

1 1. MGCL の用意する数値計算

1 1. 1 連立 1 次方程式

a_{ij} および b_i が既知、 x_j を未知とする次の式で表現される n 元連立一次方程式の解を求める。

$$(式11-1) \quad \sum_{j=0}^{j=n-1} a_{ij} x_j = b_i \quad \text{for } i=0, \dots, n-1$$

(1) 3 元連立一次方程式 ($n=3$: 未知数が 3 個の連立方程式) の解: `bz3sol_0`

```
void bz3sol_(const double * a, const double * b, double * x, int * iflag);
```

`bz3sol_0` は 3 個の未知数を含む 1 次方程式 (3 元の連立 1 次法廷式) の解を求める。連立方程式は

$$(式11-2) \quad \sum_{j=0}^{j=n-1} a_{ij} x_j = b_i \quad \text{for } i=0,1,2$$

で表現され、次のプログラムリストの配列 **a** には a_{ij} を次式のように、また配列 **b** には b_i を入力し、配列 **x** に解が返される。**Iflag** には解が求められたときは 1 が、**a** の行列式が 0 で解が求められなかったときは 2 が返される。

$$(式11-3) \quad a[] = \{a_{00}, a_{10}, a_{20}, a_{01}, a_{11}, a_{21}, a_{02}, a_{12}, a_{22}\}$$

(2) 全正 (totally positive) な対角帯 n 元連立 1 次方程式の解: `factorizeBandLU` と `solveBandLU`
 全正(Totally positive)で、ピボットなしでガウスの消去法が利用できる n 元連立 1 次方程式の解法に利用される (ピボットが必要な一般的な 1 次方程式には利用できない)。対象は右非対角帯半幅 (対角 a_{ii} $i=0, \dots, n-1$ の上または右の幅) が `nbandu` で、左非対角帯半幅 (対角 a_{ii} の下または左の幅) が `nbandl` な、対角帯行列となる連立 1 次方程式。特に B-Spline の解に利用される。`b1bfac` は行列 $[a_{ij}]$ の LU 分解を行い、`b1bslv` はその LU 分解を用いて解を求める。両プログラムでのメモリの利用法は、著名な数値計算プログラムである LINPACK の利用法に則っている。

```
//LU Factorization
```

```

int factorizeBandLU(
    MGBPointSeq& W, ///< contains interesting part of the matrix M.
    ///< W.length()=nlower+1+nupper=nbands(band width of M), and
    ///< W.sdim()=n(order of the matrix M).
    ///< The diagonals(or bands) of M(i+j, j) are stored in W as W(i+nupper, j)
    ///< for i=-nupper,..., nlower, and j=0,..., n-1.
    ///< Explicitly, M has nlower bands below the diagonal and nupper bands above the
    ///< diagonal. Thus the band width nbands=nlower+1+nupper (=W.length()).
    ///< For exapmle, the interesting entries of M of order 9, whose nlower=1 and nupper=2
    ///< would appear in the 4 subscripts of W(i, j) as follows:
    ///<
    ///<      j=          0  1  2  3  4  5  6  7  8:
    ///<      i=0:         x  x 02 13 24 35 46 57 68,
    ///<      i=1:         x 01 12 23 34 45 56 67 78,
    ///<      i=2:         00 11 22 33 44 55 66 77 88,
    ///<      i=3:         10 21 32 43 54 65 76 87  x.
    ///<
    ///< All other entries of W not identified in this way with an entry of M
    ///< are never referenced.
    int nlower ///

```

nbands=W.length()=行列 M のバンド幅、と定義する。Nlower はバンド行列 M の対角より下のバンド幅、nupper を対角より上のバンド幅とすると、nupper=nbands - nlower - 1 である。

W.sdim()=n として、W は nbands*n の行列として考え、行列 M の要素を $M = [a_{i,j}]$ とすると、

$a_{i+j,j}$ の値は W(i+nupper, j) に格納される。行列Mの LU 分解の結果が同じく W に返される。こ

の出力結果を solveBandLU に入力して結果 x_j を求める。

(例) n=9 で、nlower =1, nupper =2 の場合、行列 $[a_{ij}]$ の対角幅の長さ=nbands=1+2+1=4 で、

W(i,j)には次のように格納される。

$$(式11-4) \quad W(i,j) = \begin{bmatrix} x & x & a_{02} & a_{13} & a_{24} & a_{35} & a_{46} & a_{57} & a_{68} \\ x & a_{01} & a_{12} & a_{23} & a_{34} & a_{45} & a_{56} & a_{67} & a_{78} \\ a_{00} & a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & a_{66} & a_{77} & a_{88} \\ a_{10} & a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & a_{76} & a_{87} & x \end{bmatrix}$$

ここで、x はダミーで、何を入力してもよく、参照されることはない。

```

/// solveBandLU returns the solution of the linear system M*X = A.
/// solveBandLU is the companion routine of factorizeBandLU, and
/// factorizeBandLU executes the LU-Factorization for M in W.
///
/// NOTE: This program is translated to C++ from BANSLV of the book
/// "A Practical Guide to Splines" by Carl de Boor, Springer-Verlag.
void solveBandLU(
    const MGBPointSeq& W, ///< factorized LU matrix is input
        ///< which is obtained by factorizeBandLU.
        ///< Left-bottom triangle including the diagonal contain L,
        ///< right-upper triangle excluding the diagonal contain U.
        ///< Refer to factorizeBandLU.
    int nlower, ///< number of bands of the matrix M below the main diagonal.
    const MGBPointSeq& A, ///< right hand side vector. A.length() is W.sdim()
        ///< =n(order of the matrix M).
    MGBPointSeq& X ///< solved X will be output. X.length() will be A.length(),
        ///< and X.sdim() will be A.sdim().
);

```

A.length()=W.sdim()=n、A.sdim()=nsd とすると、A は $n \times nsd$ の行列と考えることができ、 $M * X = A$ の方程式のAを入力し、解Xが返される。

(3) 実対称帯行列を係数とする n 元連立 1 次方程式の解 : factorizeCholeLU と solveCholeLU

実対称帯行列 (式1) で、 $a_{ij} = a_{ji}$ if $i \neq j$ 、かつ $a_{ij} = 0$ for $i-j > nbands$ であるような行列 $[a_{ij}]$

の n 元連立 1 次方程式をコレスキー解法により求める。factorizeCholeLU はコレスキー分解

($[a_{ij}] = LDL^T$) を行い、solveCholeLU は factorizeCholeLU の結果を用いて解 x_j を求める。

```

void factorizeCholeLU(
    MGBPointSeq& W ///< contains nbands diagonals in W(.,j), with the main diagonal
        ///< in W(.,0). Precisely, W(i,j) contains C(i+j,j), i=0,...,nbands-1, j=0,...,n-1.
        ///< W.length()=nbands, and W.sdim()=n is the order of the matrix C.
        ///< For example, the entries of a 7 diagonal symmetric matrix C of order 9
        ///< (nbands=4) would be stored in W(i,j) as follows:
        ///<
        ///<           j=           0 1 2 3 4 5 6 7 8:
        ///<           i=0:         00 11 22 33 44 55 66 77 88,
        ///<           i=1:         10 21 32 43 54 65 76 87  x,
        ///<           i=2:         20 31 42 53 64 75 86  x  x,
        ///<           i=3:         30 41 52 63 74 85  x  x  x.
        ///<
        ///< All other entries of W not identified in this way with an entry of C
        ///< are never referenced.

```

```

///< On return, W contains the Cholesky factorization C= L*D*L-transpose
///< with W(0, j) containing 1/D(j, j) for j=0,..., n-1.
///< And W(i, j) containing L(i+j, j) for i=1,..., nbands-1, and j=0, ..., n-i-1.
);

```

W.length()=nbands は対角部を含めた帯の半幅。W.sdim()=n とすると、W は nbands*n の行列と考えることができ、行列 M の要素を $M = [a_{i,j}]$ とすると、 $a_{i+j,j}$ の値は W(i, j) に格納して、factorizeCholeLU を呼び出す (i=0,...,nbands-1, j=0,...,n-1)。Wにはコレスキー分解 LDL^T が返される。W (0,j) for j=0,...,n-1 は $1/D_{j,j}$ で、W (i , j) は $L_{i+j,j}$ for i=0,...,nbands-1。

(例) n=9 で、nbands=4 の場合、w[] は次のように格納される。

$$(式11-5) \quad w[] = \begin{bmatrix} a_{00} & a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & a_{66} & a_{77} & a_{88} \\ a_{10} & a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & a_{76} & a_{87} & x \\ a_{20} & a_{31} & a_{42} & a_{53} & a_{64} & a_{75} & a_{86} & x & x \\ a_{30} & a_{41} & a_{52} & a_{63} & a_{74} & a_{85} & x & x & x \end{bmatrix}$$

ここで、x はダミーで、何を入力してもよく、参照されることはない。

```

///< Solves the linear system C*X = A , provided W contains the Cholesky
///< factorization.
///< The Cholesky factorization is obtained by factorizeCholeLU. See factorizeCholeLU.
///<
///< NOTE: This program is translated to C++ from BCHSLV of the book
///< "A Practical Guide to Splines" by Carl de Boor, Springer-Verlag.
void solveCholeLU(
    const MGBPointSeq& W, ///< contains the Cholesky factorization
                        ///< for C, can be obtained by factorizeCholeLU.
                        ///< Refer to factorizeCholeLU.
    const MGBPointSeq& A, ///< right hand side vector. A.length() is W.sdim()=n(order of
the matrix C).
    MGBPointSeq& X      ///< solved X will be output. X.length() will be A.length(),
                        ///< and X.sdim() will be A.sdim().
);

```

A.length()=W.sdim()=n、A.sdim()=nsd とすると、A は n*nsd の行列と考えることができ、 $M * X = A$ の方程式の A を入力し、解 X が返される。

- (4) ピボットイングを必要とする一般的な n 元連立 1 次方程式の解 : `factorizeLU` と `solveLU`
 ピボットイング (すなわち、最適な解が得られるように行の入れ替えを行う) 処理を行う一般的な n 元連立一次方程式の解を求める。

```
//LU factorization using pivot.
//Factorize A to LU in the linear equation A*X=B.
//Using output A, the solution of A*X=B is obtained by solveLU.
//Here, L is n by n lower triangular matrix and U is n by n upper
//triangular matrix. U's diagonal is (1).
void factorizeLU(
    MGMatrix& A, //left-hand side matrix of n*n is input, and
                //factorized LU matrix will be output where n=A.length().
                //left-bottom triangle including the diagonal will contain L,
                //right-upper triangle excluding the diagonal will contain U.
    int* id     //array of int id[n].
                //pivot id will be output in id[i] for i=0,...,n-1
);
```

A は $n \times n$ の行列で連立方程式 $A \cdot X = B$ の A を入力する。LU 分解の結果が出力される。Id は n 元の整数値の配列でピボット情報を格納するために利用される。`factorizeLU` の出力を `solveLU` に入力する。

```
//solve the linear equation A*X=B to get X inputting factorize A and B.
//A and id are obtained by factorizeLU.
void solveLU(
    const MGMatrix& A,
                //factorized LU matrix is input which is obtained by factorizeLU.
                //left-bottom triangle including the diagonal contain L,
                //right-upper triangle excluding the diagonal contain U.
    const int* id, //array of int id[n] which contain pivot id in id[i] for i=0,...,n-1.
                //this is the output of factorizeLU.
    const MGBPointSeq& B, //right hand side vector. A.length() is W.sdim();
    MGBPointSeq& X //solved X will be output. X.length() will be A.length(),
                //and X.sdim() will be A.sdim()
);
```

`factorizeLU` により LU 分解された行列 A とピボット情報 `id[n]` と連立方程式の右辺 B を入力し、解 X を求める。

1 1. 2 数値積分

- (1) `mgDefint`

関数 $f(x)$ の 1 次元有限区間の積分の値 $\int_a^b f(x) dx$ を二重指数関数型数値積分公式(double

exponential formula)により求める。

mgDefint は関数テンプレートとして、数値積分機能を提供する。mgDefint は積分区間の端点 (x=a, b) において関数が不定 (特異点) の際にも対応できるよう作成されている。

(注) 本テンプレートは丸善出版、渡部 力、名取 亮、小国 力 監修、「Fortran77 による数値計算ソフトウェア」に収録されている Fortran プログラム DEFINT を C++の関数テンプレート化したものである。

```
// Integrate FUNC (L=0) or F (L=1, see below) over finite interval (a,b) by
// the DE formula (double exponential formula).
// Function's return value: integral of FUNC (L=0) or F (L=1)
template<class func>
double mgDefint(
    func& f,      //Function object for integrand. f(x) returns
                 //the target integration function's value.
    double a,    //lower bound of integration
    double b,    //upper bound of integration
    double eps, //absolute error tolerance
    int l=0     //0 or 1
                //If l=0 then integrate f(x) over (a,b)
                //If l=1 then integrate F(x) over (a,b)
                //In case of l=1 f must be defined as follows:
                // IF (-c .LT. y .LT. 0) THEN f(y) = F(b - y)
                // IF ( 0 .LT. y .LE. c) THEN f(y) = F(a - y)
                // where c = (b - a) / 2.
);
```

関数 f を関数オブジェクト、すなわち f(x)が double を返すようなクラス func のインスタンス f

を、 $\int_a^b f(x)dx$ の関数 f(x)として入力する。mgDefint はこの関数の区間[a, b]の積分の結果が誤

差 eps に収まるように数値積分する。L は通常 0 と入力する (default 値) が、端点 x=a, または b において不連続であるとき l=1 とする。L=1 の場合関数 f (x)は、元の関数 f(x)を次のようなパラメータ変換を施した関数 F(t)を入力する：

$$x=a-t \text{ for } a \leq t < (a+b)/2,$$

$$x=b-t \text{ for } (a+b)/2 \leq t < b$$

(2) mgGausp

関数 f(x)の 1次元有限区間の積分の値 $\int_a^b f(x)dx$ をガウスの公式により求める。

(注) 本テンプレートは丸善(株)出版、渡部 力、名取 亮、小国 力 監修、「Fortran77 による数値計算ソフトウェア」に収録されている Fortran プログラム DGAUSP を C++の関数テンプレート化したものである。

```
// Legendre-Gauss quadratuer formula over (a,b)
// ---- input parameters ----
// f = class to evaluate the integrand
// a = lower bound of integration
// b = upper bound of integration
// n = number of points of the formula, must not larger than 16
// ---- Function's return value is result of integration
template<class func> double mgGausp(
    func& f, //Function object to evaluate the function f(t).
            //f(t) must return double value.
    double a, //lower bound of integration
    double b, //upper bound of integration
    int n=10//number of points of the formula, must not be larger than 16
);
```

1 1. 3 一般方程式の解 mgNlbit

方程式 $f(x) = 0$ の解を数値計算により求める。

```
//Compute the solution f(x)=0.
// SOLUTION OF A NON-LINEAR EQUATION BY BISECTION METHOD AND REGULA
// FALSI METHOD.
//Function's return value will be the solution x obtained.
template<class func>
double mgNlbit(
    func& f, //Function object to evaluate the fucntion.
            //must return double value.
    double xl, //LEFT HAND SIDE VALUE OF THE SOLUTION
    double xr, //RIGHT HAND SIDE VALUE OF THE SOLUTION
    double eps, //tolerance allowed for the convergence in world coordnate
    int itr, //MAXIMUM NUMBER OF REPITITION
    int& ier //Error code. =0:solution successfully obtained
            // =1:mgNlbit did not converg, and solution not obtained.
);
```

mgNlbit は関数テンプレートとして、 $f(x) = 0$ の解を求める。

(注) 本テンプレートは丸善(株)出版、渡部 力、名取 亮、小国 力 監修、「Fortran77 による数値計算ソフトウェア」に収録されている Fortran プログラム NLBIT を C++の関数テンプレート化したものである。

$f(xl)*f(xr)<0$.であるような xl と xr を与えて、 $f(x)=0$ の解を、二分法 (Bisection) をはさみうち

法の併用により求める。必ず収束するが、ニュートン・ラフソン法などと比較して、収束が遅い。
Eps は収束判定のための絶対誤差、**itr** は最大繰り返し回数であり、この回数繰り返して入力
の誤差範囲に収束しなかったときは **ier=1** で復帰する。

(手法) **mgNlbit** は二分法(Bisection)とはさみうち法を併用して解を求める。この繰り返しを最
大 **itr** 回だけ繰り返す。

- (1) 二分法 : **xl** と **xr** との midpoint $x=(xl+xr)/2$ の位置と解が **[xl, x]**か**[x,xr]**のいずれの
スパンにあるかを判断してスパンを狭める。
- (2) はさみうち法 : セカント法と同じ次の公式により、前の二つの近似 **xl** と **xr** を用いて
次の近似 **x** を求め、二分法と同じ判断により、**xl** または **xr** を入れ替え次の繰り返
しに移る。

$$(式11-6) \quad x = xr - \frac{xr - xl}{f(xr) - f(xl)} f(xr)$$