情報数理工学・コンピュータサイエンス実験第二
大規模連立一次方程式に対する共役勾配法

担当：緒方秀教　(e-mail)ogata@im.uec.ac.jp

2016 年 11 月 2 日（水）

1　はじめに

科学技術計算の問題の多くは，連立 1 次方程式を解く問題に帰着されることが多い．そしてとくに大型疎行列問題—係数行列の次元が大きく非零成分が少ないような連立 1 次方程式—が重要であり，数値解析の分野で精力的に研究され，さまざまな方法が提案されている．共役勾配法はそれらの方法のひとつである．

本実験では，共役勾配法の理論を理解し，実際にプログラムをつくって実行することを目的とする．具体的には，次の 4 項を目的とする．

1. 共役勾配法の理論・特性を理解し，簡単な計算プログラムが作成・実行できる．
2. 前処理による共役勾配法の効率化について理解し，プログラムを作成・実行できる．
3. 共役勾配法を実際の科学技術計算に応用できる．具体的には，Laplace 方程式の境界値問題の差分法の解法を扱う．
4. 大型疎行列のデータ構造の扱い方を会得する（発展課題）．

なお，共役勾配法および関連解法についての詳細は，[4] を参照すること．

2　理論

共役勾配法 (conjugate gradient method, **CG 法**) は，連立 1 次方程式

$$A\boldsymbol{x} = \boldsymbol{b}, \quad A \text{ は } n \times n \text{ 正値対称行列} \tag{1}$$

の解 $\boldsymbol{x} = \boldsymbol{x}^*$ を 2 次関数 [1]

$$f(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{x}, A\boldsymbol{x}) - (\boldsymbol{b}, \boldsymbol{x}) \tag{2}$$

の最適化問題 [2]

$$\text{Find } \boldsymbol{x}^* \in \mathbb{R}^n \text{ s.t. } f(\boldsymbol{x}^*) = \min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x}) \tag{3}$$

として求める方法である．

[1] $(\boldsymbol{u}, \boldsymbol{v})$ は 2 つのベクトル $\boldsymbol{u}, \boldsymbol{v}$ の内積である，すなわち，$\boldsymbol{u} = (u_1, \ldots, u_n)^t$, $\boldsymbol{v} = (v_1, \ldots, v_n)^t$ とするとき $(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^t\boldsymbol{v} = u_1v_1 + \cdots + u_nv_n$.

[2] $\mathbb{R}$ は実数全体の集合，$\mathbb{R}^n$ は $\mathbb{R}$ の n 個の直積，すなわち，n 次元実ベクトル全体の集合である．

ここでは，最適化問題 (3) を**反復法** (iterative method) で解くことにする．反復法とは，ある近似解が得られているときそれをもとによりよい近似解を求め，その操作を反復することにより精度のよい近似解を得る方法である．すなわち，あるステップ（k 段目とする）において近似解 $\boldsymbol{x}_k$ が得られているとすると，探索方向ベクトル (search direction vector) $\boldsymbol{p}_k$ を定めて 1 次元探索問題

$$\text{Find } \alpha_k \in \mathbb{R} \text{ s.t. } f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k) = \min_{\alpha \in \mathbb{R}} f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k) \tag{4}$$

を解き，$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$ を次のステップの近似解とする．この操作は次のアルゴリズムにまとめられる．

算法 1 (反復法（ひな型）)

give an initial guess $\boldsymbol{x}_0$;
$\boldsymbol{r}_0 = \boldsymbol{p}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$;
give a residual tolerance $\epsilon(> 0)$;
for $k = 0, 1, 2, \ldots$
- find $\alpha_k \in \mathbb{R}$ s.t. $f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k) = \min_{\alpha \in \mathbb{R}} f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k)$
- $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$;
- $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k$;
- if $\|\boldsymbol{r}_{k+1}\| < \epsilon\|\boldsymbol{b}\|$, stop the iterations.
- give the search direction vector $\boldsymbol{p}_{k+1}$;

ここで，$\boldsymbol{r}_k = \boldsymbol{b} - A\boldsymbol{x}_k$ は k 段目の近似解 $\boldsymbol{x}_k$ に対する**残差ベクトル** (residual vector) と呼ばれる (誤差ベクトル $\boldsymbol{x}_k - \boldsymbol{x}^*$ と区別すること)．残差ベクトルのノルム $\|\boldsymbol{r}_k\|$ が，ϵ を十分小さい正数として，$\|\boldsymbol{r}_k\| \leqq \epsilon\|\boldsymbol{b}\|$ を満たしたら反復を終了し，$\boldsymbol{x}_k$ を近似解として返している．

$f(\boldsymbol{x})$ が 2 次関数 (2) の場合，α_k は次のように陽に求まることに注意．

$$\alpha_k = \frac{(\boldsymbol{p}_k, \boldsymbol{r}_k)}{(\boldsymbol{p}_k, A\boldsymbol{p}_k)}. \tag{5}$$

さて，ここで各段の探索方向ベクトル $\boldsymbol{p}_k$ をどうやって定めるかが問題になり，その定め方により共役勾配法を含めいろいろな方法が生成される．ここで，本題の共役勾配法の前に，2 つの反復法—最急降下法と共役方向法— について説明する．

課題 1　*1.* 連立 *1* 次方程式 *(1)* の解が最適化問題 *(3)* の解であることを示せ．

2. α_k の表式 *(5)* を導出せよ．

2.1　最急降下法

探索方向ベクトルのとり方として素朴に思いつくのは，$f(\boldsymbol{x})$ の $\boldsymbol{x}_k$ の最急降下方向

$$\boldsymbol{r}_k \equiv -\nabla f(\boldsymbol{x}_k) = \boldsymbol{b} - A\boldsymbol{x}_k$$

を用いることである．ここで，ベクトル $\boldsymbol{r}_k$ を**残差ベクトル**と呼ぶ．この反復法を**最急降下法** (steepest descent method) と呼ぶ．アルゴリズムを書き下すと次のようになる．

算法 2 (最急降下法)

give an initial guess $\boldsymbol{x}_0$;
$\boldsymbol{r}_0 = \boldsymbol{p}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$;
give a residual tolerance $\epsilon(> 0)$;
for $k = 0, 1, 2, \ldots$
$$\left[\begin{array}{l} \alpha_k = \dfrac{(\boldsymbol{p}_k, \boldsymbol{r}_k)}{(\boldsymbol{p}_k, A\boldsymbol{p}_k)}; \\ \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k; \\ \boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k; \\ \text{if } \|\boldsymbol{r}_{k+1}\| < \epsilon\|\boldsymbol{b}\|, \text{ stop the iterations}; \\ \boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1}; \end{array}\right.$$

ただし，この方法は収束が遅くなることがあるという欠点がある．詳しく言うと，探索方向ベクトル $\boldsymbol{p}_k$ で同じ方向を持つものが繰り返し現れ，近似解を求める空間 $\boldsymbol{x}_0 + \mathrm{span}\{\, \boldsymbol{p}_0, \ldots, \boldsymbol{p}_k \,\}$ がいつまでたっても拡がらず，真の解 $\boldsymbol{x}^*$ への収束が遅くなるのである．

この欠点を克服し収束性を格段に改善したのが，次に説明する共役方向法，そして，共役勾配法である．

2.2 共役方向法

共役方向法 (conjugate direction method) は，反復法（算法 1）の**探索方向ベクトル $\boldsymbol{p}_k$ として**，

***A*-直交性**

$$(\boldsymbol{p}_j, A\boldsymbol{p}_k) = 0 \quad (\, j \neq k \,) \tag{6}$$

を満たすものを用いる方法の総称であり，本実験で扱う共役勾配法もこれに含まれる．

課題 2 探索方向ベクトル系 $\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_k$ は *A*-直交性 *(6)* を満たすとき *1* 次独立であることを示せ．

課題 2 により，探索方向ベクトル系 $\boldsymbol{p}_k$, $k = 0, 1, \ldots$ は 1 次独立であるから，最急降下法のように近似解を探す部分空間が拡がらないという欠点は解消される．

さらに著しい性質として，**共役方向法は直線探索という局所的な最適化を繰り返すことにより，大域的な最適化を実現している**．すなわち，

$$f(\boldsymbol{x}_{k+1}) = \min\{\, f(\boldsymbol{x}) \mid \boldsymbol{x} \in \boldsymbol{x}_0 + \mathrm{span}\{\, \boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_k \,\} \,\}. \tag{7}$$

というふうに，k 段目の直線探索を行った段階で，これまで近似解を探してきた部分空間 $\boldsymbol{x}_0 + \mathrm{span}\{\, \boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_k \,\}$ 全体での最適化を達成している．これは，$\boldsymbol{p}_k$ が *A*-直交性 (6) を満たす場合，k 段目の近似解 $\boldsymbol{x}_k = \sum_{j=0}^{k-1} \alpha_j \boldsymbol{p}_j + \boldsymbol{x}_0$ に対する 2 次関数 $f(\boldsymbol{x})$ の値は

$$f(\boldsymbol{x}_k) = \sum_{j=0}^{k-1} \psi(\alpha_j \boldsymbol{p}_j) + f(\boldsymbol{x}_0), \quad \psi(\boldsymbol{\xi}) = \frac{1}{2}(\boldsymbol{\xi}, A\boldsymbol{\xi}) - (\boldsymbol{r}_0, \boldsymbol{\xi}) \tag{8}$$

と，α_j について**変数分離**されることが分かる．とくに $k=n$ の場合，$\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_{n-1}$ が一次独立，すなわち，$\mathbb{R}^n$ の基底をなすことから，$\boldsymbol{x}_0 + \mathrm{span}\{\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_{n-1}\} = \mathbb{R}^n$ により，$f(\boldsymbol{x}_n) = \min\limits_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x})$，すなわち，$\boldsymbol{x}_n = \boldsymbol{x}^*$ を得る．したがって，**有限解の反復で（理論的には）厳密解 $\boldsymbol{x}^*$ を得る**ことが分かる．

ここで，次の共役勾配法の説明の準備として，次の命題を証明する．

命題 1

$$(\boldsymbol{r}_{k+1}, \boldsymbol{p}_j) = 0, \quad j = 0, 1, 2, \ldots, k.$$

（証明）まず $j=k$ の場合，$f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k) = \min_{\alpha\in\mathbb{R}} f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k)$ より，

$$0 = \frac{\mathrm{d}}{\mathrm{d}\alpha} f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k)\bigg|_{\alpha=\alpha_k} = (\boldsymbol{p}_k, \nabla f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)) = -(\boldsymbol{p}_k, \boldsymbol{r}_{k+1}).$$

$j \leqq k-1$ の場合は

$$(\boldsymbol{r}_{k+1}, \boldsymbol{p}_j) = \left(\boldsymbol{r}_{j+1} - \sum_{i=j+1}^{k} \alpha_i A\boldsymbol{p}_i, \boldsymbol{p}_j\right) = \underbrace{(\boldsymbol{r}_{j+1}, \boldsymbol{p}_j)}_{0} - \sum_{i=j+1}^{k} \alpha_i \underbrace{(\boldsymbol{p}_i, A\boldsymbol{p}_j)}_{0} = 0.$$

■

2.3 共役勾配法

共役方向法において，A-直交性を満たす探索方向ベクトル $\boldsymbol{p}_k$ を最急降下方向ベクトル $-\nabla f(\boldsymbol{x}_k) = \boldsymbol{r}_k$ の Gram-Schmidt 直交化により生成するのが共役勾配法である [3]．すなわち，

$$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} - \sum_{j=0}^{k} \beta_j^{(k)} \boldsymbol{p}_j, \quad \beta_j^{(k)} = \frac{(\boldsymbol{r}_{k+1}, A\boldsymbol{p}_j)}{(\boldsymbol{p}_j, A\boldsymbol{p}_j)}. \tag{9}$$

ここで重要なことは，(9) 第 1 式右辺において **$\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_{k-1}$ の係数がすべて 0 になる** ことである．

命題 2 *(9)* において，

$$\beta_j^{(k)} = \begin{cases} 0 & j \leqq k-1 \\ -\|\boldsymbol{r}_{k+1}\|^2 / \|\boldsymbol{r}_k\|^2 & j = k. \end{cases}$$

この命題のおかげで，探索方向ベクトル $\boldsymbol{p}_k$ を更新する際，過去のステップのデータを大量に保存せずにすみ，メモリの消費の観点から都合がよいのである．

（証明）$j = 0, 1, \ldots, k-1$ に対し

$$(\boldsymbol{r}_{k+1}, A\boldsymbol{p}_j) = \left(\boldsymbol{r}_{k+1}, A\left(\frac{\boldsymbol{x}_{j+1} - \boldsymbol{x}_j}{\alpha_j}\right)\right) = -\frac{1}{\alpha_k}(\boldsymbol{r}_{k+1}, \boldsymbol{r}_{j+1} - \boldsymbol{r}_j) = 0,$$

最後の等号で $\boldsymbol{r}_{j+1} - \boldsymbol{r}_j \in \mathrm{span}\{\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots \boldsymbol{p}_{j+1}\}$ に注意して命題 1 を用いた．よって，$\beta_j^{(k)} = 0$ $(j \leqq k-1)$．$j = k$ に対しては，

$$(\boldsymbol{r}_{k+1}, A\boldsymbol{p}_k) = -\frac{1}{\alpha_k}(\boldsymbol{r}_{k+1}, \boldsymbol{r}_{k+1} - \boldsymbol{r}_k) = -\frac{1}{\alpha_k}\|\boldsymbol{r}_{k+1}\|^2,$$

2 番目の等号で $\boldsymbol{r}_k = \boldsymbol{p}_k + \beta_{k-1}^{(k-1)} \boldsymbol{r}_{k-1}$ に注意して命題 1 を用いた．ここで，α_k の式 (5) を代入し，$(\boldsymbol{p}_k, \boldsymbol{r}_k) = (\boldsymbol{r}_k, \boldsymbol{r}_k) - \beta_{k-1}^{(k-1)}(\boldsymbol{p}_{k-1}k, \boldsymbol{r}_k) = \|\boldsymbol{r}_k\|^2$ に注意すると，$\beta_k^{(k)} = -\|\boldsymbol{r}_{k+1}\|^2 / \|\boldsymbol{r}_k\|^2$ を得る． ■

[3] この辺の記述は文献 [4] による．

以上より次の共役勾配法の算法を得る．

算法 3 (共役勾配法（原型）)

give an initial guess $\boldsymbol{x}_0$;　$\boldsymbol{r}_0 = \boldsymbol{p}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$;
for $k = 0, 1, 2, \ldots$

$$\alpha_k = \frac{\|\boldsymbol{r}_k\|^2}{(\boldsymbol{p}_k, A\boldsymbol{p}_k)};$$
$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$;　$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k$;
if $\|\boldsymbol{r}_{k+1}\| \leqq \epsilon\|\boldsymbol{b}\|$, stop the iterations;

$$\beta_k = \frac{\|\boldsymbol{r}_{k+1}\|^2}{\|\boldsymbol{r}_k\|^2};$$
$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k$;

算法 2.3 から分かるとおり，共役勾配法は行列 A とベクトルの積を繰り返すことにより連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の解 $\boldsymbol{x}^* = A^{-1}\boldsymbol{b}$ を得る方法である．係数行列 A が疎行列（非零要素が少ない行列）である場合，行列・ベクトル積の演算量が少ないので，連立 1 次方程式の数値解法として共役勾配法が適している．後述の応用例に示すとおり，科学技術における数学の問題は大規模疎行列を係数行列とする連立 1 次方程式に帰着されることが多いので,（後述の「前処理」を併用することにより）共役勾配法は科学技術計算に広く応用されている．

2.4　数値実験

課題 3 行列

$$A\boldsymbol{x} = \boldsymbol{b}, \quad A = \begin{bmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & 1 & 4 & 1 & \\ & & \ddots & \ddots & \ddots \\ & & & 1 & 4 \end{bmatrix}$$

を係数行列にもつ連立 *1* 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ に対し最急降下法および共役勾配法を適用し，残差の大きさ $\|\boldsymbol{r}_k\|$ のステップ数 k に対する変化を比較せよ．そして，残差の変化をグラフに描け．最急降下法のプログラムは，緒方研究室ホームページ `http://www.uec-ogata-lab.jp/` にアクセスして得られるプログラム `sd.c` を用いよ（上記 *Web* サイトにおいて「教育ページ」をクリック，その後現れるページの「*2016* 年度の授業」の項目に「実験で使う最急降下法のプログラム *(*`sd.c`*)*」と記してあるので，`sd.c` をクリックすれば *C* プログラムがダウンドーロできる）．共役勾配法のプログラムは `sd.c` を書き直して作成せよ．

著者による数値例（次元 $n = 20$，右辺ベクトル $\boldsymbol{b} = A(1, 2, \ldots, n)^t$）を図 1 に示す．図より，共役勾配法は最急勾配法に比べて速く収束していることが分かる．共役勾配法に対しては，ステップ数 $k = n(= 20)$ で残差 $\|\boldsymbol{r}_k\| \approx 0$ になることに注意せよ．

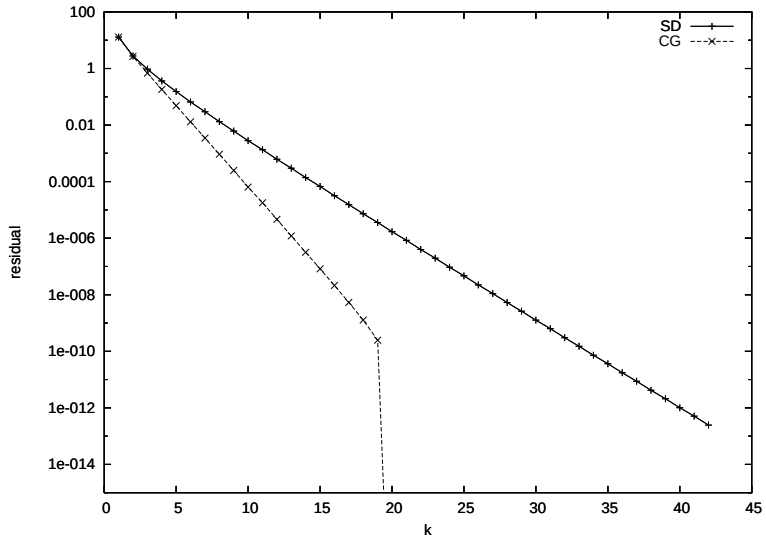

図 1: The residuals of the steepest descent method (SD) and the conjugate gradient method (CG) as functions of the number of iterations k.

2.5 収束性

最急勾配法および共役勾配法の収束性は次の定理で示される.

定理 1 係数行列 A の最大固有値を $\lambda_{\max}$，最小固有値を $\lambda_{\min}$ とし，$\kappa = \lambda_{\max}/\lambda_{\min}$ とおく [4]（A は正値対称行列であるから $\lambda_{\max} \geq \lambda_{\min} > 0$，したがって $\kappa \geq 1$ となることに注意）．このとき，最急降下法または共役勾配法の k 段目の近似解を $\boldsymbol{x}_k$ とおくと，次の不等式が成り立つ.

$$\text{最急勾配法} \quad f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*) \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^{2k} \{f(\boldsymbol{x}_0) - f(\boldsymbol{x}^*)\}, \tag{10}$$

$$\text{共役勾配法} \quad f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*) \leq \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^{2k} \cdot 4\{f(\boldsymbol{x}_0) - f(\boldsymbol{x}^*)\}. \tag{11}$$

□

$\kappa \geq 1$ より

$$\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \leq \frac{\kappa - 1}{\kappa + 1}$$

であることに注意すれば，(10), (11) より $f(\boldsymbol{x}_k) \to f(\boldsymbol{x}^*)$ ($k \to \infty$) の収束は共役勾配法は最急勾配法より速いことが分かる．したがって，定理 1 より，共役勾配法は最急勾配法より速く収束していることが理論的にも示される.

[4] κ は行列 A の条件数とよばれる数である.

3 ICCG 法

3.1 共役勾配法，その後…

1952 年 Hestines と Stiefel[2] により共役勾配法が発表された当初は，大規模線形計算の画期的な方法として大きな話題となった．ところが，この方法は**丸め誤差に弱い**という欠点があり，そのため数値計算の世界から長いこと忘れ去られていた．

共役勾配法が再び日の目を見たのは 1980 年代である．前処理（後述）の技術と併用することにより，共役勾配法は大規模疎行列の連立 1 次方程式の解法として有効であることが見出され，大規模線形計算の世界で広く用いられるようになった．その後，非対称行列への拡張である BiCG 法などが提案されるなど，共役勾配法に関連した線形計算の分野は現在も発展し続けている．

3.2 不完全 Cholesky 分解による前処理

連立 1 次方程式の数値解法における**前処理** (preconditioning) とは，もとの方程式を数値的に解きやすい同値な方程式に書き直す手続きのことをいう．具体的には，ある正則行列 C を用いてもとの方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を

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

すなわち，

$$\widetilde{A}\widetilde{\boldsymbol{x}} = \widetilde{\boldsymbol{b}}, \quad \left(\widetilde{A} = C^{-1}AC^{-T},\ \widetilde{\boldsymbol{x}} = C^T\boldsymbol{x},\ \widetilde{\boldsymbol{b}} = C^{-1}\boldsymbol{b}\right) \tag{13}$$

と変形し，$\widetilde{x}$ についての連立 1 次方程式 (13) に共役勾配法を適用する．定理 1 の (11) より κ が 1 に近いほど共役勾配法の収束は速くなる．すなわち，$\widetilde{A} = C^{-1}AC^{-T}$ **が単位行列に近いほど共役勾配法の収束は速くなる**．したがって，行列 A が近似的に $A \approx CC^T$ に分解されるならば，この行列 C を上の前処理に用いればよい．

行列の近似的分解 $A \approx CC^T$ としてここでは，下記に述べる**不完全 Cholesky 分解**を用いる．

Cholesky 分解　はじめに **Cholesky 分解** (Cholesky decomposition) について述べる．係数行列 A が対称行列である場合，A の LU 分解は

$$A = LDL^t \tag{14}$$

と書ける．ここで，L は下三角行列，D は対角行列である：

$$L = \begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix}, \quad D = \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{bmatrix}. \tag{15}$$

式 (14) を行列 A の **Cholesky 分解**という．L, D の定め方は一意的でなく任意性が残されているが，ここでは

$$l_{ii}d_i = 1, \quad i = 1, 2, \ldots, n \tag{16}$$

となるよう L, D を定めることにする．すると，等式 (14) を成分ごとに書き下した式より，L, D を求める次の算法を得る．

算法 4 (Cholesky 分解)

$$
\begin{array}{l}
\text{for } i = 1, 2, \ldots, n \\
\quad \left[\begin{array}{l}
\quad \text{for } j = 1, 2, \ldots, i \\
\qquad \left[l_{ij} = a_{ij} - \displaystyle\sum_{k=1}^{j-1} l_{ik} d_k l_{jk}; \quad (j = 1 \text{ の場合，右辺第 2 項の和} = 0 \text{ とする}) \right. \\
d_i = 1/l_{ii};
\end{array} \right.
\end{array}
$$

不完全 Cholesky 分解　大型対称疎行列 A に対する**不完全 Cholesky 分解** (incomplete Cholesky decomposition) とは，A が疎行列であるという特性を生かして，Cholesky 分解の算法 4 を不完全に行うことである．具体的には，

> A の (i,j) 成分が 0 ならば，L の (i,j) 成分も強制的に 0 におく

とする．不完全 Cholesky 分解の算法は，算法 5 のようになる [5]．

算法 5 (不完全 Cholesky 分解)

$$
\begin{array}{l}
\text{for } i = 1, 2, \ldots, n \\
\quad \left[\begin{array}{l}
\quad \text{for } j = 1, 2, \ldots, i \\
\qquad \left[l_{ij} = \begin{cases} \mathbf{0} & \text{if} \quad \mathbf{a_{ij} = 0} \\ a_{ij} - \displaystyle\sum_{k=1}^{j-1} l_{ik} d_k l_{jk} & \text{otherwise} \end{cases} \right. \\
d_i = 1/l_{ii};
\end{array} \right.
\end{array}
$$

ICCG 法　行列 A が

$$A \simeq LDL^t \tag{17}$$

と不完全 Cholesky 分解されているとする．行列 [6]

$$\widetilde{A} = D^{-1/2} L^{-1} A L^{-t} D^{-1/2}$$

は単位行列に近いと期待される．一方，**共役勾配法は，係数行列が単位行列に近いほど収束が速い**ことが知られている．そこで，もとの連立 1 次方程式 (1) の代わりに，方程式

$$
\begin{array}{c}
(D^{-1/2} L^{-1} A L^{-t} D^{-1/2})(D^{1/2} L^t \boldsymbol{x}) = D^{-1/2} L^{-1} \boldsymbol{b}, \\
\text{すなわち,} \quad \widetilde{A}\widetilde{\boldsymbol{x}} = \widetilde{\boldsymbol{b}}, \quad \widetilde{\boldsymbol{x}} = D^{1/2} L^t \boldsymbol{x}, \quad \widetilde{\boldsymbol{b}} = D^{-1/2} L^{-1} \boldsymbol{b}
\end{array} \tag{18}
$$

に共役勾配法を適用すれば，より速い収束が期待できる．方程式 (18) に共役勾配法を適用して，つぎの **ICCG 法**の算法を得る．

---

[5] 念のために一言．算法を一瞥すると，不完全 Cholesky 分解は「完全」Cholesky 分解より面倒くさいことをやっているように見える．しかし，後述するように，実際の計算では，$n \times n$ 大規模行列 A に対しては $n \times n$ ワード分の 2 次元配列を用意せず，A の非零成分のデータ分だけメモリを用意し，データ構造を設定する．だから，「完全」Cholesky 分解より不完全 Cholesky 分解のほうが安い手間でできるのである．

[6] $L^{-t} = (L^t)^{-1} = (L^{-1})^t$, $D^{1/2} = \begin{bmatrix} d_1^{1/2} & & \\ & \ddots & \\ & & d_n^{1/2} \end{bmatrix}$, $D^{-1/2} = (D^{1/2})^{-1}$. A は正値対称行列なので $d_1, \ldots, d_n > 0$ であることに注意.

算法 6 (ICCG 法) $A \simeq LDL^t$ であるとする.

give an initial guess $\boldsymbol{x}_0$;
$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$; $\quad \boldsymbol{p}_0 = (LDL^t)^{-1}\boldsymbol{r}_0$;
for $k = 0, 1, 2, \ldots$

$$\left[\begin{array}{l} \alpha_k = \dfrac{(\boldsymbol{r}_k, (LDL^t)^{-1}\boldsymbol{r}_k)}{(\boldsymbol{p}_k, A\boldsymbol{p}_k)}; \\ \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k; \quad \boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k; \\ \text{if } \|\boldsymbol{r}_{k+1}\| \leqq \epsilon \|\boldsymbol{b}\|, \text{ stop the iterations}; \\ \beta_k = \dfrac{(\boldsymbol{r}_{k+1}, (LDL^t)^{-1}\boldsymbol{r}_{k+1})}{(\boldsymbol{r}_k, (LDL^t)^{-1}\boldsymbol{r}_k)}; \\ \boldsymbol{p}_{k+1} = (LDL^t)^{-1}\boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k; \end{array}\right.$$

算法 6 において，行列 $(LDL^t)^{-1}$ の掛け算 $\boldsymbol{x} \mapsto \boldsymbol{y} = (LDL^t)^{-1}\boldsymbol{x}$ は実際には，$\boldsymbol{y}, \boldsymbol{z}$ に関する連立 1 次方程式

$$\begin{cases} LD\boldsymbol{z} = \boldsymbol{x} \\ L^t\boldsymbol{y} = \boldsymbol{z} \end{cases} \tag{19}$$

を解いて解 $\boldsymbol{y}$ を求めることによって行う．L は下三角行列，D は対角行列であるから，(19) 第 1 式は前進代入，第 2 式は後退代入により解が求められる．

ICCG 法アルゴリズムの導出 まず，方程式 (18) に CG 法を適用すると，次のアルゴリズムを得る（収束判定は省略）.

give $\widetilde{\boldsymbol{x}}_0$; $\quad \widetilde{\boldsymbol{r}}_0 = \widetilde{\boldsymbol{p}}_0 = \widetilde{\boldsymbol{b}} - \widetilde{A}\widetilde{\boldsymbol{x}}_0$;
for $k = 0, 1, 2, \ldots$

$$\left[\begin{array}{l} \widetilde{\alpha}_k = \dfrac{\|\widetilde{\boldsymbol{r}}_k\|^2}{(\widetilde{\boldsymbol{p}}_k, \widetilde{A}\widetilde{\boldsymbol{p}}_k)}; \\ \widetilde{\boldsymbol{x}}_{k+1} = \widetilde{\boldsymbol{x}}_k + \widetilde{\alpha}_k \widetilde{\boldsymbol{p}}_k; \quad \widetilde{\boldsymbol{r}}_{k+1} = \widetilde{\boldsymbol{r}}_k - \widetilde{\alpha}_k \widetilde{A}\widetilde{\boldsymbol{p}}_k; \\ \widetilde{\beta}_k = \dfrac{\|\widetilde{\boldsymbol{r}}_{k+r}\|}{\|\widetilde{\boldsymbol{r}}_k\|}; \\ \widetilde{\boldsymbol{p}}_{k+1} = \widetilde{\boldsymbol{r}}_{k+1} + \widetilde{\beta}_k \widetilde{\boldsymbol{p}}_k; \end{array}\right. \tag{20}$$

$\widetilde{\boldsymbol{x}}$ の定義より $\widetilde{\boldsymbol{x}}_k = D^{1/2}L^t\boldsymbol{x}_k$, そして,

$$\begin{aligned} \widetilde{\boldsymbol{r}}_k = \widetilde{\boldsymbol{b}} - \widetilde{A}\widetilde{\boldsymbol{x}}_k &= D^{-1/2}L^{-1}\boldsymbol{b} - (D^{-1/2}L^{-1}AL^{-t}D^{-1/2})D^{1/2}L^t\boldsymbol{x}_k \\ &= D^{-1/2}L^{-1}(\boldsymbol{b} - A\boldsymbol{x}_k) = D^{-1/2}L^{-1}\boldsymbol{r}_0 \end{aligned}$$

により $\widetilde{\boldsymbol{r}}_k = D^{-1/2}L^{-1}\boldsymbol{r}_k$ とおく．$\widetilde{\boldsymbol{p}}_k$ は $\widetilde{\boldsymbol{x}}_k = D^{1/2}L^t\boldsymbol{x}_k$ を更新するときの探索方向ベクトル，$\boldsymbol{p}_k$ は $\boldsymbol{x}_k$ を更新するときの探索方向ベクトルであることを考慮して，$\widetilde{\boldsymbol{p}}_k = D^{1/2}L^t\boldsymbol{p}_k$ とおく．すると，(20) において

$$\widetilde{\boldsymbol{r}}_0 = D^{-1/2}L^{-1}\boldsymbol{r}_0 \quad \text{より} \quad \boldsymbol{r}_0 = LD^{1/2}\widetilde{\boldsymbol{r}}_0,$$

$$
\begin{aligned}
\boldsymbol{p}_0 &= L^{-t}D^{-1/2}\widetilde{\boldsymbol{p}}_0 = L^{-t}D^{-1/2}\widetilde{\boldsymbol{r}}_0 = L^{-t}D^{-1/2}D^{-1/2}L^{-1}\boldsymbol{r}_0 \\
&= L^{-t}D^{-1}L^{-1}\boldsymbol{r}_0 = (LDL^t)^{-1}\boldsymbol{r}_0, \\
\widetilde{\alpha}_k &= \frac{(D^{-1/2}L^{-1}\boldsymbol{r}_k, D^{-1/2}L^{-1}\boldsymbol{r}_k)}{(D^{1/2}L^t\boldsymbol{p}_k, (D^{-1/2}L^{-1}AL^{-t}D^{-1/2})D^{1/2}L^t\boldsymbol{p}_k)} \\
&= \frac{(\boldsymbol{r}_k, L^{-t}D^{-1/2}D^{-1/2}L^{-1}\boldsymbol{r}_k)}{(\boldsymbol{p}_k, (LD^{1/2})(D^{-1/2}L^{-1})A\boldsymbol{p}_k)} \\
&= \frac{(\boldsymbol{r}_k, L^{-1}D^{-1}L^{-t}\boldsymbol{r}_k)}{(\boldsymbol{p}_k, A\boldsymbol{p}_k)} = \frac{(\boldsymbol{r}_k, (LDL^t)^{-1}\boldsymbol{r}_k)}{(\boldsymbol{p}_k, A\boldsymbol{p}_k)} =: \alpha_k, \\
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k; \\
\boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \alpha_k LD^{1/2}(D^{-1/2}L^{-1}AL^{-t}D^{-1/2})D^{1/2}L^t\boldsymbol{p}_k = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k, \\
\widetilde{\beta}_k &= \frac{(D^{-1/2}L^{-1}\boldsymbol{r}_{k+1}, D^{-1/2}L^{-1}\boldsymbol{r}_{k+1})}{(D^{-1/2}L^{-1}\boldsymbol{r}_k, D^{-1/2}L^{-1}\boldsymbol{r}_k)} = \frac{(\boldsymbol{r}_{k+1}, L^{-t}D^{-1/2}(D^{-1/2}L^{-1}\boldsymbol{r}_{k+1}))}{(\boldsymbol{r}_k, L^{-t}D^{-1/2}(D^{-1/2}L^{-1}\boldsymbol{r}_k))} \\
&= \frac{(\boldsymbol{r}_{k+1}, L^{-t}D^{-1}L^{-1}\boldsymbol{r}_{k+1})}{(\boldsymbol{r}_k, L^{-t}D^{-1}L^{-1}\boldsymbol{r}_k)} = \frac{(\boldsymbol{r}_{k+1}, (LDL^t)^{-1}\boldsymbol{r}_{k+1})}{(\boldsymbol{r}_k, (LDL^t)^{-1}\boldsymbol{r}_k)} =: \beta_k, \\
\boldsymbol{p}_{k+1} &= (L^{-t}D^{-1/2})(D^{-1/2}L^{-1})\boldsymbol{r}_{k+1} + \beta_k\boldsymbol{p}_k \\
&= (L^{-t}D^{-1}L^{-1})\boldsymbol{r}_{k+1} + \beta_k\boldsymbol{p}_k = (LDL^t)^{-1}\boldsymbol{r}_{k+1} + \beta_k\boldsymbol{p}_k
\end{aligned}
$$

となるから，ICCG 法のアルゴリズムを得る． ■

3.3 数値実験

課題 4 式 *(21)* の A を係数行列にもつ連立 *1* 次方程式に対し（前処理なし）*CG* 法および *ICCG* 法を適用し，残差 $\|\boldsymbol{r}_k\|$ の減衰の速さを比較せよ．

$$
A = \begin{bmatrix}
8 & 1 & 0 & 1 & & & & & \\
1 & 8 & 1 & 0 & 1 & & & & \\
0 & 1 & 8 & 1 & 0 & 1 & & & \\
1 & 0 & 1 & 8 & 1 & 0 & 1 & & \\
 & \ddots & & \ddots & \ddots & \ddots & & \ddots & \\
 & & 1 & 0 & 1 & 8 & 1 & 0 & 1 \\
 & & & 1 & 0 & 1 & 8 & 1 & 0 \\
 & & & & 1 & 0 & 1 & 8 & 1 \\
 & & & & & 1 & 0 & 1 & 8
\end{bmatrix} \tag{21}
$$

4 共役勾配法の応用—偏微分方程式に対する差分法

ここでは，共役勾配法の応用として，偏微分方程式の差分法による近似解法を考える．具体的には，2 次元正方領域 $\Omega = (0,1)^2$ における Laplace 方程式の境界値問題

$$
\begin{cases}
\Delta u = 0 & 0 < x, y < 1, \\
u(x, y=0) = g_1(x) & 0 < x < 1 \\
u(x=1, y) = g_2(y) & 0 < y < 1 \\
u(x, y=1) = g_3(x) & 0 < x < 1 \\
u(x=0, y) = g_4(y) & 0 < y < 1
\end{cases} \tag{22}
$$

を考える．

差分法 (finite difference method) とは，関数の微分を有限差分に置き換えることによる偏微分方程式の近似解法である．

本実験で扱う問題 22 に対して差分法を具体的に説明する．まず，区間 $0 \leqq x \leqq 1$, $0 \leqq y \leqq 1$ をそれぞれ N 個の小区間に分割し，正方領域 Ω を長方形グリッドに分割する．そして，

$$h = \frac{1}{N}, \quad \begin{cases} x_i = ih, \ i = 0, 1, 2, \ldots, N, \\ y_j = jh, \ j = 0, 1, 2, \ldots, N \end{cases} \tag{23}$$

と置く（図 2 参照）．このとき，Laplace 方程式（(22) 第 1 式）左辺の導関数を有限差分近似することにより，もとの Laplace 方程式は次の代数方程式で近似される

$$\begin{aligned} -\frac{u_{i+1\,j} - 2u_{ij} + u_{i-1\,j}}{h^2} - \frac{u_{i\,j+1} - 2u_{ij} + u_{i\,j-1}}{h^2} = 0, \\ i = 1, 2, \ldots, N-1, \ j = 1, 2, \ldots, N-1, \end{aligned} \tag{24}$$

$$\begin{aligned} u_{i0} = g_1(x_i), \ u_{iN} = g_3(x_i), \quad i = 1, 2, \ldots, N-1, \\ u_{0j} = g_4(y_j), \ u_{Nj} = g_2(y_j), \quad j = 1, 2, \ldots, N-1, \end{aligned} \tag{25}$$

ここで，$u_{ij}, i, j = 1, 2, \ldots, N$ は格子点 $(x_i, y_j) = (ih, jh)$ における解 $u(x_i, y_j)$ の近似値である．2 次元配列 u_{ij} を 1 次元配列 v_k に

$$\begin{aligned} &v_1 = u_{11}, && v_2 = u_{21}, && \ldots, && v_{N-1} = u_{N-1\,1}, \\ &v_N = u_{12}, && v_{N+1} = u_{22}, && \ldots, && v_{2(N-1)} = u_{N-1\,2}, \\ &v_{2(N-1)+1} = u_{13}, && v_{2(N-1)+2} = u_{23}, && \ldots, && v_{3(N-1)} = u_{N-1\,3}, \\ &\ldots \\ &v_{(N-2)(N-1)+1} = u_{1\,N-1}, && v_{(N-2)(N-1)+2} = u_{2\,N-1}, && \cdots, && v_{(N-1)(N-1)} = u_{N-1\,N-1} \end{aligned}$$

（一般に $v_{(N-1)(i-1)+j} = u_{ij}$）

により書き直すと，方程式 (24), (25) は v_k, $k = 1, 2, \ldots$ に対する連立 1 次方程式

$$
\begin{cases}
4v_1 - v_2 - v_N = g_1(x_1) + g_4(y_1), \\
-v_1 + 4v_2 - v_3 - v_{N+1} = g_1(x_2), \\
\cdots \\
-v_{N-3} + 4v_{N-2} - v_{N-1} - v_{(N-2)+(N-1)} = g_1(x_{N-2}), \\
-v_{N-2} + 4v_{N-1} - v_{2(N-1)} = g_1(x_{N-1}) + g_2(y_1), \\
-v_1 + 4v_N - v_{N+1} - v_{2(N-1)+1} = g_4(y_2), \\
-v_2 - v_N + 4v_{N+1} - v_{N+2} - v_{2(N-1)+2} = 0, \\
\cdots \\
-v_{N-2} - v_{2(N-1)-2} + 4v_{2(N-1)-1} - v_{2(N-1)} - v_{3(N-1)-1} = 0, \\
-v_{N-1} - v_{2(N-1)-1} + 4v_{2(N-1)} - v_{3(N-1)} = g_2(y_2), \\
\cdots \\
-v_{(N-3)(N-1)+1} + 4v_{(N-2)(N-1)+1} - v_{(N-2)(N-1)+2} = g_3(x_1) + g_4(y_{N-1}), \\
-v_{(N-3)(N-1)+2} - v_{(N-2)(N-1)+1} + 4v_{(N-2)(N-1)+2} - v_{(N-2)(N-1)+3} = g_3(x_2), \\
\cdots \\
-v_{(N-2)(N-1)-1} - v_{(N-1)(N-1)-2} + 4v_{(N-1)^2-1} - v_{(N-1)^2} = g_3(x_{N-2}), \\
-v_{(N-2)(N-1)} - v_{(N-1)^2-1} + 4v_{(N-1)^2} = g_3(x_{N-1}) + g_2(y_{N-1})
\end{cases}
$$

となり，その係数行列は正値対称疎行列となる．この連立 1 次方程式に対し ICCG 法を適用すれば，v_k の値，そして，格子点における近似解の値 u_{ij} を得る．

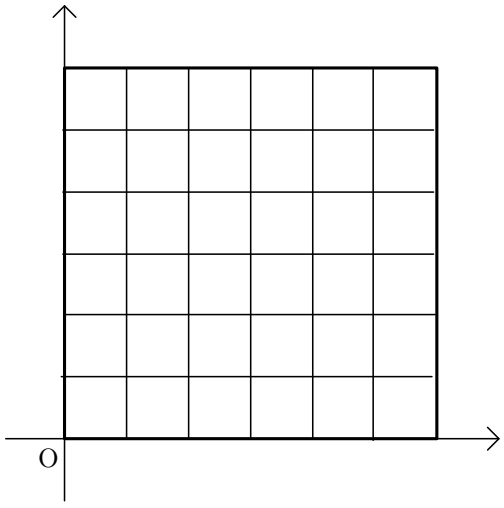

図 2: Square meshes on the domain $\Omega = (0, 1)^2$.

課題 5 方程式 *(24), (25)* より *1* 次元配列 v_k に対する連立 *1* 次方程式を導出し，それを *ICCG* 法で解くことにより *Laplace* 方程式境界値問題 *(22)* の近似解を求めよ．ただし，境界関数 g_1, g_2, g_3, g_4 は各自自由に与えてよい．そして，近似解を *gnuplot* などを用いて *3* 次元グラフで表示せよ．

境界関数

$$\begin{cases} g_1(x) = -\sin \pi x, \quad g_2(y) = 0, \\ g_3(x) = 1 - x, \quad g_4(y) = y^2 \end{cases} \tag{26}$$

に対する計算結果を図 3 に示す．

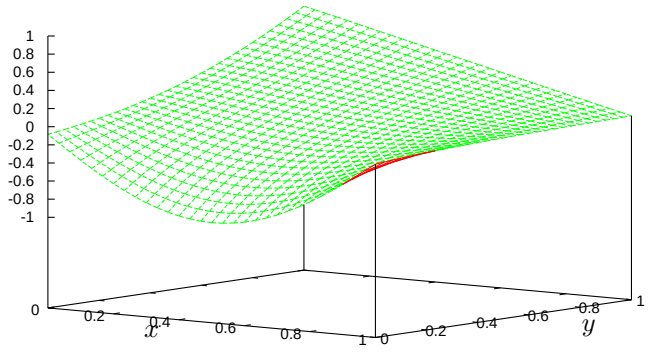

図 3: The approximate solution by the finite difference method of the boundary value problem of the Laplace equation (22) with the boundary data (26).

5　大規模対称疎行列のデータ構造 *

$n \times n$ 大規模疎行列 A はその成分の大部分は 0 なので，行列 A に 2 次元配列 $n \times n$ を用意するのはメモリの無駄遣いである．そこで，行列 A に対するデータ構造を工夫することにより，メモリを節約することを考える．ここでは，

- A の対角成分の値
- A の非零対角成分の位置および値

のみを記憶するようにする．不完全 Cholesky 分解の行列 L, D も A と同様にデータ構造を工夫する．具体的には，表 1 のように配列を用意する．

行列 A は対称行列である．そのため非対角成分は，左下三角部分，右上対角部分どちらか一方を用意して，他方はそれをコピーして用いるようにしたい．具体的なプログラムをつくるのは案

データの内容	プログラム中の変数（配列）名	
A, D の i 番目の対角成分	`ad[i], ld[i]`	
	左下三角部分	右上三角部分
i 行目の非零成分の数	`nl[i]`	`nu[i]`
i 行目の左から k 番目の非零成分の列番号	`jl[i][k]`	`ju[i][k]`
A の i 行目の左から k 番目の非零成分の値	`al[i][k]`	`au[i][k]`
L の i 行目の左から k 番目の非零成分の値	`ll[i][k]`	
L^t の i 行目の左から k 番目の非零成分の値		`lu[i][k]`

表 1: 行列 A のデータ構造.

外難しい．文献 [3] の ICCG 法のプログラムを真似て，次のようにする（左下三角部分を右上三角部分にコピーする場合について記す．ただし，逆に右上三角部分を左下三角部分にコピーする場合も，まったく同じ方法で実行できるので，その使い方もできるよう，下記プログラムでは `n1=nl`, `j1=jl`, `a1=al`, `n2=nu`, `j2=ju`, `a2=au` としている．

算法 7 (非対角成分のデータのコピー) //

```
// convert the information of the lower triangular part
// to the one of the upper triangular part or vise versa
//
void lu_convert(int n1[NDIM+1], int j1[NDIM+1][NB+1], double a1[NDIM+1][NB+1],
                int n2[NDIM+1], int j2[NDIM+1][NB+1], double a2[NDIM+1][NB+1])
{
  int i, j, m;
  int nn2[NDIM+1], jj2[NDIM+1][NB+1];
  double aa2[NDIM+1][NB+1];
  //
  for (i=1; i<=NDIM; ++i) nn2[i] = 0;
  //
  for (i=1; i<=NDIM; ++i)
    for (m=1; m<=n1[i]; ++m)
      {
        j = j1[i][m];
        ++nn2[j];
        jj2[j][nn2[j]] = i;
        aa2[j][nn2[j]] = a1[i][m];
      }
  //
  for (i=1; i<=NDIM; ++i)
    {
      n2[i] = nn2[i];
      for (m=1; m<=n2[i]; ++m)
        {
          j2[i][m] = jj2[i][m];
          a2[i][m] = aa2[i][m];
        }
    }
}
```

課題 6* *1.* 算法 *7*のプログラムを作成し，実際に行列の左下三角部分だけ用意して実行し，実際に対角行列が生成されていることを確かめよ.

2. 行列 A のデータ構造を表 *1* のようにとった場合の（前処理なし）共役勾配法のプログラムを作成し，行列の対角成分・左下三角部分だけ用意して連立 *1* 次方程式を解くようにせよ.

さて，行列 L, D のデータ構造を表 1 のようにとるとき，不完全 Cholesky 分解の算法を実際に

プログラムに書いてみるのは，案外と難しい．これも文献 [3] に習って，算法 8 のようにするとよいだろう．こうして行列 L が得られたならば，行列 A の左下三角成分を算法 7 で右上三角成分にコピーしたように，算法 7 により L の成分を L^t にコピーすればよい．これにより，前処理に必要なデータが揃った．

課題 7* 算法 *8* にしたがい，行列 A の不完全 *Cholesky* 分解のプログラムを作成・実行せよ．さらに，この不完全 *Cholesky* 分解を用いて，行列 A, L, D のデータ構造を表 *1* のようにとった場合の *ICCG* 法のプログラムを作成し，実行せよ．

6 最後に

CG 法に関連した連立 1 次方程式の解法は今日も，非対称行列問題への拡張 (BiCG 法，CGS 法，BiCG-STAB 法，GP-BiCG 法など) をはじめとして，今日も精力的に研究が行われている分野である．CG 法系列の解法の現状については，文献 [1, 4] を参照すること．

算法 8 (不完全 Cholesky 分解（実装版）) //

```
// incomplete Cholesky decomposition and IC preconditioning
//
void incomplete_cholesky_decomp(void)
{
  int i, j, k, k1, mu0, mu, mu1;
  int mui, muj;
  double s, ss, eps = 2.2e-16;
  //
  for (i=1; i<=NDIM; ++i) {
    ld[i] = 0.0;
    for (mu0=1; mu0<=NB; ++mu0)
      ll[i][mu0] = 0.0;
  }
  //
  for (i=1; i<=NDIM; ++i) {
    for (mu=1; mu<=nl[i]; ++mu) {
      j = jl[i][mu];
      mui = muj = 1;
      ll[i][mu] = al[i][mu];
      do {
        if (jl[j][muj] > jl[i][mui]) {
          ++mui;
        } else {
          if (jl[j][muj] < jl[i][mui]) {
            ++muj;
          } else {
            ll[i][mu] -= ld[jl[i][mui]] * ll[i][mui] * ll[j][muj];
            ++mui;
            ++muj;
          }
        }
      } while ((muj<=nl[j]) && (mui<=nl[i]));
    }
    //
    s = ad[i];
    for (mu=1; mu<=nl[i]; ++mu) {
      ss = ll[i][mu];
      s -= ld[jl[i][mu]] * ss * ss;
    }
    if (fabs(s) < eps) s = eps;
    ld[i] = 1.0 / s;
  }
}
```

参考文献

[1] 藤野清次，張紹良：反復法の数理，朝倉書店，1996 年.

[2] M. R. Hestines and E. Stiefel, Methods of conjugate gradients for solving linear systems, *Journal of Research of the National Bureau of Standards*, **49** (1952) 926–943.

[3] 森正武：FORTRAN77 数値計算プログラミング　増補版，岩波書店，1987 年.

[4] 杉原正顯，室田一雄：岩波数学叢書　線形計算の数理，岩波書店，2009 年.