2017 年度桂田研究室卒業研究レポート

反復法での連立一次方程式の解の収束速度

(仮)

明治大学 総合数理学部 現象数理学科
大畑 佑樹

指導教諭：桂田 祐史 准教授

2018 年 2 月 14 日

目次

１　はじめに

今回、私は「反復法での連立一次方程式の収束速度」を卒業研究のテーマとした。

実際の現象において、その数値解析を行うためには連立一次方程式を使用する事が多く、また、スーパーコンピュータの計算速度を決めるのも連立一次方程式の計算速度だということが知られている。

今回の研究では、二次元 Poisson 方程式のモデルを、CG 法や PCG 法を用いた連立一次方程式で数値計算し、その計算速度や精度はどちらの方が上か、実際にプログラムを作成し、検証していく事とした。

２　反復法を用いた連立一次方程式の考察

2.1　Poisson 方程式

$$-\Delta u(x,y) = f(x,y)$$

この式が Poisson 方程式であり、これを連立一次方程式にする方法をについて考えていく。上の式は領域Ω内の座標(*x,y*)における数値 *u=u(x,y)*を求める方程式であり、f : Ω→**R** である。また、ここでの境界条件として、

$$u(x,y) = g(x,y) \quad ((x,y) \in \partial\Omega)$$

を加える。今回は$g(x,y) = 0$として行う事としたので、すなわちΩの周囲が 0 で固定されている状況を考える。今回はΩ=(0,1)×(0,1)と設定し、その各辺を N 等分して格子に区切るため、以下のように文字を置くことにした。

$$h = \frac{1}{N}$$
$$x_i = ih \quad (0 \le i \le N)$$
$$y_j = jh \quad (0 \le j \le N)$$
$$u_{i,j} = u(x_i, y_j) \quad (0 \le i \le N, 0 \le j \le N)$$
$$f_{i,j} = f(x_i, y_j) \quad (1 \le i \le N-1, 1 \le j \le N-1)$$

この時、Poisson 方程式の解である $u_{i,j}$ の近似値を $U_{i,j}$ とすることで以下のように差分化出来る。

$$-\Delta U(x_i, y_j) = -\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right) U(x_i, y_j) = -\frac{1}{h^2} U(x_i, y_j) - \frac{1}{h^2} U(x_i, y_j)$$

$$= -\frac{-U_{i+1,j} + 2U_{i,j} - U_{i-1,j}}{h^2} - \frac{-U_{i,j-1} + 2U_{i,j} - U_{i,j+1}}{h^2}$$

$$= -\frac{-U_{i+1,j} - U_{i-1,j} - U_{i,j+1} - U_{i,j-1} + 4U_{i,j}}{h^2}$$

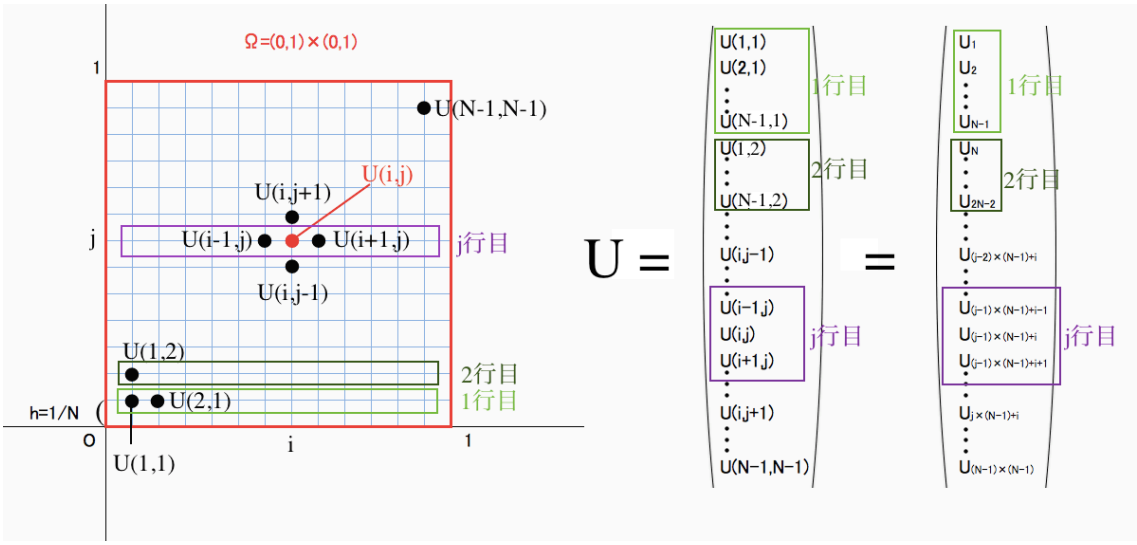
$$= \frac{U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} - 4U_{i,j}}{h^2}$$

よって、Poisson 方程式は以下のようになる。

$$\frac{U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} - 4U_{i,j}}{h^2} = f_{i,j} \quad ((x_i, y_j) \in \Omega \text{ となる(i, j)})$$
$$U_{i,j} = g(x_i, y_j) \quad ((x_i, y_j) \in \partial\Omega \text{ となる(i, j)})$$

ここで、$\varphi(i,j) = (j-1)\times(N-1) + i$とすると、$U_{\varphi(i,j)} = U_{(j-1)\times(N-1)+i}$とする

事が出来るので、U を(N-1)² 次元のベクトル化する事が出来る。



ここで各(i,j)について$\frac{1}{h^2}AU_{i,j} = -f$が成り立つような行列 A を考えると、以下のようになる。

$$C = \begin{pmatrix} -4 & 1 & 0 & & & & \\ 1 & -4 & 1 & \cdots & & 0 & \\ 0 & 1 & -4 & & & & \\ & \vdots & & \ddots & & \vdots & \\ & & & & -4 & 1 & 0 \\ & 0 & & \cdots & 1 & -4 & 1 \\ & & & & 0 & 1 & -4 \end{pmatrix}, \quad I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & & \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & 1 \quad 0 \\ & & & 0 \quad 1 \end{pmatrix}$$

$$A = \begin{pmatrix} C & I & O & & & & \\ I & C & I & \cdots & & O & \\ O & I & C & & & & \\ & \vdots & & \ddots & & \vdots & \\ & & & & C & I & O \\ & O & & \cdots & I & C & I \\ & & & & O & I & C \end{pmatrix}$$

この A を用いて AU=−h²f とすることで、U に関する連立一次方程式が完成する。

また、i=1、i=N-1、j=1、j=N-1 の少なくとも 1 つを満たす $U_{i,j}$ について計算する際に、境界条件を用いる必要がある。すなわち、

$$U_{i,0} = g_i\,, U_{i,N} = g_{N\times(N+1)+i}\,, U_{0,j} = g_{j\times(N+1)}\,, U_{N,j} = g_{j\times(N+1)+N}$$

とおき、右辺に移項する。すると、

$$AU = -F, F = h^2 f - g$$

という形とすることができる。この形を用いて後に連立一次方程式で解を求めていく。

2.2　非定常反復法

連立一次方程式の反復法には定常反復法と非定常反復法の２種類がある。今回は使用してはいないが、定常反復法には jacobi 法、SOR 法、Gauss-Sidel 法などの解法が存在する。一方非定常反復法は、係数行列 A が正定値対称行列である場合と正定値対称行列でない場合で分かれていて、正定値対称行列である場合には共役勾配法(CG 法)が決定版とされている。今回はその共役勾配法と、行列に前処理を加え共役勾配法を行う PCG 法の２つについて取り組む事とした。

2.2.1　共役勾配法(CG 法)

連立一次方程式 Ax=b について、A は N 次正定値対称行列であり、$b \in R^N$ である。

このとき、以下の記号について定義する。

① $X, Y \in R^N$で、

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \qquad y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

としたとき、

$$(x\,, y) = y^t x = \sum_{i=1}^{n} x_i y_i\,, \left|\left|x\right|\right| = \sqrt{(x\,, x)} = (x_1^2 + \cdots + x_n^2)^{1/2}$$

$$< x\,, y > = (Ax\,, y), \left|\left|\left|x\right|\right|\right| = \sqrt{< x\,, x >}$$

② x^*を連立一次方程式 Ax=b の真の解としたとき、

$$x^* = A^{-1}b$$

$$\phi(x) = \frac{1}{2}\left|\left|\left|x - x^*\right|\right|\right|^2 = \frac{1}{2}(A(x - x^*), x - x^*)$$

③ N 次正方行列 A、N 次ベクトル v から生成される以下のような線形部分空間を Krylov 部分空間という。

$$\mathrm{K}_k(A, v) = Span(v, Av, A^2v, \cdots, A^{k-1}v)$$

ここで、

$$\phi(x_0) > \phi(x_1) > \phi(x_2) > \cdots$$

となるように点列{x_n}(n≧0)を取ることで計算を行う。この時、

$$\lim_{n\to\infty} x_n = x^*$$

となることがわかる。ここで最後まで計算せず、十分大きな番号 n について x_n を x^*の近似とする方法を***傾斜法***と言う。これを以下のように用いる事で、連立一次方程式を解く事が出来る。

1、　探索方向 $p_k \neq 0$ を定める。 2、　x_k が求まっている時、$x_{k+1} = x_k + \alpha_k p_k$ として、α_k を $\phi(x_{k+1})$が最小になるように取る。

これを***逐次最小化法***と言う。

この時、

$$\phi(x_{k+1}) = \phi(x_k + \alpha_k p_k) = \frac{1}{2}\left|\left|(x_k + \alpha_k p_k) - x^*\right|\right|^2 \geq 0$$

であることから、α_k を $\phi(x_{k+1})$が最小になるように選ぶようにすると、

$$\phi(x_{k+1}) = 0$$

とした時の α_k を用いる。この式を展開すると、以下のようになる。

$$\frac{1}{2}\left|\left|(x_k + \alpha_k p_k) - x^*\right|\right|^2 = \frac{1}{2}\left(\left|\left|x_k - x^*\right|\right|^2 - 2\alpha_k\langle x^* - x_k\,, p_k\rangle + {\alpha_k}^2\left|\left|p_k\right|\right|^2\right) = 0$$

これに解の公式を使って α_k の値を求めると、

$$\alpha_k = \frac{\langle x^* - x_k\,, p_k\rangle}{\left|\left|p_k\right|\right|^2} = \frac{(r_k\,, p_k)}{\left|\left|p_k\right|\right|^2}$$

となる。つまり、

$$x_{k+1} = x_k + \alpha_k p_k\,, \alpha_k = \frac{(r_k\,, p_k)}{\left|\left|p_k\right|\right|^2}$$

として点列{x_n}(n≧0)を取る事が出来る。

次に探索方向の取り方として、***共役方向法***と呼ばれる方法を用いる事とした。***共役方向法***とは、以下の様な条件のもと{p_j}(j=0,1,⋯,k−1)を取る方法である。

$$\langle p_i\,, p_j\rangle = 0 \ \ (0 \leq i < j \leq k-1)$$

この時、以下が成り立つ。

$$\phi(x_k) = \min_{x\in D_k}\phi(x)\ , D_k = x_0 + Span(p_0, p_1, \cdots, p_{k-1})$$

また、n=k とするとき、

$$r_k = b - Ax_k = A(x^* - x_k)$$

で構成される r_k を***残差***と呼ぶ。この残差r_kと共役方向法で得られた$\{p_j\}$について、以下の事が分かる。

$$(r_k, p_j) = 0(j = 0,1,\cdots,k-1)\ , < r_k, p_i >= 0(i = 0,1,\cdots,k-2)$$

ここで先ほどの条件のもと探索方向 p_k を求める場合、残差を用いて Gram - Schmidt の直行化を施せば良い。つまり、

k=0 の場合

$$p_0 = r_0$$

k≧1 の場合

$$p_k = r_k - \sum_{i=0}^{k-1} \frac{< r_k, p_i >}{\left|\left||p_i|\right|\right|^2} p_i = r_k - \frac{< r_k, p_{k-1} >}{\left|\left||p_{k-1}|\right|\right|^2} p_{k-1}$$

このように探索方向 p_k を設定すると、先ほどの条件が満たされる。

ここまでの解ベクトル、残差、探索方向の式についてまとめると、次のようになる。

$$\begin{cases} x_{j+1} = x_j + \alpha_j p_j\ , \alpha_j = \dfrac{(r_j\ , p_j)}{\left|\left||p_j|\right|\right|^2} \\ r_{j+1} = b - Ax_{j+1} \\ p_{j+1} = r_{j+1} - \dfrac{< r_{j+1}, p_j >}{\left|\left||p_j|\right|\right|^2} p_j \end{cases}$$

ここで残差r_kについて以下のように変形する。

$$r_{k+1} = b - Ax_{k+1} = b - A(x_k + \alpha_k p_k) = b - Ax_k - \alpha_k Ap_k = r_k - \alpha_k Ap_k$$

このα_kはx_{k+1}を求める段階で既に導出されているので、それを利用する事によって元の式より計算回数を少なくする事が出来る。よって、上の式は次のように変化する。

$$\begin{cases} x_{j+1} = x_j + \alpha_j p_j \\ r_{j+1} = r_j - \alpha_j Ap_j \end{cases}, \alpha_j = \frac{(r_j\ , p_j)}{\left|\left||p_j|\right|\right|^2}$$
$$p_{j+1} = r_{j+1} - \frac{< r_{j+1}, p_j >}{\left|\left||p_j|\right|\right|^2} p_j$$

この式に補助ベクトルなどを交えて計算しやすくすると、以下のようなアルゴリズムで解を求める事が出来る。

初期ベクトル x_0 をとり、目標とする相対残差 ε を決める。

$r_0 = b - Ax_0$

$p_0 = r_0$

for k=0,1,… until $||r_k|| \leqq \varepsilon ||b||$ do

begin

$c_k = Ap_k$

$q_k = (p_k, c_k)$

$\alpha_k = \frac{(r_k, p_k)}{q_k}$

$x_{k+1} = x_k + \alpha_k p_k$

$r_{k+1} = r_k - \alpha_k c_k$

$\beta_k = -\frac{(r_{k+1}, c_k)}{q_k}$

$p_{k+1} = r_{k+1} + \beta_k p_k$

end

この時の停止条件として$||r_k|| \leq \varepsilon ||b||$を使用しているが、この ε は相対残差である。この ε が小さければ小さいほど精度が高くなる。

これを繰り返し行う事で、連立一次方程式の解x^*を求める。

これが共役勾配法(CG 法)である。

2.2.2 前処理付き CG 法(PCG 法)

これまで連立一次方程式 Ax=b における係数行列 A をそのまま使用して解を求めていたが、A を計算しやすい形に変形する事で、より CG 法の計算を速くすることが出来る。その変形の事を『前処理』という。

適当な正則行列 C を用いて、連立一次方程式を

$Ax = b \leftrightarrow AC^{-T}C^T x = b \leftrightarrow C^{-1}AC^{-T}C^T x = C^{-1}b \leftrightarrow (C^{-1}AC^{-T})(C^T x) = (C^{-1}b)$

と変形出来る。この時、

$$\begin{cases} C^{-1}AC^{-T} = \tilde{A} \\ C^T x_k = \widetilde{x_k} \\ C^{-1}b = \tilde{b} \end{cases}$$

というように書き換えることで、以下のように前処理を含んだ新たな連立一次方程式を作る事が出来る。

$$(C^{-1}AC^{-T})(C^T x) = (C^{-1}b) \leftrightarrow \tilde{A}\tilde{x} = \tilde{b}$$

この C を前処理行列と言い、ここで新たに導出された$\tilde{A} = C^{-1}AC^{-T}$が単位行列 E に近ければ近いほど、つまり CC^T が A に近ければ近いほど収束は早くなる。

次に、以下のように残差と探索方向をおく。

$$\tilde{r_k} = C^{-1}r_k$$
$$\tilde{p_k} = C^T p_k$$

すると、CG 法で用いられた式は次のように変化する。

$$\widetilde{\alpha_j} = \frac{(\widetilde{r_j}, \widetilde{p_j})}{\left|\left|\widetilde{p_j}\right|\right|^2} = \frac{(\widetilde{r_j}, \widetilde{p_j})}{(\tilde{A}\tilde{p_j}, \tilde{p_j})} = \frac{(C^{-1}r_j, C^T p_j)}{((C^{-1}AC^{-T})C^T p_j, C^T p_j)} = \frac{(C^{-1}r_j, C^T p_j)}{(C^{-1}Ap_j, C^T p_j)}$$

$$= \frac{(C^T p_j)^T C^{-1} r_j}{(C^T p_j)^T C^{-1} A p_j} = \frac{p_j^T C C^{-1} r_j}{p_j^T A^T C^{-T} C^T p_j} = \frac{p_j^T C r_j}{p_j^T A^T p_j} = \frac{(r_j, p_j)}{\left|\left|p_j\right|\right|^2}$$

$$\widetilde{\beta_j} = \frac{< \widetilde{r_{j+1}}, \widetilde{p_j} >}{\left|\left|\tilde{p_j}\right|\right|^2} = \frac{(\tilde{A}\widetilde{r_{j+1}}, \widetilde{p_j})}{\left|\left|p_j\right|\right|^2} = \frac{(\widetilde{r_{j+1}}, \tilde{A}\widetilde{p_j})}{\left|\left|p_j\right|\right|^2} = \frac{(C^{-1}r_{j+1}, (C^{-1}AC^{-T})C^T p_j)}{\left|\left|p_j\right|\right|^2}$$

$$= \frac{(C^{-1}r_{j+1}, C^{-1}Ap_j)}{\left|\left|p_j\right|\right|^2} = \frac{p_j^T A^T C^{-T} C^{-1} r_{j+1}}{\left|\left|p_j\right|\right|^2} = \frac{p_j^T A^T (CC^T)^{-1} r_{j+1}}{\left|\left|p_j\right|\right|^2}$$

$$= \frac{((CC^T)^{-1}r_{j+1}, Ap_j)}{\left|\left|p_j\right|\right|^2}$$

$$\widetilde{x_{j+1}} = \widetilde{x_j} + \widetilde{\alpha_j}\widetilde{p_j} \leftrightarrow C^T x_{j+1} = C^T x_j + \alpha_j C^T p_j \leftrightarrow x_{j+1} = x_j + \frac{< r_j, p_j >}{\left|\left|p_j\right|\right|^2} p_j$$

$$\widetilde{r_j} = \tilde{b} - \tilde{A}\widetilde{x_j} \leftrightarrow C^{-1}r_j = C^{-1}b - C^{-1}AC^{-T}C^T x_j \leftrightarrow r_j = b - Ax_j$$

$$\widetilde{p_j} = \widetilde{r_j} - \tilde{\beta_j}\widetilde{p_{j-1}} \leftrightarrow C^T p_j = C^{-1}r_j - \widetilde{\beta_j} C^T p_{j-1} \leftrightarrow p_j = (CC^T)^{-1}r_j - \widetilde{\beta_j} p_{j-1}$$

解ベクトルと残差には何の変更も無く、探索方向 p_k の中での残差が用いられている部分に、左から$(CC^T)^{-1}$をかければ良いだけとなった。
この事を踏まえて先ほどの CG 法のアルゴリズムを前処理を用いて計算出来るように変換すると、次のようになる。

初期ベクトル x_0 をとり、目標とする相対残差 ε を決める。

$r_0 = b - Ax_0$

$p_0 = (CC^T)^{-1} r_0$

for k=0,1,… until $||r_k|| \leqq \varepsilon ||b||$ do

begin

$c_k = Ap_k$

$q_k = (p_k, c_k)$

$\alpha_k = \frac{(r_k, p_k)}{q_k}$

$x_{k+1} = x_k + \alpha_k p_k$

$r_{k+1} = r_k - \alpha_k c_k$

$\beta_k = -\frac{((CC^T)^{-1} r_{k+1}, c_k)}{q_k}$

$p_{k+1} = (CC^T)^{-1} r_{k+1} + \beta_k p_k$

end

今回はこのアルゴリズムを用いて CG 法と計算速度の比較を行い、その結果をまとめる事とした。

2.2.3　高橋秀俊版 CG 法

共役勾配法のアルゴリズムを以下のように工夫し、変化させる事で、計算回数を少なくしながら同等の結果が得られる事が分かっている。

これを高橋秀俊版 CG 法と言う。

初期ベクトル x_0 をとる。目標とする相対残差 ε を決める。

$r_0 = b - Ax_0$ ， $\beta_0 = {}^{1}/_{(r_0\,,\ r_0)}$ ， $p_0 = \beta_0\, r_0$

for k=0,1,…until $||r_k|| \le \varepsilon||b||$ do

begin

$\alpha_k = {}^{1}/_{(p_k, Ap_k)}$

$x_{k+1} = x_k + \alpha_k p_k$

$r_{k+1} = r_k - \alpha_k A p_k$

$\beta_k = {}^{1}/_{(r_{k+1}\,,\ r_{k+1})}$

$p_{k+1} = p_k + \beta_k r_{k+1}$

end

また、これについて前処理を行う事で、以下のようにアルゴリズムが変化する。

初期ベクトル x_0 をとる。目標とする相対残差 ε を決める。

$r_0 = b - Ax_0$ ， $\beta_0 = {}^{1}/_{(\,(CC^T)^{-1}r_0\,,\ r_0)}$ ， $p_0 = \beta_0\,(CC^T)^{-1}r_0$

for k=0,1,…until $||r_k|| \le \varepsilon||b||$ do

begin

$\alpha_k = {}^{1}/_{(p_k, Ap_k)}$

$x_{k+1} = x_k + \alpha_k p_k$

$r_{k+1} = r_k - \alpha_k A p_k$

$\beta_k = {}^{1}/_{(\,(CC^T)^{-1}r_{k+1}\,,\ r_{k+1})}$

$p_{k+1} = p_k + \beta_k\,(CC^T)^{-1}r_{k+1}$

end

これを用いる事で、通常の CG 法・PCG 法より早く計算出来る。

2.3　不完全コレスキー分解

不完全コレスキー分解とは、修正コレスキー分解を不完全に行うことで、A≒LDL^Tとなる左下三角行列 L と対角行列 D を求める手段である。今回はこれを用いて前処理行列 C を導出し、実際にプログラムに組み込み計算を行う事とした。

2.3.1　修正コレスキー分解

正定値対称行列 A の k 列目までに対してガウスの消去法を用いた行列を

$$A^{(k)} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1k}^{(1)} & \cdots & a_{1N}^{(1)} \\ & a_{22}^{(2)} & \cdots & a_{2k}^{(2)} & \cdots & a_{2N}^{(2)} \\ & & \ddots & \vdots & & \vdots \\ & & & a_{kk}^{(k)} & \cdots & a_{kN}^{(k)} \\ & 0 & & \vdots & \ddots & \vdots \\ & & & a_{Nk}^{(k)} & \cdots & a_{NN}^{(k)} \end{pmatrix}$$

として、A を LU 分解すると、$A^{(n)}$が上三角行列になる事から、

$$L = \begin{pmatrix} 1 & & & \\ \frac{a_{21}^{(1)}}{a_{11}^{(1)}} & 1 & & 0 \\ \frac{a_{31}^{(1)}}{a_{11}^{(1)}} & \frac{a_{32}^{(2)}}{a_{22}^{(2)}} & 1 & \\ \vdots & \vdots & \ddots & \ddots \\ \frac{a_{n1}^{(1)}}{a_{11}^{(1)}} & \frac{a_{N2}^{(2)}}{a_{22}^{(2)}} & \cdots & \frac{a_{N,N-1}^{(N-1)}}{a_{N-1,N-1}^{(N-1)}} \quad 1 \end{pmatrix}, A^{(N)} = U = \begin{pmatrix} a_{11}^{(1)} & a_{21}^{(1)} & \cdots & a_{1N}^{(1)} \\ & a_{22}^{(2)} & \cdots & a_{2N}^{(2)} \\ & 0 & \ddots & \vdots \\ & & & a_{NN}^{(N)} \end{pmatrix}$$

となる。ここで対称行列 A を LU 分解して得られる上の式について、U の対角成分を D としたとき、

$$U = DL^T$$

という関係がある事が明らかなので、これを利用し、

$$A = LU = LDL^T$$

の形に変えることができる。これを修正コレスキー分解と言う。この時、

$$\begin{cases} \sum_{j=1}^{i} l_{kj} d_j l_{ij} = a_{ki} \quad i = 1,2,\cdots,k-1 \\ \sum_{i=1}^{k} l_{ki} d_i l_{ki} = a_{kk} \end{cases}$$

という式が成り立つ。また、$l_{ii}=1$ であることに注意すると、

$$\begin{cases} l_{ki} d_i = a_{ki} - \sum_{j=1}^{i-1} l_{kj} d_j l_{ij} \quad i = 1,2,\cdots,k-1 \\ d_k = a_{kk} - \sum_{i=1}^{k-1} {l_{ki}}^2 d_i \end{cases}$$

と変形する事が出来る。また、各 k について$\omega_i^{(k)} = l_{ki} d_i$とおくと、L の成分および D の成分は以下の手順で求められる。

```
for k=1,2,…until k>N do
  begin
    for i=1,2,…until i>k-1 do
      begin
        ω_i^(k) = a_ki − Σ_{j=1}^{i−1} ω_j^(k) l_ij
        l_ki = ω_i^(k) × d_i^{−1}
      end
    d_k^{−1} = (a_kk − Σ_{i=1}^{k−1} ω_i^(k) l_ki)^{−1}
  end
```

このアルゴリズムを用いることにより、A を修正コレスキー分解する事が可能である。

2.3.2 不完全コレスキー分解

不完全コレスキー分解は、修正コレスキー分解を不完全に行ったものである。「不完全に」というのは、修正コレスキー分解を行う事で導出した L の項である l_{ij} について、ある条件 P を満たしていた場合は計算をせず、強制的に 0 とおく、ということである。

例えば以下のような条件を満たす(i,j)の項を強制的に 0 と置いて修正コレスキ

一分解を行うと、左下三角行列を不完全に作成する事ができる。

$$P = \{(i,j)|a_{ij} = 0\}$$

これは元の行列 A の(i,j)の項が 0 であった場合、L の(i,j)の項を強制的に 0 にする、ということである。このように計算する事で $A \fallingdotseq LDL^T$ となる L を求める事が出来て、また $C=LD^{1/2}$ とすることで、$CC^T \fallingdotseq A$ とする C を作成する事が出来る。これを用いて

$$C^{-1}AC^{-T} = (LD^{\frac{1}{2}})^{-1}A(LD^{\frac{1}{2}})^{-T} = \tilde{A}$$

として$\tilde{A}$を求める事が出来る。ここで $CC^T \fallingdotseq A \Leftrightarrow E \fallingdotseq C^{-1}AC^{-T}$ であるため、$\tilde{A}$は単位行列に近いという事が分かる。

2.3.3　前処理行列を含んだ残差の計算

PCG 法の計算で、前処理行列を用いるのは共役方向を求める際の

$$p_{k+1} = (CC^T)^{-1}r_{k+1} - \frac{((CC^T)^{-1}r_{k+1}, Ap_k)}{(p_k, Ap_k)}p_k$$

のみである。ここで

$$(CC^T)^{-1}r_{k+1} = v_{k+1}$$

とした時、

$$r_{k+1} = (CC^T)v_{k+1}$$

と変形出来る。また、

$$y_{k+1} = C^T v_{k+1}$$

とすることで、

$$y_{k+1} = C^T v_{k+1}, r_{k+1} = Cy_{k+1}$$

という形にする事が出来る。この時、C は下三角行列、C^Tは上三角行列であるため、計算を楽に行う事が出来る。

・$r_{k+1} = Cy_{k+1}$の計算アルゴリズム

```
for i=1,2,…  until i>N do
 begin
  y_{k+1}[i] = (r_{k+1}[i] − Σ_{j=1}^{i−1} y_{k+1}[j] * C[i][j])/C[i][i]
 end
```

・$y_{k+1} = C^T v_{k+1}$の計算アルゴリズム

for i=N,N-1,… until i<1 do
begin

$$v_{k+1}[i] = (y_{k+1}[i] - \sum_{j=i+1}^{N} v_{k+1}[j] * C[j][i])/C[i][i]$$

end

これを連続して行う事で、$(CC^T)^{-1} r_{k+1} = v_{k+1}$となる$v_{k+1}$を導出する事が出来る。

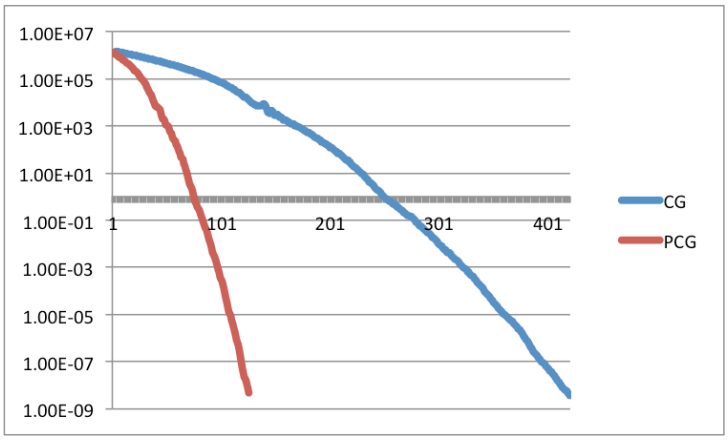
2.4　CG 法・PCG 法比較

これまで CG 法と PCG 法のアルゴリズムの導出や、Poisson 方程式を連立一次方程式に変換する流れなどをまとめてきたが、今度はそれを C を用いて実際に PC 上で計算し、その残差はどのように減っていくか、また Poisson 方程式は $(N-1)^2$ 行の解ベクトル x_k について計算する事になっているが、その N の値を変更する事で、相対残差を用いた条件に該当し、停止するまでに何回ループを行うか等を調べてみた。

まずは残差について比較してみる。

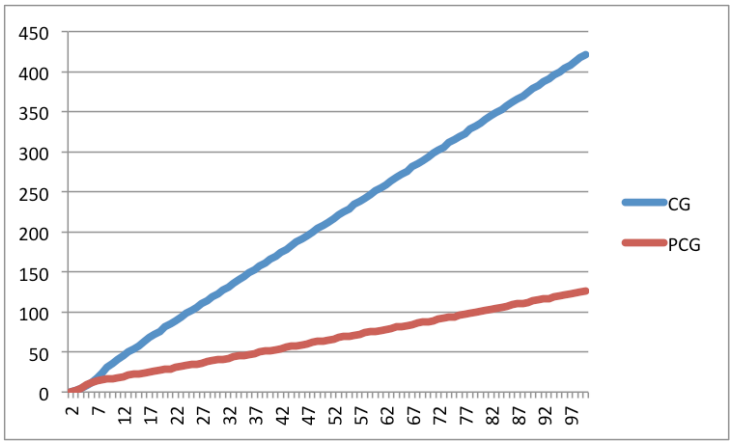
$\varepsilon = 1.0 \times 10^{-14}$, N=100 として設定し、収束するまでループを行うと、以下のように残差は減少していった。

(横軸：停止するまでのループ回数、縦軸：残差のノルムの値)



これを見ると、PCG 法は CG 法の大体 3～4 倍の早さで残差が減っていく事がわかる。

次に N の値を増やしていった場合、何回目のループで初めて停止するかをグラフにすると、以下のようになった。
(横軸：Poisson 方程式での N の値、縦軸：停止するまでのループ回数)



こちらも先ほどと同様、PCG 法の方が明らかに収束するまでが早い。
これらのことから、Poisson 方程式に置いては連立一次方程式をそのまま計算するのではなく、前処理を用いて PCG 法で計算する方が早く収束することが分かる。しかし実際にプログラムを動かしてみると、前処理に時間がかかることが多く、前処理のためのプログラム作成も今後の課題となるだろう。

2.5　収束速度について

今回、連立一次方程式 Ax=b について計算する方法として CG 法と PCG 法を用いたが、残差で比較してもループ回数で比較しても PCG 法の方が少ないループ回数で収束する事が分かっている。この収束速度についてチェビシェフ多項式を用いて導出されるある法則があるので、それについて記述する事とした。
まずチェビシェフ多項式とは、

$$T_n(x) = \cos\left(n * cos^{-1}(x)\right)\ (-1 \le x \le 1)$$

で定義される多項式の事を言い、三角関数の公式

$$\cos(n+1)\,\theta + \cos(n-1)\,\theta = 2\cos \mathrm{n}\theta \cos\theta$$

を$\cos\theta = x\ (\theta = \cos^{-1}x)$として利用すると、

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

という漸化式を得る。この漸化式を解いて一般項を求めると、

$$T_k(x) = \frac{1}{2}\left\{\left(x + \sqrt{x^2-1}\right)^k + \left(x - \sqrt{x^2-1}\right)^k\right\}, |x| > 1$$

という式が得られる。この時、一般項は$|T_k(x)| < 1$となる。ここで行列 A の固有値を$\lambda_i\ (1 \le i \le N)$とした時、

$$\rho = \frac{\lambda_{MAX}}{\lambda_{min}}\ , \lambda_{MAX} = \max_{1 \le i \le N} \lambda_i\ , \lambda_{min} = \min_{1 \le i \le N} \lambda_i$$

となるρを用いて$x = \frac{\rho+1}{\rho-1}$としてチェビシェフ多項式の一般項に代入すると、

$$\frac{\rho+1}{\rho-1} \pm \sqrt{\left(\frac{\rho+1}{\rho-1}\right)^2 - 1} = \frac{\rho+1}{\rho-1} \pm \frac{\sqrt{(\rho+1)^2 - (\rho-1)^2}}{\rho-1} = \frac{\rho+1 \pm 2\sqrt{\rho}}{\rho-1}$$

$$= \frac{(\sqrt{\rho} \pm 1)^2}{(\sqrt{\rho}+1)(\sqrt{\rho}-1)} = \frac{\sqrt{\rho} \pm 1}{\sqrt{\rho} \mp 1}$$

これを用いると、

$$T_k\left(\frac{\rho+1}{\rho-1}\right) = \frac{1}{2}\left\{\left(\frac{\sqrt{\rho}+1}{\sqrt{\rho}-1}\right)^k + \left(\frac{\sqrt{\rho}-1}{\sqrt{\rho}+1}\right)^k\right\} \ge \frac{1}{2}\left(\frac{\sqrt{\rho}+1}{\sqrt{\rho}-1}\right)^k$$

という不等式を作成する事が出来る。
また、新たに$\varphi(x), h(x), \mu$を導入し、以下のように定義する。

$$\varphi(x) = \frac{1}{\mu} T_k(h(x))\ , h(x) = \frac{\lambda_{MAX} + \lambda_{min} - 2x}{\lambda_{MAX} - \lambda_{min}}\ , \mu = T_k(h(0))$$

この時、$\lambda_{min} \le x \le \lambda_{MAX}$を満たす x について、

$$|\varphi(x)| \le \frac{1}{\mu}$$

という事が分かる。
この他に、μについては次のような事が分かる。

$$\mu = T_k(h(0)) = T_k\left(\frac{\lambda_{MAX} + \lambda_{min}}{\lambda_{MAX} - \lambda_{min}}\right) = T_k\left(\frac{\rho+1}{\rho-1}\right) \ge \frac{1}{2}\left(\frac{\sqrt{\rho}+1}{\sqrt{\rho}-1}\right)^k$$

次に、n 次の多項式である$\varphi(t)\ (\varphi(0) = 1)$を用いることで表される、以下のような命題を利用する。

$$\left|\left||x_k - x^*|\right|\right| = \left|\left||x^*|\right|\right| |\varphi(\lambda_{MAX})|$$

[証明]
まず各$x_k \in D_k$について次の事が成り立つ。

$$\phi(x_k) = \min_{x \in D_k} \phi(x) \leftrightarrow \left|\left|\left|x_k - x^*\right|\right|\right| = \min_{x \in D_k} \left|\left|\left|x - x^*\right|\right|\right|$$

次に以下のように記号を定義する。

$$g(t) = \big(1 - \varphi(t)\big)t^{-1}, y = g(A)b$$

この y について、$y \in D_k = \mathrm{K}_k(A, b)$であり、以下が成立する。

$$y - x^* = g(A)b - x^* = \big(1 - \varphi(A)\big)A^{-1}b - x^* = \big(1 - \varphi(A)\big)x^* - x^* = -\varphi(A)x^*$$

$$\left|\left|\left|x_k - x^*\right|\right|\right| = \min_{x \in D_k} \left|\left|\left|x - x^*\right|\right|\right| \leq \left|\left|\left|y - x^*\right|\right|\right|$$

ここで A の j 番目の固有値である λ_j に対応した固有ベクトルを正規直交化して作られた v_j を用いて$c_j = \big(x^*, v_j\big)$とおくと、$x^* = \sum_{j=1}^{n} c_j v_j$と書ける。これを利用すると、

$$\varphi(A)x^* = \sum_{j=1}^{n} c_j \varphi(A) v_j = \sum_{j=1}^{n} c_j \varphi\big(\lambda_j\big) v_j$$

$$\left|\left|\left|\varphi(A)x^*\right|\right|\right| = \big(A\varphi(A)x^*, \varphi(A)x^*\big) = \left(\sum_{j=1}^{n} c_j \varphi\big(\lambda_j\big) v_j\right)^T A \left(\sum_{j=1}^{n} c_j \varphi\big(\lambda_j\big) v_j\right)$$

$$= \left(\sum_{j=1}^{n} c_j \varphi\big(\lambda_j\big) v_j\right)^T \left(\sum_{j=1}^{n} c_j \varphi\big(\lambda_j\big) A v_j\right)$$

$$= \left(\sum_{j=1}^{n} c_j \varphi\big(\lambda_j\big) v_j\right)^T \left(\sum_{j=1}^{n} c_j \varphi\big(\lambda_j\big) \lambda_j v_j\right)$$

$\big(v_i, v_j\big) = 1(i = j), \big(v_i, v_j\big) = 0(i \neq j)$ であるため、

$$\left|\left|\left|\varphi(A)x^*\right|\right|\right| = \sqrt{\sum_{j=1}^{n} c_j^2 \lambda_j \varphi^2\big(\lambda_j\big)}$$

これらより、

$$\left|\left||x_k - x^*|\right|\right| \leq \left|\left||y - x^*|\right|\right| = \left|\left||-\varphi(A)x^*|\right|\right| = \left|\left||\varphi(A)x^*|\right|\right| = \sqrt{\sum_{j=1}^{n} c_j^2 \lambda_j \varphi^2(\lambda_j)}$$

$$\leq \varphi(\lambda_{MAX}) \sqrt{\sum_{j=1}^{n} c_j^2 \lambda_j} = \left|\left||x^*|\right|\right| |\varphi(\lambda_{MAX})|$$

これにより先ほどの命題は成立する。

したがって、

$$\left|\left||x_k - x^*|\right|\right| \leq \left|\left||x^*|\right|\right| |\varphi(\lambda_{MAX})| \leq \frac{1}{\mu}\left|\left||x^*|\right|\right| \leq 2\left(\frac{\sqrt{\rho}-1}{\sqrt{\rho}+1}\right)^k \left|\left||x^*|\right|\right|$$

これにより、

$$\left|\left||x_k - x^*|\right|\right| \leq 2\left(\frac{\sqrt{\rho}-1}{\sqrt{\rho}+1}\right)^k \left|\left||x^*|\right|\right|$$

が示された。これはρが小さければ小さいほど、すなわち固有値の幅が小さければ小さいほど反復法の収束が早くなる、という事を表している。
前処理をして得られた$\tilde{A}$は元の A より単位行列 E に近くなっており、単位行列に近ければ近いほど固有値も全て 1 に近づく、ということになるので、前処理をする事でρが小さくなることが分かる。そのために収束するまでの残差の減少ペースや、相対残差を用いた終了条件によって停止するまでが早い、ということになる。

3　まとめと展望

今回、Poisson 方程式から得られる連立一次方程式を CG 法・PCG 法を用いて解く事により、正定値対称行列を用いて作られる連立一次方程式の解の導出までの流れを確認する事が出来た。また、その計算を実際に C のプログラムを用いて解き、その収束ペースや停止するまでのループ回数を可視化する事で、数学・情報の両分野において関わりを見せ、相互理解が深まったように思う。

しかし、時間がなかったため最終的には基本的な Poisson 方程式しか解く事が出来ず、また C で作成したプログラムも基本的な内容はできてはいるものの、N の値が大きい場合に対して計算までに時間がかかっていた。もうすこし効率的なプログラムを作成する方法も存在しているのだろうが、そこまで至れなかったのが残念である。

連立一次方程式は様々な分野において利用されていて、その正確性、スピードが重要になってくる。そのために今後もより早く、正確に解く方法が研究されていく事であろう。

4　参考文献

[1]齊藤 宣一 著 「数値解析入門」 東京大学出版会(2012)
[2]齊藤 宣一 著 「数値解析」 共立出版(2017)
[3]森 正武 著「数値計算プログラミング」岩波コンピュータサイエンス(1986)
[4]森 正武 著 「数値解析」 共立出版(2002)
[5]名取 亮 著 「線形計算」 朝倉書店(1993)
[6]桂田 祐史 著 「連立１次方程式Ⅱ」