで，サブルーチンに省略値を渡すことができます．利用者は自身でパラメータまたは手法を選択する必要のある場合にのみ引数を指定します．その場合キーワードを用いた引数を書くことができます．利用者が引数を指定したかどうかの判断は組込み関数 PRESENT で知ることができます．

```
function linear_solve_s(A,b,eps,sv,delta) result(x)
                 :
  real(kind(1E0)),intent(in),optional      :: eps
  real(kind(1E0)),intent(out),optional     :: sv
  real(kind(1E0)),intent(in),optional      :: delta
  real(kind(1E0))                          :: epsz=0.0E0,deltaz=1.0E-5
                :
  if(present(eps))   epsz=eps
  if(present(delta)) deltaz=delta
                :
  call GECP_S(Z,k1,n,x,epsz,isw,ip,iq,vp,vs,vw,icon)
                :
  if(present(sv)) then
     allocate(y(n),u(n))
                :
```

この例では関数 linear_solve_s の引数 eps, sv および delta が省略可能です．利用者が引数を指定する場合にはキーワード引数として以下の 3 つの例のように指定します．

```
x=linear_solve_s(A,b,eps=1.0E-5)
```

```
x=linear_solve_s(A,b,sv=t)
```

```
x=linear_solve_s(A,b,sv=t,delta=1.0E-3,eps=1.0E-2)
```

ただし LINEAR_SOLVE では，次節の総称名を用いることで型に依存しないインターフェイスにしています．

3.3 総称名

組込み手続きの数学関数として利用できる ABS, MATMUL などは引数の型によらず同一の名前 (総称名) で記述することができます．利用者作成の手続きに対しても，INTERFACE 文を用いることにより複数の関数を同じ総称名として呼び出すことができます．

次の例では総称名 LINEAR_SOLVE により単精度型・倍精度型・4 倍精度型の関数を型と精度に応じてそれぞれ呼び出すように指定しています．個別の関数副プログラム部分には引数と結果の宣言を記述します．

```
interface linear_solve                                 ! 総称名の指定
     function linear_solve_s(A,b,eps,sv,delta) result(x) ! 単精度実数型
        real(kind(1E0)),dimension(:,:),intent(in) :: A
        real(kind(1E0)),dimension(:),intent(in)   :: b
        real(kind(1E0)),intent(in),optional       :: eps
        real(kind(1E0)),intent(out),optional      :: sv
        real(kind(1E0)),intent(in),optional       :: delta
        real(kind(1E0)),dimension(size(b))        :: x
     end function linear_solve_s

     function linear_solve_d(A,b,eps,sv,delta) result(x) ! 倍精度実数型
        real(kind(1D0)),dimension(:,:),intent(in) :: A
        real(kind(1D0)),dimension(:),intent(in)   :: b
        real(kind(1D0)),intent(in),optional       :: eps
        real(kind(1D0)),intent(out),optional      :: sv
        real(kind(1D0)),intent(in),optional       :: delta
        real(kind(1D0)),dimension(size(b))        :: x
     end function linear_solve_d

     function linear_solve_q(A,b,eps,sv,delta) result(x) ! 4 倍精度実数型
        real(kind(1Q0)),dimension(:,:),intent(in) :: A
        real(kind(1Q0)),dimension(:),intent(in)   :: b
        real(kind(1Q0)),intent(in),optional       :: eps
        real(kind(1Q0)),intent(out),optional      :: sv
        real(kind(1Q0)),intent(in),optional       :: delta
        real(kind(1Q0)),dimension(size(b))        :: x
     end function linear_solve_q
end interface
```

3.4 モジュール

総称名として記述した関数副プログラムは，モジュールと呼ばれる副プログラムにまとめて記述することにより他のプログラム単位から参照することができます．

```
module GECP                  ! モジュールの宣言
  interface linear_solve     ! 総称名の指定
        :
  end interface
end module GECP
```

モジュールに記述した関数副プログラムを利用するには，USE 文を用います．

4 GECP のアルゴリズム

ここでは GECP で用いられている計算手法を紹介します．

4.1 正規化

GECP では LU 分解の計算の前に正規化 (行列の均等化) と呼ばれる操作を行ないます．まず連立 1 次方程式

$$A\boldsymbol{x} = \boldsymbol{b}$$

の係数行列 $A = [a_{ij}]$ に対し，i 行の絶対値最大の逆数を対角成分に持つ行列 V を作成します．

$$V = \mathrm{diag}\{\max_{1\le j\le n} |a_{1j}|, \ldots, \max_{1\le j\le n} |a_{nj}|\}^{-1}$$

そして方程式の両辺の左から V を作用させます．

$$VA\boldsymbol{x} = V\boldsymbol{b}.$$

次に係数行列 $VA = [c_{ij}]$ に対し，i 列の絶対値最大の逆数を対角成分に持つ行列 W を作成します．

$$W = \mathrm{diag}\{\max_{1\le i\le n} |c_{i1}|, \ldots, \max_{1\le i\le n} |c_{in}|\}^{-1}.$$

そして $\boldsymbol{x} = W\boldsymbol{y}$ と変換することで方程式を同値変形します．

$$VAW\boldsymbol{y} = V\boldsymbol{b}, \quad \boldsymbol{x} = W\boldsymbol{y}.$$

係数行列 VAW はどの要素も絶対値が 1 以下であって，どの行にも，どの列にも少なくとも一つの絶対値 1 の要素が含まれます．

4.2 完全ピボット選択付き LU 分解

正規化によって作られた行列 VAW を，下三角行列 L と上三角行列 U に分解します．

$$PVAWQ = LU.$$

ここで P と Q は置換行列です．置換行列とは要素がすべて 1 か 0 のどちらかで，1 の要素がどの行も，どの列も，ちょうど 1 箇所ある行列のことです．行列 A に対して置換行列を左から作用させるということは，A の行の順序を並べかえることに，同様に置換行列を右から作用させるということは列の順序を並べかえることに相当します．

4.2.1 Gauss の消去法による LU 分解

LU 分解の方法は数学的に同値ないくつかの手法があります．GECP では Gauss の消去法のアルゴリズムによる LU 分解を行なっています．一般の実行列 $A=[a_{ij}]$ を LU 分解するアルゴリズムは以下の通りです．

Gauss の消去法による LU 分解

$1 \le k \le n-1,\ \ i \ge k+1,\ \ j \ge k+1$ に対し次を計算.

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

$$L=\begin{pmatrix} 1 & & & & \\ m_2^{(1)} & 1 & & 0 & \\ m_3^{(1)} & m_3^{(2)} & 1 & & \\ \vdots & \vdots & & \ddots & \\ m_n^{(1)} & m_n^{(2)} & \cdots & m_n^{(n-1)} & 1 \end{pmatrix},\quad U=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ & & \cdots & \vdots \\ 0 & & & a_{nn}^{(n-1)} \end{pmatrix}.$$

4.2.2 ピボット選択

LU 分解の過程:

$$m_i^{(k)} = a_{ik}^{(k-1)}/\underline{a_{kk}^{(k-1)}}$$

における要素 $a_{kk}^{(k-1)}$ をピボットと呼びます．丸め誤差の混入を防ぐ安定した計算のためには，適切なピボットを選択する必要があります．よく行なわれるピボットの選び方は，k 回目の消去過程で列成分の絶対値最大を探して，行を入れ換える (P を左から作用させる) という操作です．この操作は部分ピボット選択と呼ばれます．

⋄ 部分ピボット選択: $\max\limits_{k\le i\le n} |a_{ik}^{(k-1)}|$

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & \boxed{a_{22}^{(1)}} & \cdots & a_{2n}^{(1)} \\ \vdots & \vdots & \cdots & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{pmatrix} \longleftarrow \text{対象となる成分 } (k=2)$$

GECP で採用している完全ピボット選択は，ピボットを消去の対象となる行と列すべての要素から選択する方法です．この場合，行と列の入れ換えが必要になるため，置換行列 Q を右から作用させる必要があります．

◇ 完全ピボット選択: $\max_{k\le i,j\le n}|a_{ij}^{(k-1)}|$

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & \vdots & \cdots & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{pmatrix} \longleftarrow \text{対象となる成分 } (k=2)$$

4.3 前進 / 後退代入

係数行列が LU 分解された方程式は，行列の分解に比べごくわずかな手間で解くことができます．簡単のため正規化とピボット選択の過程は省略して，行列 A が下三角行列 L と上三角行列 U によって既に $A = LU$ と分解されているとした $A\boldsymbol{x} = \boldsymbol{b}$ の求め方について説明します． $A = LU$ と分解されていることからまず

$$L\boldsymbol{z} = \boldsymbol{b}$$

となる $\boldsymbol{z} = [z_i]$ を求め，それから

$$U\boldsymbol{x} = \boldsymbol{z}$$

を解きます．前者を前進代入，後者を後退代入と呼びます．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．

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

後退代入計算

$i = n, \cdots, 1$ の順に次を計算．

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

同様の考え方で A の LU 分解が完了しているならば，方程式

$$A^T\boldsymbol{x} = \boldsymbol{b}$$

も，まず $U^T\boldsymbol{z} = \boldsymbol{b}$ を解き，次に $L^T\boldsymbol{x} = \boldsymbol{z}$ を解くことで求めることができます． $A^T\boldsymbol{x} = \boldsymbol{b}$ は次節の最小特異値の計算で使用します．

4.4 最小特異値の計算

4.4.1 特異値分解

任意の $n \times n$ 実行列 A は直交行列 C_1, C_2 ($C_1 C_1^T = C_2 C_2^T = I$; I は単位行列) によって

$$A = C_1 D C_2^T$$

の形に分解することができます[5][2]．ここで D は対角行列

$$D = \mathrm{diag}\{\sigma_1, \ldots, \sigma_n\}$$

です．この分解を特異値分解と呼びます．σ_i $(1 \leq i \leq n)$ を A の特異値と呼びます．A が正則ならばすべての i において $\sigma_i > 0$ となります．

4.4.2 連立 1 次方程式の誤差評価

もし A の特異値分解が得られるならば，連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の近似解の正確な誤差評価が得られます．$A\boldsymbol{x} = \boldsymbol{b}$ の近似解を $\hat{\boldsymbol{x}}$，残差を $\boldsymbol{r} := A\hat{\boldsymbol{x}} - \boldsymbol{b}$，$C_1$ を構成する列ベクトルを

$$C_1 = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_n]$$

とすれば，

$$\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_2 = \sqrt{\sum_{i=1}^{n} \frac{1}{\sigma_i^2}(\boldsymbol{u}_i, \boldsymbol{r})^2}$$

という評価が成り立つからです[6]．ここで $\|\cdot\|_2$ は Euclid ノルム，$(\cdot,\cdot)$ はベクトルの内積です．

通常，行列の特異値分解は連立 1 次方程式を解く以上にコストがかかります．したがって連立 1 次方程式誤差評価のためだけに D と C_1 を求めることはあまり現実的ではありません．しかしながら，正則な A の最小特異値 $\sigma_{\min} > 0$ が分かるならば，不等式評価

$$\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_2 \leq \frac{1}{\sigma_{\min}}\|\boldsymbol{r}\|_2$$

を導くことができます[7]．GECP では `LINEAR_SOLVE` のオプションとして A の最小特異値の近似解を計算します．もちろん数値的な近似解ですので厳密な誤差評価が得られる訳ではありません．しかし，

- 残差ノルム $\|\boldsymbol{r}\|_2$ を計算することによって連立 1 次方程式の数値解の誤差をある程度見積もることができる
- A の特異性 ($\sigma_{\min}$ が零に近い場合，特異性が高いと判断できます) を調べることができる

などの利点があります．

[5]一般の $m \times n$ 行列でも可能です．
[6]$\boldsymbol{x} - \hat{\boldsymbol{x}} = A^{-1}\boldsymbol{r}$ として A の特異値分解を用います．
[7]誤差評価式を最小特異値で括り出して C_1 の直交性を用います．

4.4.3 逆反復法

最小特異値の計算は対称行列の最小固有値の計算によく使われる逆反復法のアルゴリズムを採用しました．

逆反復法による最小特異値計算

set an initial guess $\boldsymbol{x}$ such that $\|\boldsymbol{x}\|_2 = 1$
repeat
  solve $A\boldsymbol{y} = \boldsymbol{x}$
  solve $A^T\boldsymbol{z} = \boldsymbol{y}$
  $\boldsymbol{\xi} = \boldsymbol{z}/\|\boldsymbol{z}\|_2$
  if $\|\boldsymbol{\xi} - \boldsymbol{x}\|_\infty/\|\boldsymbol{\xi}\|_\infty \le \varepsilon$
    $\gamma = 1/\sqrt{\|\boldsymbol{z}\|_2}$ and quit
  end if
  $\boldsymbol{x} = \boldsymbol{\xi}$
end

アルゴリズム中の $\|\cdot\|_\infty$ は最大値ノルム (要素の絶対値最大) です．他のノルムを採用することも可能です．$\varepsilon > 0$ は逆反復法の収束判定を行なうパラメータです．`LINEAR_SOLVE` ではオプション `DELTA` で変更することができます．逆反復法の最大反復回数は 100 回です．これ以上の反復を行ないたい場合は，ソースプログラムの整数型変数 `KMAX` の値を変更してください．

5 性能評価

ここでは GECP の実行性能について報告します．

5.1 正規化・ピボッティングの精度比較

GECP と正規化およびピボッティングを行なわないまたは一部しか行なわない場合との計算精度を比較しました．比較に用いた手法は以下の表の通りです．“—” は正規化またはピボッティングを行なわなかった場合を，A に行列の作用があるものは対応する操作を行なったことをそれぞれ意味します．

手法	正規化	ピボッティング
GECP01	—	—
GECP02	$A*W$	—
GECP03	$V*A$	—
GECP04	$V*A*W$	—
GECP05	—	$P*A$
GECP06	$A*W$	$P*A$
GECP07	$V*A$	$P*A$
GECP08	$V*A*W$	$P*A$
GECP09	—	$P*A*Q$
GECP10	$A*W$	$P*A*Q$
GECP11	$V*A$	$P*A*Q$
GECP	$V*A*W$	$P*A*Q$

5.1.1　直交行列

$n = 1000$ として，A, b を

$$A_{ij} = \sqrt{2/(n+1)}\sin(ij\pi)/(n+1),$$

$$b_i = \sum_{j=1}^{n} A_{ij} \tag{1}$$

で作成しました．解は $\boldsymbol{b}$ 作成時の丸め誤差を無視すれば $\boldsymbol{b} = \boldsymbol{1}$ になります．精度の比較は計算結果 $\boldsymbol{x}$ に対する最大値ノルムの誤差:

$$\|\boldsymbol{x} - \boldsymbol{1}\|_\infty = \|\boldsymbol{x} - \boldsymbol{1}\|_\infty / \|\boldsymbol{1}\|_\infty \tag{2}$$

で与えています．数字は最後の表示桁を切捨てています．

計算は汎用 UNIX サーバ GP7000F で行ないました．“SSL II” は富士通提供のサブルーチンライブラリ SSL II [7] の VLAX(単精度) および DVLAX(倍精度) を用いました．SSL II は 4 倍精度が未サポートのため 4 倍精度部分は “—” で表示しています．

手法	単精度	倍精度	4 倍精度
GECP01	$*$	$*$	1.31×10^{8}
GECP02	$*$	4.31×10^{9}	1.46×10^{9}
GECP03	$*$	6.85×10^{7}	4.51×10^{7}
GECP04	27.4	4.08×10^{7}	6.22×10^{7}
GECP05	1.80×10^{-5}	2.69×10^{-14}	2.73×10^{-32}
GECP06	3.07×10^{-5}	4.80×10^{-14}	2.88×10^{-32}
GECP07	2.42×10^{-5}	3.74×10^{-14}	2.99×10^{-32}
GECP08	1.04×10^{-4}	1.35×10^{-12}	1.49×10^{-30}
GECP09	5.43×10^{-5}	1.19×10^{-13}	6.80×10^{-32}
GECP10	3.50×10^{-5}	5.75×10^{-14}	4.33×10^{-32}
GECP11	3.45×10^{-5}	6.03×10^{-14}	4.68×10^{-32}
GECP	5.35×10^{-5}	5.06×10^{-14}	3.94×10^{-32}
SSL II	1.45×10^{-5}	3.13×10^{-14}	—

表中の “$*$” は，消去の過程でピボットが数値的にゼロと判定されて処理が打ち切られたことを意味します．これらの結果から一般の行列に対してはピボット選択が不可欠であることがわかります．

5.1.2　一様乱数

$n = 100$ として，$[-1, 1]$ の一様乱数を生成する組込み関数 `RANDOM_NUMBER` により行列 A を作成しました，右辺は (1) 式で，誤差は (2) 式で評価します．連立 1 次方程式の誤差は 100 回分の平均を取っています．

手法	単精度	倍精度	4 倍精度
GECP01	1.92×10^{-2}	5.16×10^{-11}	1.44×10^{-29}
GECP02	1.67×10^{-2}	4.14×10^{-11}	9.56×10^{-30}
GECP03	7.18×10^{-3}	5.42×10^{-11}	9.49×10^{-30}
GECP04	8.86×10^{-3}	1.00×10^{-10}	1.14×10^{-29}
GECP05	6.66×10^{-5}	1.49×10^{-13}	9.02×10^{-32}
GECP06	6.72×10^{-5}	7.39×10^{-13}	8.69×10^{-32}
GECP07	5.59×10^{-5}	7.63×10^{-13}	1.04×10^{-31}
GECP08	7.03×10^{-5}	7.09×10^{-13}	8.83×10^{-32}
GECP09	4.40×10^{-5}	5.69×10^{-13}	7.83×10^{-32}
GECP10	5.60×10^{-5}	2.52×10^{-13}	5.78×10^{-32}
GECP11	4.71×10^{-5}	3.43×10^{-13}	6.25×10^{-32}
GECP	3.88×10^{-5}	9.32×10^{-13}	8.11×10^{-32}
SSL II	5.98×10^{-5}	2.11×10^{-12}	—

この場合もピボット選択を行なわない場合には精度が悪くなることがわかります．

5.1.3 Foster の行列

最後に Foster さんが 1994 年に報告した，積分方程式の差分近似から導かれる連立 1 次方程式に対して部分ピボット選択が失敗する例 [1] で実験しました． $n = 500$ です．

手法	単精度	倍精度	4 倍精度
GECP01	2.17×10^{6}	5.23	5.55×10^{-18}
GECP02	1.11×10^{9}	3.06	4.24×10^{-18}
GECP03	1.11×10^{9}	3.07	1.93×10^{-18}
GECP04	5.44×10^{5}	1.00	2.32×10^{-18}
GECP05	2.17×10^{6}	5.23	5.55×10^{-18}
GECP06	1.11×10^{9}	3.06	4.24×10^{-18}
GECP07	1.11×10^{9}	3.07	1.93×10^{-18}
GECP08	3.39×10^{-6}	8.32×10^{-15}	6.74×10^{-33}
GECP09	3.33×10^{-6}	3.44×10^{-15}	3.65×10^{-33}
GECP10	4.05×10^{-6}	4.44×10^{-15}	4.81×10^{-33}
GECP11	3.57×10^{-6}	6.21×10^{-15}	6.74×10^{-33}
GECP	2.98×10^{-6}	6.21×10^{-15}	5.20×10^{-33}
SSL II	2.14×10^{9}	5.00	—

ピボット選択を行なわない場合，および部分ピボット選択の場合は，単精度・倍精度ではほとんど意味のない解を出力しています． 4 倍精度でも極端に精度が落ちています．ただし部分ピボット選択でも，あらかじめ V を左から W を右から作用させる正規化を行なった場合には GECP と同等の精度が得られています．この例によって正規化の重要性がお分かりになるかと思います．

5.2 各ステップ毎の実行時間

GECP の各処理毎の CPU 時間を測定してみました．計算機は FUJITSU VPP5000/64 の 1PE，行列 A とベクトル $\boldsymbol{b}$ は RANDOM_NUMBER で生成しました． 2002 年 6 月 6 日の測定，翻訳時オプションは -Kfast，精度は倍精度です．比較のため SSL II [6] のサブルーチン DASVD1 による A の特異値分解の時間も示します．

次数	2500×2500		5000×5000	
方程式の正規化	0.05 秒	(0.72%)	0.37 秒	(0.69%)
未知数の正規化	0.01 秒	(0.14%)	0.03 秒	(0.05%)
LU 分解	6.82 秒	(98.69%)	52.54 秒	(98.96%)
$A\boldsymbol{x}=\boldsymbol{b}$ を解く	0.02 秒	(0.28%)	0.13 秒	(0.24%)
$A^T\boldsymbol{x}=\boldsymbol{b}$ を解く	0.01 秒	(0.14%)	0.02 秒	(0.03%)
特異値分解	25.04 秒		448.87 秒	

この結果から GECP の実行時間のほとんどは LU 分解に集中すること，逆に LU 分解が終了した行列を持っていれば，わずかな時間で解が求まることがわかります．

5.3 インターフェイスのオーバーヘッド

組込み関数的な利用のできるインターフェイス LINEAR_SOLVE は，内部で作業用の行列を動的に割り当て，配列のコピーなどを行なっています．この部分のオーバーヘッドが，サブルーチンライブラリをそのまま利用する場合と比較してどれくらいかかるのかを測定しました．

計算機は汎用 UNIX サーバ GP7000F/900, スカラー並列サーバ GS320, スーパーコンピュータ VPP5000/64 です．単位は秒, 測定日は 2002 年 6 月 11 日です．行列は 5.1.1 節の直交行列を用いました． $n = 2000$ です．翻訳時オプションは GP7000F/900, VPP5000/64 が -Kfast，GS320 が -fast です．

型	手法	GP7000F/900	GS320	VPP5000/64
単精度	LINEAR_SOLVE	105.02	90.54	3.34
	DECP_S	104.93	90.42	3.34
倍精度	LINEAR_SOLVE	237.00	158.20	3.48
	DECP_D	234.47	158.01	3.50
4 倍精度	LINEAR_SOLVE	3550.83	631.53	183.92
	DECP_Q	3545.20	630.71	183.83

この結果から LINEAR_SOLVE を用いる場合のオーバヘッドは無視できることがわかります．もちろん，一度 LU 分解した行列を再利用する場合や小さな次元の方程式を何回も繰り返して解く場合などには，各サブルーチンを個別に用いる方が効率がよいことがあります．

5.4 速度比較

最後に，他のサブルーチンライブラリとの速度比較のデータを紹介します．比較に用いたサブルーチンは次の通りです．

	単精度	倍精度	4 倍精度	参考文献
SSL II	VLAX	DVLAX	—	[7]
NUMPAC	LEQLUS	LEQLUD	LEQLUQ	[4]
LAPACK	SGESV	DGESV	—	[5]

A, $\boldsymbol{b}$ は RANDOM_NUMBER で生成し，次数 $n = 1000$, $n = 2000$ に対して数値解が得られるまでの CPU 時間を Fortran 組込みサブルーチン CPU_TIME で測定しました．翻訳時オプションは -Kfast，2002 年 6 月 12 日に測定しました．表中の単位は秒です．

NUMPAC, LAPACK の欄の数値は，ソースプログラムを翻訳して得られた測定値，“built-in” は，メーカー提供のライブラリを利用したものです．

手法	単精度	倍精度	4 倍精度
GECP	8.22	11.34	423.22
SSL II	1.27	1.67	—
NUMPAC	20.52	59.28	441.56
[built-in]	20.25	58.23	438.16
LAPACK	3.11	4.34	—
[built-in]	1.24	1.64	—

GP7000F/900 (次数 1000)

手法	単精度	倍精度	4 倍精度
GECP	104.54	235.65	3521.06
SSL II	9.75	13.34	—
NUMPAC	541.63	670.67	3837.23
[built-in]	542.11	662.55	3811.85
LAPACK	26.47	36.52	—
[built-in]	9.68	13.06	—

GP7000F/900 (次数 2000)

手法	単精度	倍精度	4 倍精度
GECP	0.55	0.56	23.44
SSL II	0.07	0.10	—
NUMPAC	1.18	1.48	45.15
[built-in]	7.45	15.33	507.25
LAPACK	0.48	0.50	—
[built-in]	0.09	0.11	—

VPP5000/64 (次数 1000)

手法	単精度	倍精度	4 倍精度
GECP	3.34	3.42	184.94
SSL II	0.64	0.70	—
NUMPAC	10.47	17.18	345.22
[built-in]	69.87	141.46	4023.06
LAPACK	2.22	2.25	—
[built-in]	0.74	0.85	—

VPP5000/64 (次数 2000)

GECP はさすがに計算機にあわせて最適化されたライブラリには速度の面では劣るものの，特に VPP5000/64 のソースレベルでは NUMPAC や LAPACK と遜色ない結果となっています．

6 おわりに

GECP の有効な利用方法としては，

- どの計算機でも動作する簡便なインターフェースを持つソルバーとして用いる
- 4 倍精度型や数値的な安定性を重視したい場合に利用する
- 他の連立 1 次方程式ソルバーとの性能比較に使用する

などが考えられます．

もちろん，プログラムのデバッグが完了した時点でその計算機に応じて最適化されたライブラリがあり，計算精度も問題ない場合には，積極的にそれらのサブルーチンと差し替えることも，性能向上の観点からは大切なことです．

参考文献

[1] 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).

[2] 伊理 正夫: 線形代数 II, 岩波講座 応用数学 [基礎 1], 岩波書店, 1994.

[3] 渡部 善隆: 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 ~ , 九州大学大型計算機センター広報, Vol.28, No.4 (1995), pp.291–349.
`http://www.cc.kyushu-u.ac.jp/RD/watanabe/RESERCH/MANUSCRIPT/KOHO/GEPP/intro.html`

[4] NUMPAC 利用手引書, 富士通株式会社, 1994．
`http://netnumpac.fuis.fukui-u.ac.jp/`

[5] BLAS, LAPACK, ScaLAPACK 使用手引書, 富士通株式会社, 1999. (PDF 形式; 33 ページ; 133KB)
`http://www.netlib.org/lapack/`

[6] 富士通 SSL II 使用手引書 (科学用サブルーチンライブラリ), 富士通株式会社, 99SP-4020, 1987.

[7] FUJITSU 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), 富士通株式会社, 99SP-4070, 1991．