連立一次方程式の数値解法について

菖蒲迫 健介 (@九州大学) [*1]

(2023/06/02)

■ はじめに

楕円型偏微分方程式 (例：ラプラス方程式) の近似解を求めるには，離散化して得られる N 次元の連立一次方程式 ($A\boldsymbol{x} = \boldsymbol{b}$) を解く必要がある．ここで N は離散点の数 (計算点の数) であり，N が大きいほど解像度が高くなる．例えば，2 次元の正方形領域を計算対象とし，1 辺に 10 点の計算点を用いる場合，$N = 100$ となる．この時，各点の解を求めようと思ったら 100 個の式を連立して解かなければならないので，何かしらの工夫が必要である．

こんな時に役に立つのがコンピュータによる高速な数値解析である．連立一次方程式の数値解法には大きく分けて 2 種類の方法がある．一方は**直接解法**と呼ばれ，行列の基本変形を行うことで解を求める．もう一方は**反復解法**と呼ばれ，ある演算を繰り返し行うことで解を求める．各解法の中に様々な手法 (ソルバー) が存在する．本ノートでは連立一次方程式の有名な数値解法の一部をまとめ，そのメリットやデメリットを実装を通じて理解することを目標とする．

目次

[*1] 九州大学大学院 理学府地球惑星科学専攻 地球内部ダイナミクス研究室 博士 1 年.

1　直接解法

本質的には掃き出し法と同等である．ここでは，ガウス・ジョルダン法 (Gauss-Jordan method)，ガウスの消去法 (Gaussian elimination method)，LU 分解 (LU decomposition) の 3 つの方法を述べる．簡単のため，係数行列 A が 3×3 である連立一次方程式 ($A\boldsymbol{x} = \boldsymbol{b}$) を考える[*2]．

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \tag{1.1}$$

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned} \tag{1.2}$$

1.1　ガウス・ジョルダン法

目標は掃き出し法によって，$\boldsymbol{x} = A^{-1}\boldsymbol{b}$ の形を導くことである．これを以下の手順で解く．

1. (1.2) 式の 1 行目を a_{11} で割る．この演算における a_{11} をピボット (pivot) という．

$$\begin{aligned} x_1 + a'_{12}x_2 + a'_{13}x_3 &= b'_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned} \tag{1.3}$$

ここで，記号「$'$」は演算の影響を 1 回受けていることを表す．

2. 上式の 1 行目を a_{21} 倍して，2 行目との差を取る．同様に，1 行目を a_{31} 倍して，3 行目との差を取る．

$$\begin{aligned} x_1 + a'_{12}x_2 + a'_{13}x_3 &= b'_1 \\ 0 + a'_{22}x_2 + a'_{23}x_3 &= b'_2 \\ 0 + a'_{32}x_2 + a'_{33}x_3 &= b'_3 \end{aligned} \tag{1.4}$$

3. 次に a'_{22} をピボットとして，同様の演算を行う．

$$\begin{aligned} x_1 + 0 \ \ + a''_{13}x_3 &= b''_1 \\ 0 + x_2 + a''_{23}x_3 &= b''_2 \\ 0 + 0 \ \ + a''_{33}x_3 &= b''_3 \end{aligned} \tag{1.5}$$

4. 最後に a''_{33} をピボットとして，同様の演算を行うことで解を得る．

$$\begin{aligned} x_1 + 0 \ \ + 0 \ \ &= b'''_1 \\ 0 + x_2 + 0 \ \ &= b'''_2 \\ 0 + 0 \ \ + x_3 &= b'''_3 \end{aligned} \tag{1.6}$$

[*2] A は正則行列とする．

■ 計算コスト

この方法による計算コストを考えてみる (乗除の計算回数でカウント*3)．N 次元の連立一次方程式では，最初のピボット選択で N^2 回乗除計算を行う．その後のピボット選択では $N \times (N-1), N \times (N-2), \cdots, N \times 1$ と乗除回数が減っていくから

$$\mathcal{O} \sim \sum_{m=1}^{N} Nm = \frac{N^2(N+1)}{2} \sim \frac{N^3}{2} \tag{1.7}$$

となる*4．なお，実際にはピボットが 0 になるのを避けるように行の入れ替え作業 (ピボット選択) を行う必要があるが．このノートで扱うラプラス方程式の離散式 (*The Laplacian Difference Equation*, LDE [Chapra and Canale, 2015]]) は既に対角優位なので詳しくは書かない*5．

1.2 ガウスの消去法

ガウス・ジョルダン法の効率化を図った方法である．以下の手順で解く．

1. a_{11} をピボットとするところまでは同じである．

$$\begin{aligned} x_1 + a'_{12}x_2 + a'_{13}x_3 &= b'_1 \\ 0 + a'_{22}x_2 + a'_{23}x_3 &= b'_2 \\ 0 + a'_{32}x_2 + a'_{33}x_3 &= b'_3 \end{aligned} \tag{1.8}$$

2. 次に a_{22} をピボットとするが，ピボット行よりも下の行を計算対象とする．

$$\begin{aligned} x_1 + a'_{12}x_2 + a'_{13}x_3 &= b'_1 \\ 0 + x_2 \quad + a''_{23}x_3 &= b''_2 \\ 0 + 0 \quad + a''_{33}x_3 &= b''_3 \end{aligned} \tag{1.9}$$

この操作を最下行まで行うことで，係数行列を対角要素が 1 の上三角行列に変形できる．

$$\begin{aligned} x_1 + a'_{12}x_2 + a'_{13}x_3 &= b'_1 \\ 0 + x_2 \quad + a''_{23}x_3 &= b''_2 \\ 0 + 0 \quad + x_3 \quad &= b'''_3 \end{aligned} \tag{1.10}$$

この演算処理を**前進消去**という．この地点で x_3 の階が b'''_3 であることが分かる．

3. これより，下から順に代入を繰り返すことで解を得ることができる．例えば x_i を求める際には

$$x_i = b_i - \sum_{j=i+1}^{n} a_{ij}x_j \tag{1.11}$$

を用いる．ここで，i 行目の方程式をプライムを外した形で書いた．最下行の方程式から順に上の行に向かって解を求める演算処理を**後退代入**という．

*3 コンピュータの四則演算において，掛け算と割り算は足し算と引き算に比べて遅い．
*4 プライム付きの量を数えるが，係数が 0 や 1 になるものは勘定しない．
*5 詳しくは牛島 (2007) を参照されたい．

■ 計算コスト

N 次元の連立一次方程式に関する計算コストを見積もってみる．最初のピボット選択では N^2 のコストを要する．その次からは順に下方向だけを計算するから，$(N-1)\times(N-1), (N-2)\times(N-2), \cdots, 1\times 1$ と乗除回数が減っていく．すなわち

$$\mathcal{O} \sim \sum_{m}^{N-1} m^2 = \frac{1}{6}(N-1)N(2N-1) \sim \frac{N^3}{3} \tag{1.12}$$

となる．後退代入では上三角行列分だけ代入演算が行われるから，$\mathcal{O}(N^2/2)$ 程度である．ゆえに N が大きいと，<u>ガウスの消去法はガウス・ジョルダン法に比べて約 1.5 倍速い</u>と予想される．

1.3　LU 分解

(1.1) 式の係数行列 A を下三角成分 L と上三角成分 U に分解する．

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix} \tag{1.13}$$

これを次のように表す．

$$A = LU \tag{1.14}$$

この時，元の連立一次方程式は

$$A\boldsymbol{x} = \boldsymbol{b} \;\rightarrow\; L\underbrace{U\boldsymbol{x}}_{\boldsymbol{y}} = \boldsymbol{b} \;\rightarrow\; L\boldsymbol{y} = \boldsymbol{b} \tag{1.15}$$

と変形できる．これを利用して，次の手順で解を求める (LU 分解).

1. $A = LU$ となる三角行列を求めておく．
2. $L\boldsymbol{y} = \boldsymbol{b}$ の解 $\boldsymbol{y}$ を求める．ここで，$\boldsymbol{y}$ は以下を満たす．

   $$U\boldsymbol{x} = \boldsymbol{y} \tag{1.16}$$

   L の性質から，上から順に解が求まる (**前進代入**).
3. 最後に上式を解く (**後退代入**).

■ 計算コスト

最初の分解作業は掃き出し法を用いるので，ガウスの消去法と同じ $\mathcal{O}(N^3/3)$ の計算量である．これに加えて，前進代入と後退代入が加わるので，全体としては

$$\mathcal{O} \sim \frac{N^3}{3} + \frac{N^2}{2} \times 2 \tag{1.17}$$

となる．A が一定で，$\boldsymbol{b}$ が変わるような場合では高々 $\mathcal{O}(N^2)$ となるので，ガウスの消去法よりも速くなる．そのため，格子法による非圧縮性流体の圧力ポアソン方程式を解く際などに用いられる．

2　反復解法

ある初期値から出発して，演算の反復により近似解を求める方法である．ここでは，近似解以外のデータを固定する「定常反復解法」を扱う[*6]．具体的には，ヤコビ法 (Jacobi method)，ガウス・ザイデル法 (Gauss-Seidel method)，SOR 法 (Successive Over-Relaxation, 逐次加速緩和法) の概要を書く．

2.1　ヤコビ法

k 回目の反復計算で得られる解ベクトル $\boldsymbol{x}_k$ の i 番目の要素を $x_i^{(k)}$ とする $(1 \leq i \leq n)$．k 回目の演算における積 $A\boldsymbol{x}$ の計算に対して，第 i 行の対角要素に $x_i^{(k)}$，非対角要素に $x_i^{(k-1)}$ を用いる．この時，元の方程式の第 i 行の関係は以下のようになる．

$$\sum_{j=1}^{i-1} a_{ij} x_j^{(k-1)} + a_{ii} x_i^{(k)} + \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} = b_i \tag{2.1}$$

これより

$$x_i^{(k)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k-1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right) \tag{2.2}$$

を得る．得られた $x_i^{(k)}$ を再び右辺に用いて，残差 $\boldsymbol{b} - A\boldsymbol{x}_k$ を十分小さくする収束解 $\boldsymbol{x}_k$ を求める．

2.2　ガウス・ザイデル法

ヤコビ法では k 回目の値を求めるのに，$(k-1)$ 回目の値を使う．これに対して，ガウス・ザイデル法では同じ k 回目の解を用いる．例えば，計算の順番として $j = 1, 2, \cdots, n$ としているなら，$x_i^{(k)}$ を計算する際に，$x_1^{(k)}, x_2^{(k)}, \cdots, x_{i-1}^{(k)}$ は既に得られているため

$$x_i^{(k)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right) \tag{2.3}$$

として反復演算を進める．より未来の値を使っているので収束が速い (らしい)．

2.3　SOR 法

ガウス・ザイデル法に加速パラメーター ω を乗じて，高速化を図る方法である．

$$\begin{aligned} x_i^{(k)} &= x_i^{(k-1)} + \omega \Delta x_i^{(k)} \\ &= x_i^{(k-1)} + \omega \left[\frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right) - k_i^{(k-1)} \right] \end{aligned} \tag{2.4}$$

$\Delta x_i^{(k)} \equiv x_i^{(k)} - x_i^{(k-1)}$ とした．$\omega = 1$ にすると，ガウス・ザイデル法に還元される．通常は $1 < \omega < 2$ の値を用いる．最適な ω の値は後述．

*6 反復の回数によって変化するデータを使う方法は非定常反復解法と呼ばれ，共役勾配法 (CG 法, Conjugate Gradient method) などがある．詳しいアルゴリズムは牛島 (2007) や中島 (2014) などを参考にされたい．

2.4　反復行列と収束条件

係数行列 A を対角行列 D，対角要素を除く下三角行列 E および上三角行列 F を用いて

$$A = D - E - F \tag{2.5}$$

と分解する．この時，ガウス・ザイデル法の式をベクトルで書くと

$$\boldsymbol{x}_k = (D-E)^{-1}(\boldsymbol{b} + F\boldsymbol{x}_{k-1}) \tag{2.6}$$

のような形になる．そこで

$$T = (D-E)^{-1}F, \qquad C = (D-E)^{-1} \tag{2.7}$$

とおくと

$$\boldsymbol{x}_k = T\boldsymbol{x}_{k-1} + C\boldsymbol{b} \tag{2.8}$$

と書ける．定常反復解法では反復式が上式のような形で表現される．T を特に反復行列という．

収束解を $\boldsymbol{x}^*$ とすると，(2.6) 式は

$$\boldsymbol{x}^* = (D-E)^{-1}(\boldsymbol{b} + F\boldsymbol{x}^*) \tag{2.9}$$

となり，整理すると $A\boldsymbol{x}^* = \boldsymbol{b}$ となることが確かめられる．すなわち，収束すれば元の方程式の解となる．

次に収束条件を見る．真の解を $\boldsymbol{X}$ とすると，(2.8) 式は

$$\boldsymbol{X} = T\boldsymbol{X} + C\boldsymbol{b} \tag{2.10}$$

となる．(2.8) 式との差を取ると，誤差 $\boldsymbol{e} = \boldsymbol{x}_k - \boldsymbol{X}$ の関係式を以下のように得る．

$$e_k = Te_{k-1} = T^2 e_{k-2} = \cdots = T^k e_0 \tag{2.11}$$

誤差が 0 になれば収束するので

$$T^k \ \rightarrow \ \mathsf{O} \quad (k \rightarrow \infty) \tag{2.12}$$

となればよい．ここで，O は零行列である．この性質を持つ行列を収束行列と呼ぶ．収束行列であるための必要十分条件は

$$\max |\lambda_i| < 1 \tag{2.13}$$

であることが示されている [森, 2002]．ここで，λ_i は行列 T の固有値である．従って，行列 T の固有値の最大値を調べれば良い．ここで，上式の左辺をスペクトル半径と呼ぶ．さらに，係数行列 A が**狭義の対角優位行列**である場合にはスペクトル半径は 1 より小さいことが分かっている [森, 2002]．すなわち，実用的には A が以下の関係を満たしている時，収束解が得られる．

$$|a_{ii}| > \sum_{j=1, j\neq i}^{N} |a_{ij}| \quad (1 \leq i \leq N) \tag{2.14}$$

これは「係数行列 A の全ての行において，対角成分の絶対値が非対角成分の絶対値の和よりも大きければ収束解が得られる」ことを示す．

■ 最適な加速パラメーター ω_o

SOR 法における加速パラメーター ω の最適な値 ω_o は次式で与えられる[*7]．

$$\omega_o = \frac{2}{1+\sqrt{1-\rho^2(T)}} \tag{2.15}$$

ここで $\rho(T)$ は反復行列 T のスペクトル半径である．特に，2 次元の離散化されたラプラス方程式において，格子間距離が等間隔な場合には，$\rho(T)$ の値が理論的に求められており

$$\omega_o = \frac{2}{1+\sin\left[\pi/(n-1)\right]} \tag{2.16}$$

となる．ここで，n は各方向の格子点数である ($n \leq 3$)．これを図示すると図 2.1 のようになり，n が大きくなると 2 に漸近してゆくことが分かる．$\omega < 1$ の場合はガウス・ザイデル法よりも収束が遅くなるが，より安定になるらしい [山本, 2007].

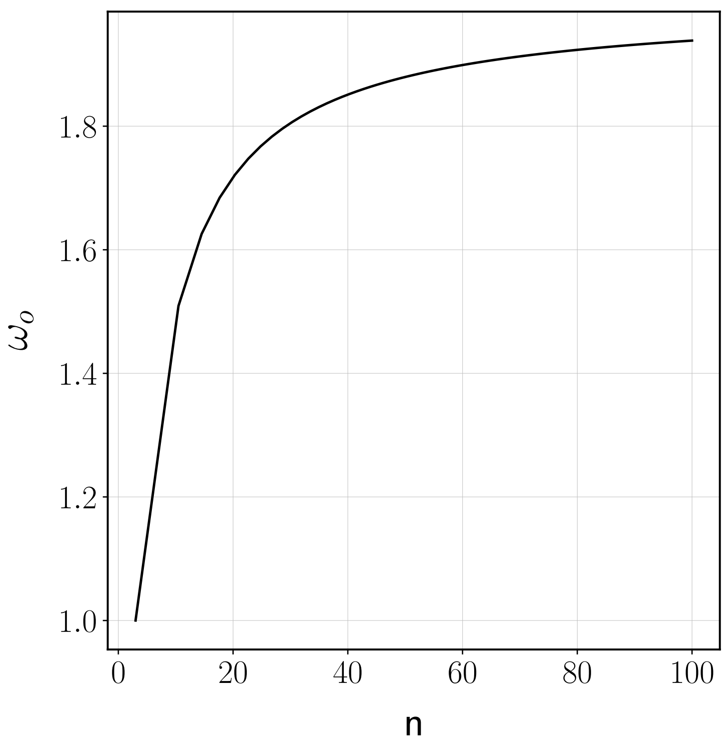

図 2.1: 最適化された SOR パラメーター ω_o．ただし，2 次元の離散化されたラプラス方程式 ($\Delta_x = \Delta_y$) の場合に有効．横軸は一方向の格子点数．$n > 50$ では $\omega_o > 1.9$ となり，n が大きくなるほど 2 に漸近する．

*7 証明は森 (2002) を参照されたい．

3　ラプラス方程式への実装

2 次元のラプラス方程式に上記の数値解法を実装した．使用言語は Python とし，標準ライブラリに加えて，numpy, scipy, matplotlib モジュールがあれば動く．プログラムソースは以下の URL に置いた．ダウンロードはもちろん，二次配布や書き換えも OK である．

https://github.com/shobuzako-kensuke/Material-Storage/tree/main/02_simulation_code

3.1　問題設定

■ 物理背景

ラプラス方程式は流体力学，電磁気学，熱力学などの多くの自然科学分野でお見掛けする楕円型偏微分方程式の一種である[*8]．

$$\nabla^2\phi(\boldsymbol{x},t)=0 \ \Leftrightarrow \ \Delta\phi(\boldsymbol{x},t)=0 \tag{3.1}$$

これから扱う問題は温度に関するラプラス方程式である．これは定常的な熱拡散の問題と見ることもできる．まず，熱拡散方程式は以下である．

$$\frac{\partial T(\boldsymbol{x},t)}{\partial t}=\kappa\nabla^2 T(\boldsymbol{x},t) \tag{3.2}$$

ここで，$T(\boldsymbol{x},t)$ は時刻 t における位置 $\boldsymbol{x}$ での温度で，κ は熱拡散率 (大きいほど熱が速く伝わる) である．定常状態では時間微分が消えるので

$$\nabla^2 T=0 \tag{3.3}$$

となる．以降は 2 次元問題を考えることにすると

$$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}=0 \tag{3.4}$$

を最終的に得る．言うまでもなく時間に依存しない問題であるから，初期条件を必要とせず，境界条件のみで解が一意に決まる．すなわち，境界の温度を適当に与えることで，内部の全ての点の温度が決まることになる．以降，$T(\boldsymbol{x},t)\to T(x,y)$ とする．

*8 2 階の偏微分方程式の分類について [Chapra and Canale, 2015]．次のような $\phi(x,y)$ に関する 2 階の線形微分方程式を考える．

$$a\,\frac{\partial^2\phi}{\partial x^2}+b\frac{\partial^2\phi}{\partial x\partial y}+c\,\frac{\partial^2\phi}{\partial y^2}=F\left(x,y,\phi,\frac{\partial\phi}{\partial x},\frac{\partial\phi}{\partial y}\right)$$

これは以下の 3 つに分類される．

1. 楕円型偏微分方程式： $b^2-4ac<0$ $(a=c=1,b=0)$ (例) ラプラス方程式，ポアソン方程式
2. 放物型偏微分方程式：$b^2-4ac=0$ $(a=-1,b=c=0)$ (例) 拡散方程式
3. 双曲型偏微分方程式：$b^2-4ac>0$ $(a=1,b=0,c=-1)$ (例) 波動方程式

■ 離散化

中心差分で離散化する．まず，任意の点 (x_i, y_j) から Δx だけ離れた点における $T(x_i + \Delta x, y_j)$ の値は，テーラー展開から

$$T(x_i + \Delta x, y_j) = T(x_i, y_j) + \frac{\partial T}{\partial x}(\Delta x) + \frac{1}{2!}\frac{\partial^2 T}{\partial x^2}(\Delta x)^2 + \frac{1}{3!}\frac{\partial^3 T}{\partial x^3}(\Delta x)^3 + \mathcal{O}\left[(\Delta x)^4\right] \tag{3.5}$$

一方，$-\Delta x$ だけ離れた点における $T(x_i - \Delta x, y_j)$ の値は同様に

$$T(x_i - \Delta x, y_j) = T(x_i, y_j) + \frac{\partial T}{\partial x}(-\Delta x) + \frac{1}{2!}\frac{\partial^2 T}{\partial x^2}(-\Delta x)^2 + \frac{1}{3!}\frac{\partial^3 T}{\partial x^3}(-\Delta x)^3 + \mathcal{O}\left[(-\Delta x)^4\right] \tag{3.6}$$

辺々足して整理すると

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{(\Delta x)^2} + \mathcal{O}\left[(\Delta x)^2\right] \tag{3.7}$$

となる．ここで，$T(x_i, y_j)$ を $T_{i,j}$ と表し，$+\Delta x$ だけ離れた点を $i+1$，$-\Delta x$ だけ離れた点を $i-1$ とした．y についても同様に

$$\frac{\partial^2 T}{\partial y^2} = \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{(\Delta y)^2} + \mathcal{O}\left[(\Delta y)^2\right] \tag{3.8}$$

そこで (3.4) 式は $\mathcal{O}\left[(\Delta x)^2\right]$ と $\mathcal{O}\left[(\Delta y)^2\right]$ の範囲で

$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{(\Delta x)^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{(\Delta y)^2} = 0 \tag{3.9}$$

という離散化された式に書き換えられる．今 $\Delta x = \Delta y$ とすると

$$T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1} - 4T_{i,j} = 0 \tag{3.10}$$

を最終的に得る．すなわち，$T_{i,j}$ の温度は周りの 4 点の平均的な温度で与えられる (図 3.1)．

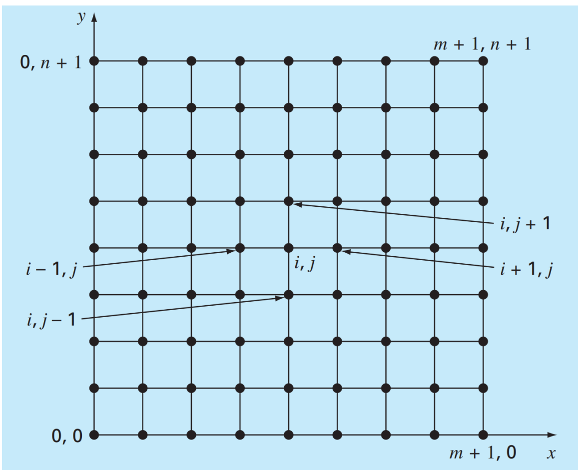

図 3.1: ラプラス方程式における離散点の関係．図の出典は Chapra and Canale (2015)．

例として，$3 \times 3 = 9$ ($N = 9$) の問題を考え，境界条件を図 3.2 のように設定する．

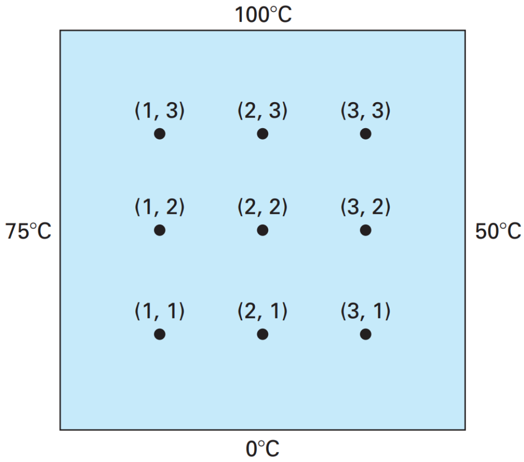

図 3.2: 離散化の例題．図の出典は Chapra and Canale (2015)．

まず，点 $(1, 1)$ に関する (3.10) 式は

$$T_{21} + T_{01} + T_{12} + T_{10} - 4T_{11} = 0 \tag{3.11}$$

となるが，境界条件より $T_{01} = 75, T_{10} = 0$ なので

$$-4T_{11} + T_{12} + T_{21} = -75 \tag{3.12}$$

となる．知りたい変数は 9 つあるので，各計算点に対して上式と同様の関係式を求めて整理すると

$$\begin{pmatrix} -4T_{11} & T_{21} & & T_{12} & & & & & \\ T_{11} & -4T_{21} & T_{31} & & T_{22} & & & & \\ & T_{21} & -4T_{31} & & & T_{32} & & & \\ T_{11} & & & -4T_{12} & T_{22} & & T_{13} & & \\ & T_{21} & & T_{12} & -4T_{22} & T_{32} & & T_{23} & \\ & & T_{31} & & T_{22} & -4T_{32} & & & T_{33} \\ & & & T_{12} & & & -4T_{13} & T_{23} & \\ & & & & T_{22} & & T_{13} & -4T_{23} & T_{33} \\ & & & & & T_{32} & & T_{23} & -4T_{33} \end{pmatrix} = \begin{pmatrix} -75 \\ 0 \\ -50 \\ -75 \\ 0 \\ -50 \\ -175 \\ -100 \\ -150 \end{pmatrix} \tag{3.13}$$

を得る．左辺を変形すると

$$\begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1- & 4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix} \begin{pmatrix} T_{11} \\ T_{21} \\ T_{31} \\ T_{12} \\ T_{22} \\ T_{32} \\ T_{13} \\ T_{23} \\ T_{33} \end{pmatrix} = \begin{pmatrix} -75 \\ 0 \\ -50 \\ -75 \\ 0 \\ -50 \\ -175 \\ -100 \\ -150 \end{pmatrix} \tag{3.14}$$

という形にできる．これは $A\boldsymbol{x} = \boldsymbol{b}$ という 9 次元の一次方程式となる．係数行列は $N \times N$ と巨大であるが，そのほとんどの要素が 0 である．このような行列は疎行列 (sparse matrix) と呼ばれ，アルゴリズムを工夫することでメモリを節約できる．これは対角優位なので，反復法によって収束解が得られる (正しいかは別問題)．

■ 計算条件

計算条件は以下のようにした．

- 2 次元の正方形領域 $(x, y) = ([0, L], [0, L])$ を解く
- 各辺の温度を固定温度とする (ディリクレ条件) ⋯ $T_{\rm top}, T_{\rm bottom}, T_{\rm right}, T_{\rm left}$
- 数値解法として以下を実装
  1. Gauss-Jordan method (GJ)
  2. Gaussian elimination method (GE)
  3. LU decomposition (LU)
  4. Jacobi method (Ji=Jacobi iteration)
  5. Gauss-Seidel method (GS)
  6. SOR method (SOR)
  7. np.linalg.solve (NP) ⋯ Python の一次方程式を解くソルバー*9

  ただし，LU 分解は Python のソルバー lu_solve(lu_factor(A), b) を用いた．

■ 調べたいこと

(1)　反復法における収束条件の基準値はどれくらいが良いのか？

(2)　最も速いソルバー (数値解析法) は何か？

3.2　プログラムの使い方

■ プログラムの概要

zip ファイルを解凍すると，以下の python ファイルとフォルダが出てくるはずである．

data フォルダ	⋯	「異なる N に対する CPU time の比較」に使うデータの保管場所
fig フォルダ	⋯	図が生成される場所
LDE.py	⋯	メインプログラム
plot_different_N.py	⋯	「異なる N に対する CPU time の比較」を行うプログラム

■ LDE.py の使い方

まず LDE.py を開くと，始めの方にパラメーター入力の部分がある (図 3.3)．基本的な説明はコメントアウトを参照されたい．プログラム内の重要な変数 (引数) は表 3.1 の通りである．各スイッチを起動すると，次のプログラムが追加で動き出す (※ 0 で起動)．

- plot_fig：図を生成
- save_data：「異なる N に対する CPU time の比較」に使うデータを data フォルダに保存
- iteration：反復回数と残差 (後述) の関係を計算 (plot_fig と併用すること)
- ana_sol：解析解を表示 (非推奨)

*9 [使い方] 一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ に対して，np.linalg.solve(A, b) とするだけ．正則行列に対して有効．LU 分解と同じアルゴリズム？ (以下の URL を参照)
https://runebook.dev/ja/docs/numpy/reference/generated/numpy.linalg.solve

解析解は上側温度 T_t 以外の温度が 0 の場合に有効であるが，無限級数の形で書かれる上に，この級数の収束が悪いため計算結果との定量的な比較には用いていない．詳しくは付録 A「解析解の導出」を参照されたい．

表 3.1: 変数の説明．1 辺の格子点数 n は num_grid と同義である．

変数名	次元	各次元の説明	変数の説明
num_grid	–	–	1 辺の格子点数 n
stop_cri	–	–	収束判定のしきい値
T_t, T_b, T_r, T_l	–	–	境界の固定温度
A_ij	(n^2, n^2)	(式番号, 格子点の係数)	$A\boldsymbol{x} = \boldsymbol{b}$ の A
x_ij	$(n^2, 4)$	(格子点番号, $(x, y, T^{(k-1)}, T^{(k)})$)	$A\boldsymbol{x} = \boldsymbol{b}$ の $\boldsymbol{x}$
b_ij	$(n^2, 1)$	–	$A\boldsymbol{x} = \boldsymbol{b}$ の $\boldsymbol{b}$

```
#==================#
# INPUT
#==================#

# Tex user:0, NOT Tex user:1
Tex_user = 0
# file name
file_name = 'LDE_01'
# the number of grid points along an edge
num_grid = 20
# stopping criterion in iteration
stop_cri = 0.01
# boundary condition (unit:C)
T_t = 100.0
T_b = 10.0
T_r = 50.0
T_l = 75.0
# switch (0:exe)
plot_fig  = 0 # make figures
save_data = 0 # make dat files for 'plot_different_N.py'
iteration = 0 # execute iteration check program
ana_sol   = 1 # find analytical solution for a specific case
# length (unit:m)
length = 1
```

図 3.3: LDE.py のパラメーター入力画面．

ターミナルからプログラムを動かすと，各ソルバーの経過時間などが出力される (3.4)．

```python
from matplotlib import rc

#===================#
# INPUT
#===================#

# Tex user:0, NOT Tex user:1
Tex_user = 0
# file name
file_name = 'LDE_00'
# the number of grid points along an edge
num_grid = 30
# stopping criterion in iteration
stop_cri = 1E-4
# boundary condition (unit:C)
T_t = 100.0
T_b = 0.0
T_r = 50.0
T_l = 75.0
# switch (0:exe)
plot_fig  = 0 # make figures
save_data = 1 # make dat files for 'plot_different_N.py'
iteration = 1 # execute iteration check program
ana_sol   = 1 # find analytical solution for a specific case
# length (unit:m)
length = 1

#===================#
```

PROBLEMS　OUTPUT　DEBUG CONSOLE　TERMINAL

```
(base) C:\Users\skent\Desktop\勉強会・ゼミ\合同セミナー\2023年度\前期
[Message 1/7] Guass-Jordan          : 3.26 [s]
[Message 2/7] Guassian elimination : 1.70 [s]
[Message 3/7] LU decomposition     : 0.02 [s]
[Message 4/7] Jacobi               : 177.93 [s]
              Iteration            : 694.0
[Message 5/7] Gauss-Seidel         : 188.81 [s]
              Iteration            : 430.0
[Message 6/7] SOR                  : 31.38 [s]
              SOR parameter        :1.805
              Iteration            : 71.0
[Message 7/7] np.linalg.solve      : 0.03 [s]
[Message] producing figure...
[Message] Program has finished !   : 412.17 [s]
```

図 3.4: LDE.py の使用例．

■ plot_different_N.py の使い方

save_data によって生成された各ソルバーの CPU time を使って，「異なる N に対する CPU time の比較」の図を作成する．単に起動するだけで OK．

3.3　結果と考察

3.3.1　収束判定のしきい値

反復解法における収束判定のしきい値 ε_0 の取り方は様々である．今回は以下を採用した．

(1)　各点の収束を見る方法 [Chapra and Canale, 2015]

$$\max \left| \frac{x_{ij}^{(k)} - x_{ij}^{(k-1)}}{x_{ij}^{(k-1)}} \right| < \varepsilon_0 \tag{3.15}$$

(2)　残差ノルム (2 ノルム) を使う方法 [山本, 2007]

$$\frac{||\boldsymbol{b} - A\boldsymbol{x}^{(k)}||}{||\boldsymbol{b}||} < \varepsilon_0 \tag{3.16}$$

ここで

$$||\boldsymbol{b} - A\boldsymbol{x}^{(k)}|| = \sqrt{\sum_{i=1}^{n^2} \left| b_i - \sum_{j=1}^{n^2} a_{ij} x_j^{(k)} \right|^2} \tag{3.17}$$

$$||\boldsymbol{b}|| = \sqrt{\sum_{i=1}^{n^2} |b_i|^2} \tag{3.18}$$

実装プログラムでは反復法ループにおける if 文の使用を避けるために，やや面倒ではあるがコメントアウト形式で書いてあるので，手動で切り替える必要がある (図 3.5)[*10]．

T_t=100, T_b=0, T_r=50, T_l=75，$n = 30$ とした場合の結果を，表 3.2 と図 3.6 から図 3.10 までに示した (SOR パラメータは (2.16) 式で決定)．この結果から以下が言える．

- ε_0 が緩いと正しい解とは異なるところへ収束してしまう
- Ji, GS では判定法 (2) の方が厳しい条件となる一方で，SOR では (1) の方が厳しくなる
- SOR 法は速い上に，正しい解へ行きやすい

従って，この 3 つの反復法ソルバーの中では SOR 法が最も有効的だと考えられる．

[*10] Python に限らず if 文 (条件分岐文) や for 文 (ループ文) は時間がかかるので，なるべく書きたくない．

表 3.2: 異なる反復終了条件による結果の違い．表中の数字はしきい値 ε_0 に至るまでの反復回数を表す．記号○，△，×は他のソルバーによる解との比較 (見た目判断)．

	判定法 \ ε_0	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
Jacobi	(1)	5(×)	48(×)	238(△)	694(○)	1143(○)
	(2)	8(×)	123(×)	496(○)	944(○)	1391(○)
Gauss-Seidel	(1)	5(×)	42(×)	188(△)	430(○)	654(○)
	(2)	5(×)	63(×)	256(○)	480(○)	704(○)
SOR	(1)	15(×)	55(○)	62(○)	71(○)	87(○)
	(2)	12(×)	33(○)	53(○)	62(○)	71(○)

```
def iteration_SOR(num_grid, A_ij, b_ij, x_ij, stop_cri, SOR_para, ite_max):
    global ite
    ite = 0
    x_ij[:, 2] = (T_b + T_t + T_b + T_r)/4.0 # initial value
    for m in range(int(ite_max)):
        ite += 1
        for me in range(num_grid**2):
            c_ij = 0.0
            d_ij = 0.0
            for you_l in range(me):
                c_ij += A_ij[me, you_l] * x_ij[you_l, 3]
            for you_r in range(me+1, num_grid**2):
                d_ij += A_ij[me, you_r] * x_ij[you_r, 2]
            x_ij[me, 3] = x_ij[me, 2] \
                        + SOR_para * ((b_ij[me] - c_ij - d_ij) / A_ij[me, me] - x_ij[me, 2])
        #----- stopping criterion -----#
        #1. each grid
        #-----------------------------#
        err = abs((x_ij[:, 3] - x_ij[:, 2]) / x_ij[:, 2])
        if np.all(err < stop_cri):
            break
        else:
            x_ij[:, 2] = x_ij[:, 3]
            x_ij[:, 3] = 0.0
        #-----------------------------#
        #2. Residual
        #-----------------------------#
        # matrix_Ax = np.dot(A_ij, x_ij[:, 3])
        # tmp_sum_bAx = 0.0
        # tmp_sum_b   = 0.0
        # for i in range(num_grid**2):
        #     tmp_sum_bAx += abs(b_ij[i] - matrix_Ax[i])**2
        #     tmp_sum_b   += abs(b_ij[i])**2
        # residual = math.sqrt(tmp_sum_bAx) / math.sqrt(tmp_sum_b)
        # if (residual < stop_cri):
        #     break
        # else:
        #     x_ij[:, 2] = x_ij[:, 3]
        #     x_ij[:, 3] = 0.0

```

図 3.5: 2 種類の反復法の判定式．この図は SOR 法の場合であるが，ヤコビ法，ガウス・ザイデル法でも同様．stopping criterion の#1. each grid と#2. Residual を手動で切り替えること．

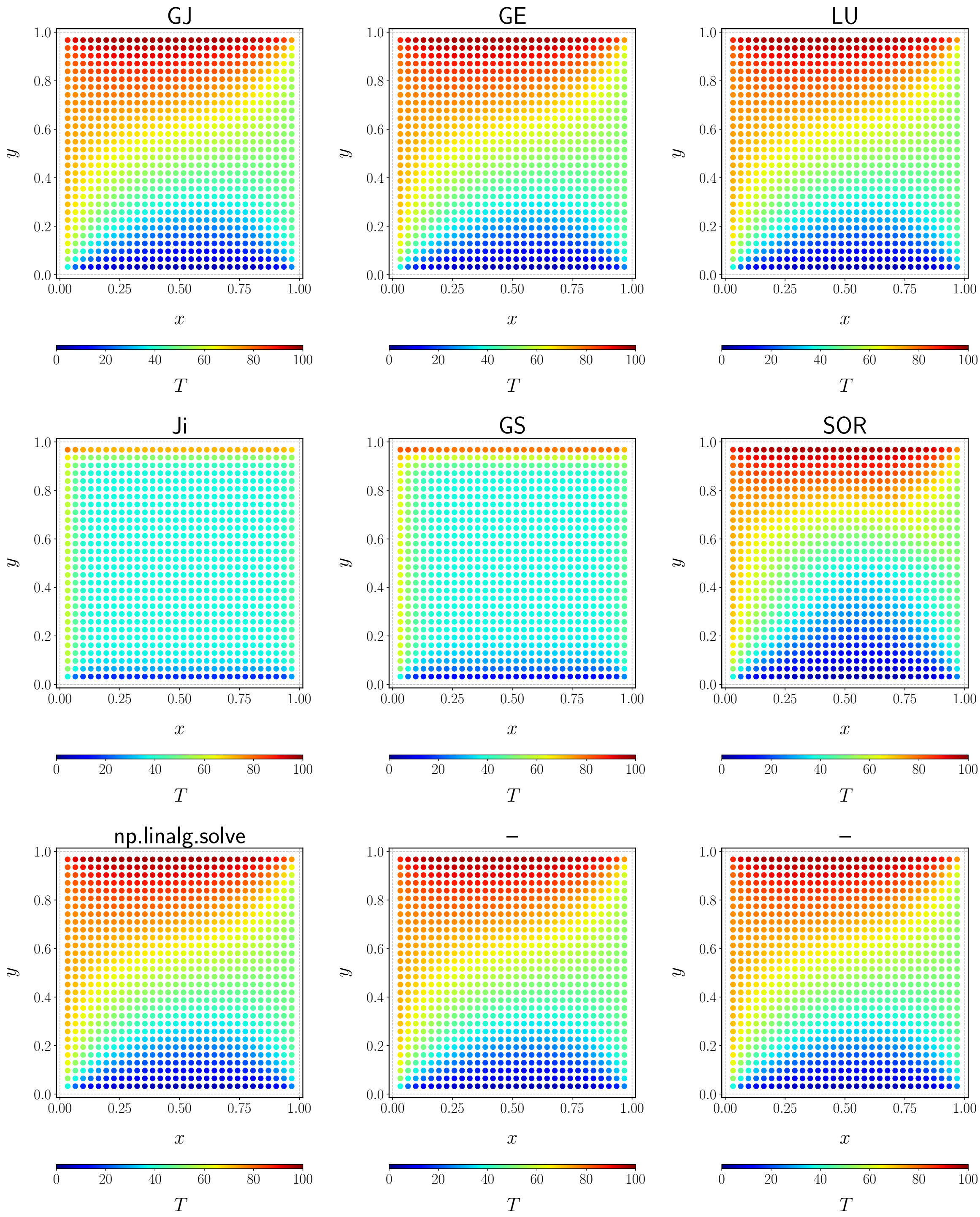

図 3.6: 反復終了条件が (3.15) 式と $\varepsilon_0 = 10^{-1}$ の場合の結果．GJ=Gauss-Jordan, GE=Gaussian elimination, LU=LU decomposition, Ji=Jacobi iteration, GS=Gauss-Seidel, SOR=SOR, np.linalg.solve=python の連立 1 次方程式を解くソルバー．Ji と GS は明らかに異なる解に収束している．

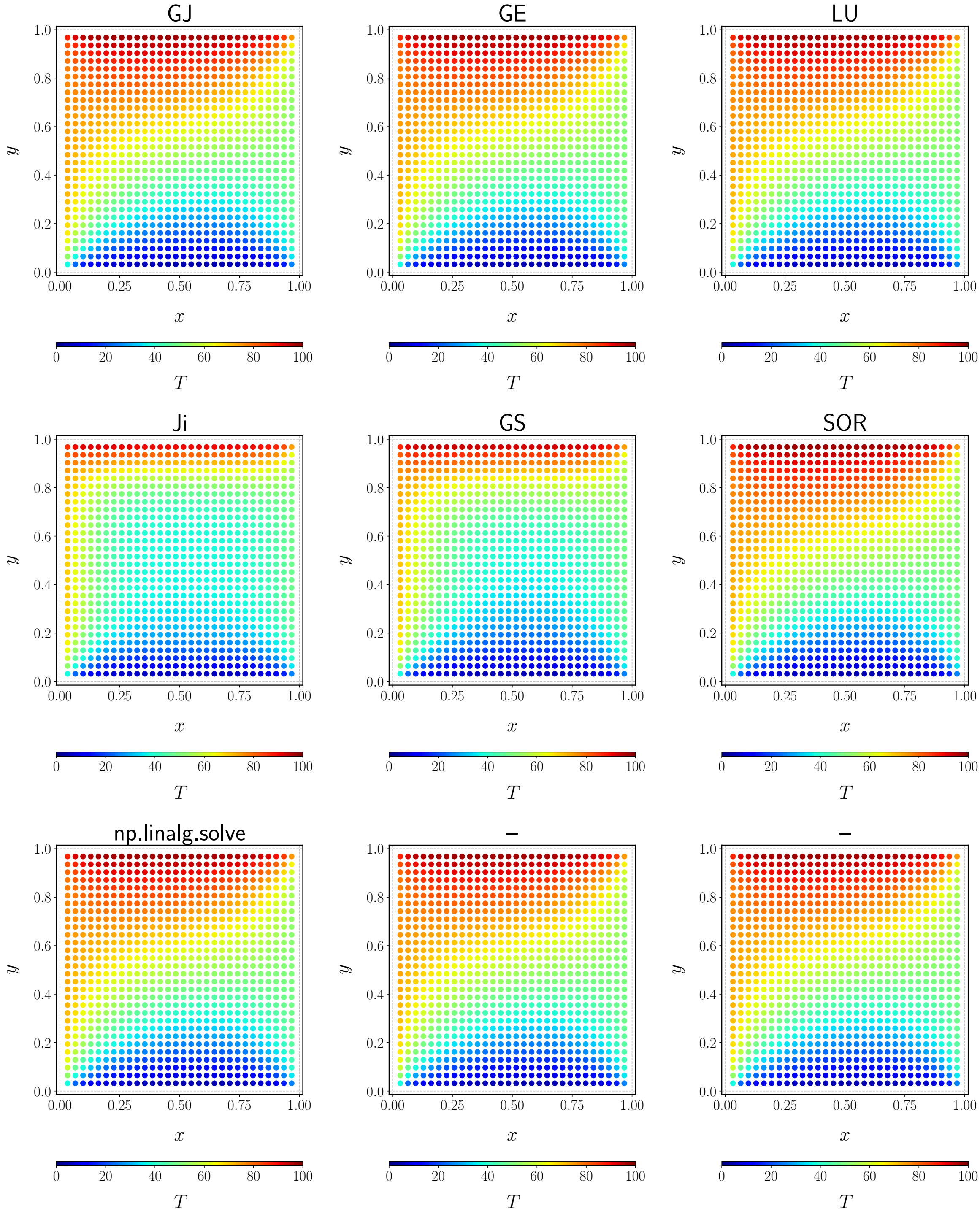

図 3.7: 反復終了条件が (3.15) 式と $\varepsilon_0 = 10^{-2}$ の場合の結果．GJ=Gauss-Jordan, GE=Gaussian elimination, LU=LU decomposition, Ji=Jacobi iteration, GS=Gauss-Seidel, SOR=SOR, np.linalg.solve=python の連立 1 次方程式を解くソルバー．SOR は既に正解にたどり着いてそうである．

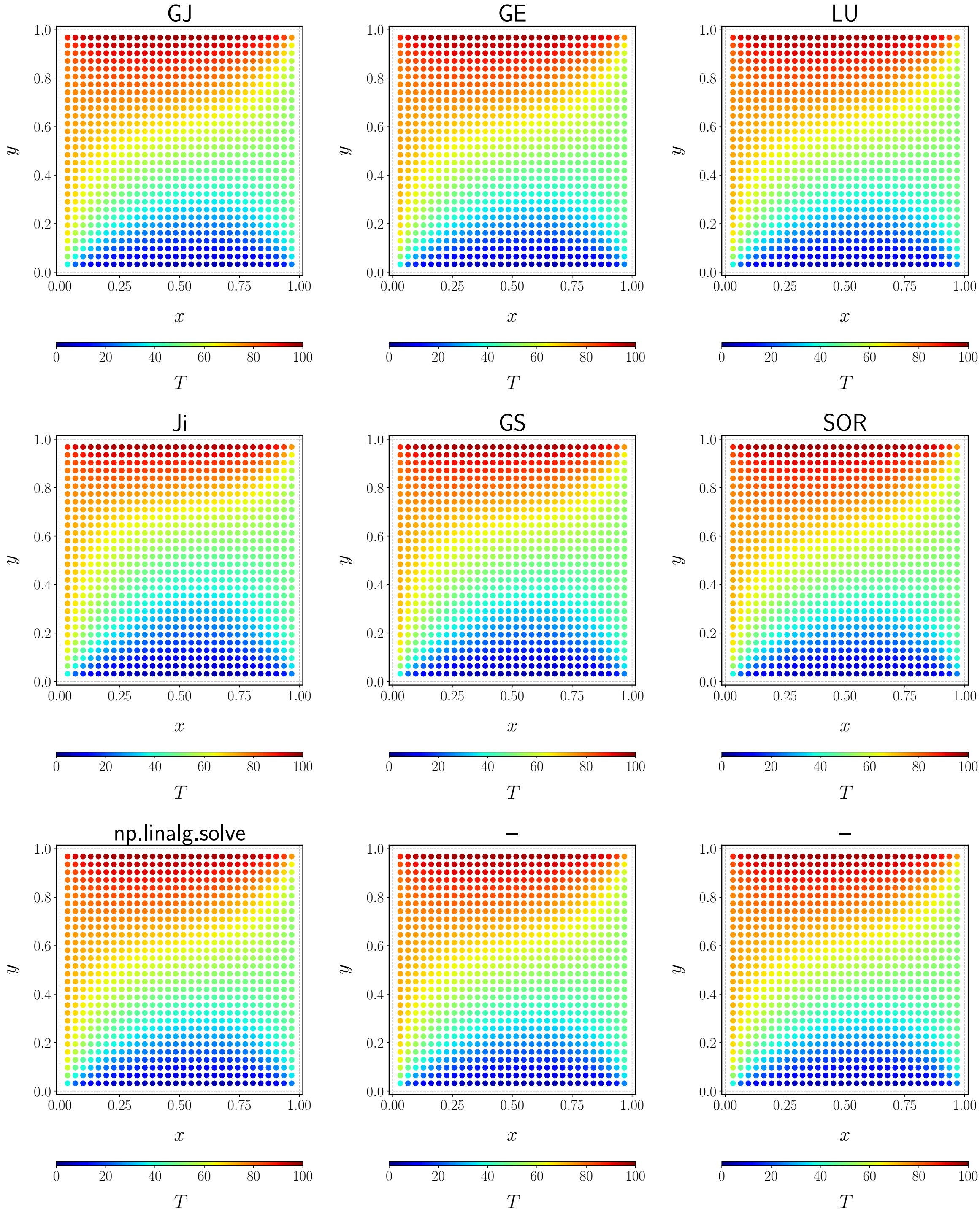

図 3.8: 反復終了条件が (3.15) 式と $\varepsilon_0 = 10^{-3}$ の場合の結果．GJ=Gauss-Jordan, GE=Gaussian elimination, LU=LU decomposition, Ji=Jacobi iteration, GS=Gauss-Seidel, SOR=SOR, np.linalg.solve=python の連立 1 次方程式を解くソルバー．ようやく Ji と GS が正解に近づいてきた．

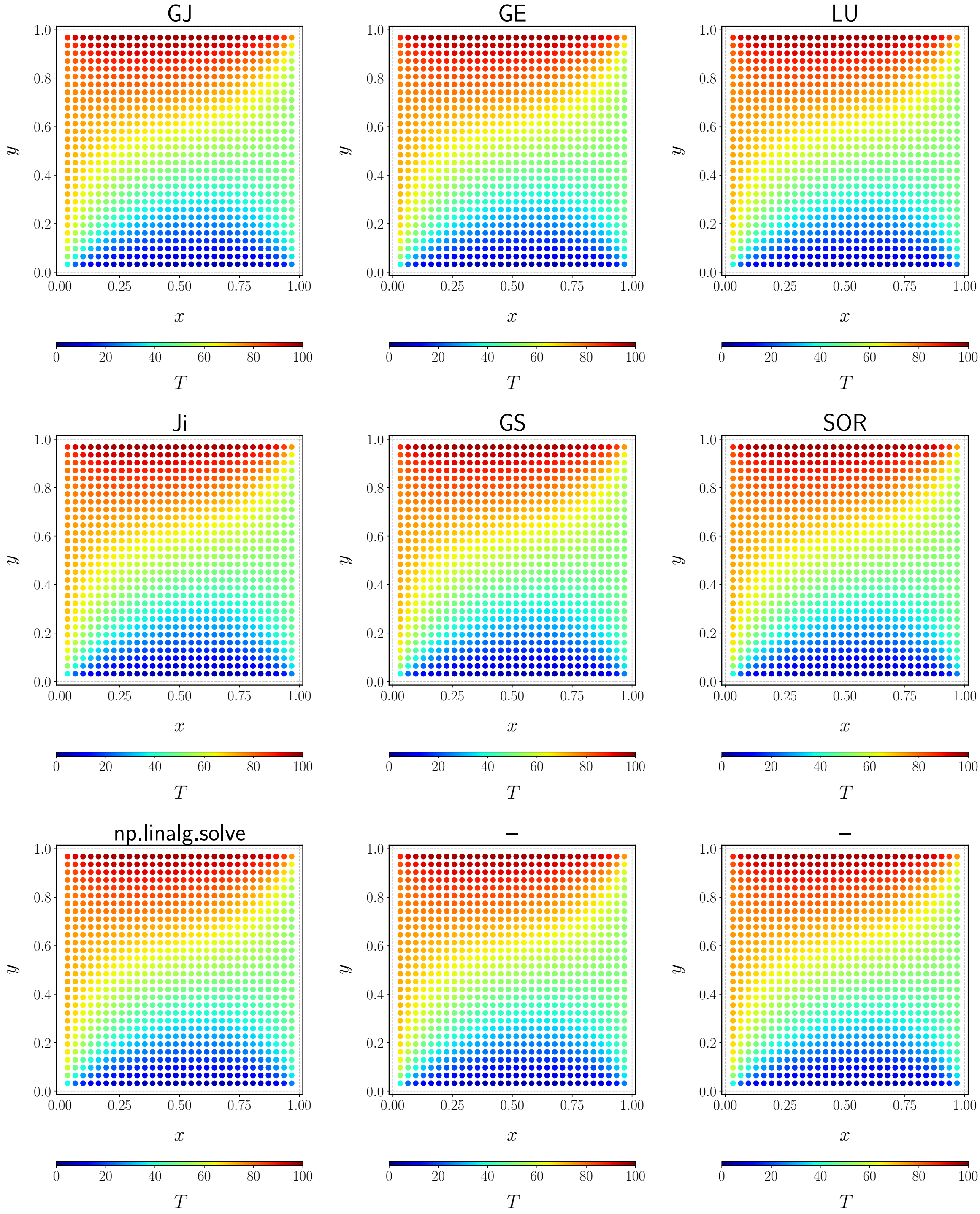

図 3.9: 反復終了条件が (3.15) 式と $\varepsilon_0 = 10^{-4}$ の場合の結果．GJ=Gauss-Jordan, GE=Gaussian elimination, LU=LU decomposition, Ji=Jacobi iteration, GS=Gauss-Seidel, SOR=SOR, np.linalg.solve=python の連立 1 次方程式を解くソルバー．全てのソルバーで同じような解となった．

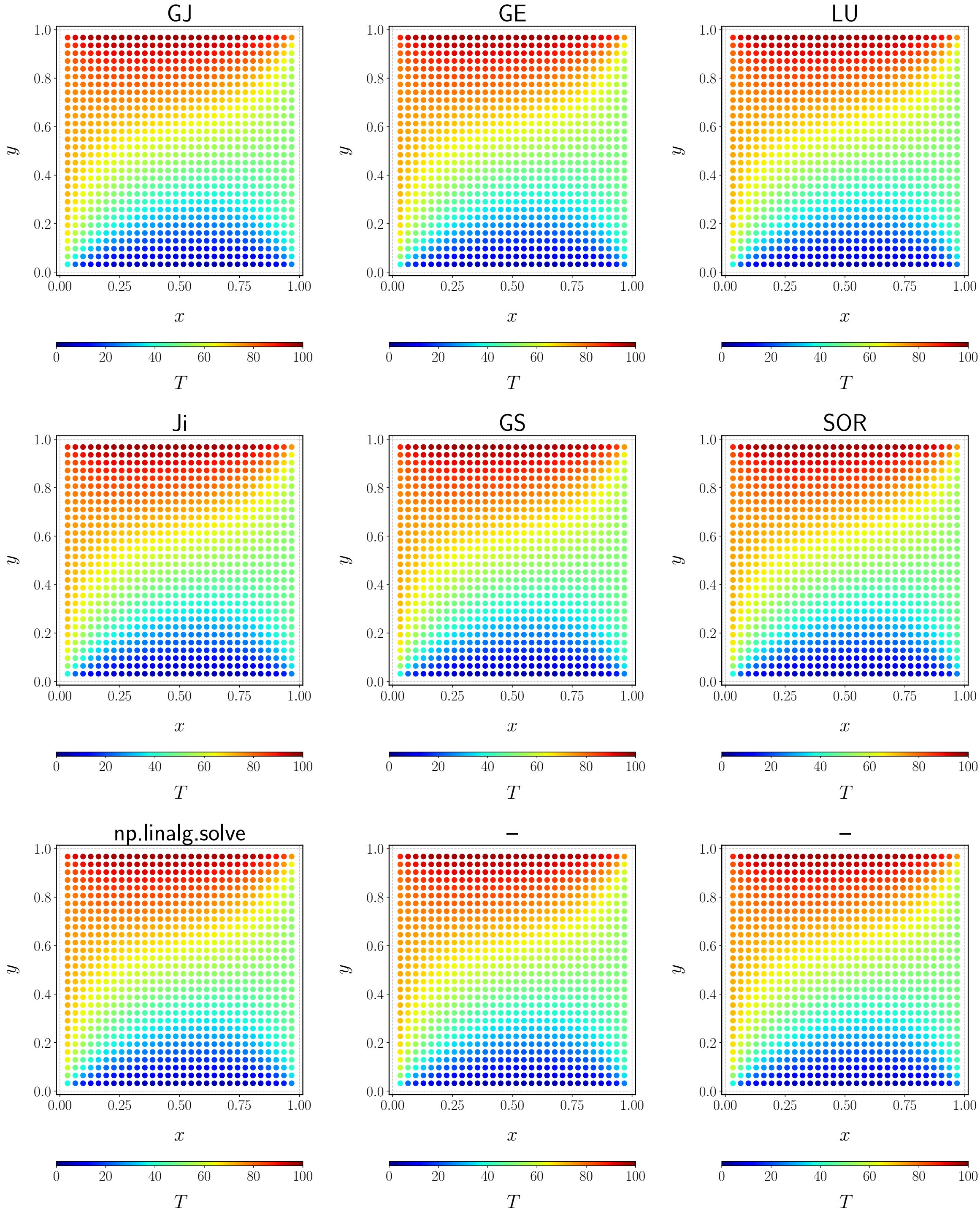

図 3.10: 反復終了条件が (3.15) 式と $\varepsilon_0 = 10^{-5}$ の場合の結果．GJ=Gauss-Jordan, GE=Gaussian elimination，LU=LU decomposition，Ji=Jacobi iteration，GS=Gauss-Seidel，SOR=SOR, np.linalg.solve=python の連立 1 次方程式を解くソルバー．全てのソルバーで同じような解となった．

3.3.2　収束速度について

反復を重ねるごとに解は収束してゆくが，実用的には収束速度が重要である．そこで，先ほどの実験と同じ境界条件で各反復ステップ k に対して正解との残差を取ってみた．ここで，正解を Gauss-Jordan 法によって得られた数値解とし，以下の rms を残差と定義した．

$$R_{\mathrm{rms}} = \sqrt{\frac{1}{n^2}\sum_{i}^{n^2}\left(\frac{T_i^{(k)} - T_{i(\mathrm{GJ})}}{T_{i(\mathrm{GJ})}}\right)^2} \tag{3.19}$$

ここで，$T_i^{(k)}$ は k ステップ目における格子点番号 i の温度である．結果を図 3.11 と図 3.12 に示した．これらの結果から以下が言える．

- ヤコビ法やガウス・ザイデル法に比べ，SOR 法の収束はかなり速い
- SOR パラメータを 2 よりも大きくすると解は収束せず，むしろ発散してしまう
- 格子点数が増えると，収束させるまでにたくさんループを踏まないといけない

3.3.3　計算コストの比較

次の条件で計算コストを比較した．

1. 境界条件として T_t=100, T_b=0, T_r=50, T_l=75 と T_t=100, T_b=0, T_r=0, T_l=0 を選択
2. 反復終了条件は (3.15) 式および $\varepsilon = 10^{-4}$ とする
3. $n = 5, 10, 15, 20, 30$ で比較
4. 使用マシンは Intel(R) Core(TIM) i7-9700 CPU @ 3.00GHz

結果は図 3.13 と図 3.14 のようになった．この結果から以下が言える．

- Python のソルバーがとても速い (もしくは自作プログラムがとてもノロい)
- 全格子点数 N が増加すると，CPU time は増加する
- Python のソルバーの増加率はやや寝ているので優秀
- 一方，自作プログラムの GJ や GE では格子点数が 2 倍増えると，CPU time が大体 1 桁大きくなっているので，理論から推測される $\mathcal{O}(N^3)$ の関係が成り立っていそうである

最後に

実装を通して色々勉強になった．結論は「高速ライブラリを積極的にを使おう」である．

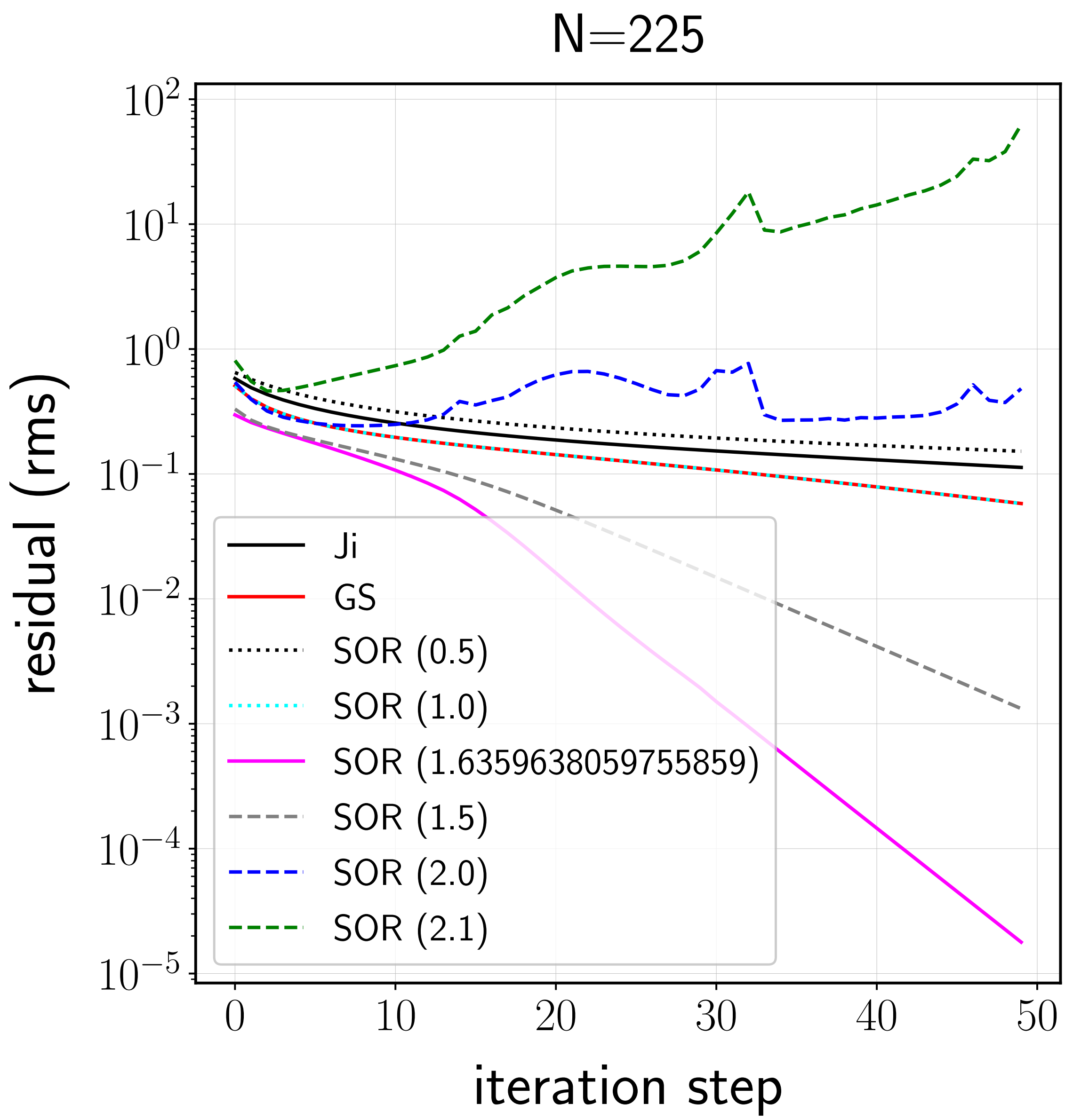

図 3.11: $n = 15$ の場合の収束速度に関する結果．横軸はステップ k を表し，縦軸は残差 (3.19) 式を表す．Ji=Jacobi iteration, GS=Gauss-Seidel である．SOR の括弧の値は SOR パラメータの値であり，シアン色の SOR(1.0) は GS と同じとなる．マゼンタ色は最適化されたパラメータ値である．傾きの絶対値が大きいほど収束速度が速いので，SOR>GS>Ji の順に高速であることを示唆する．

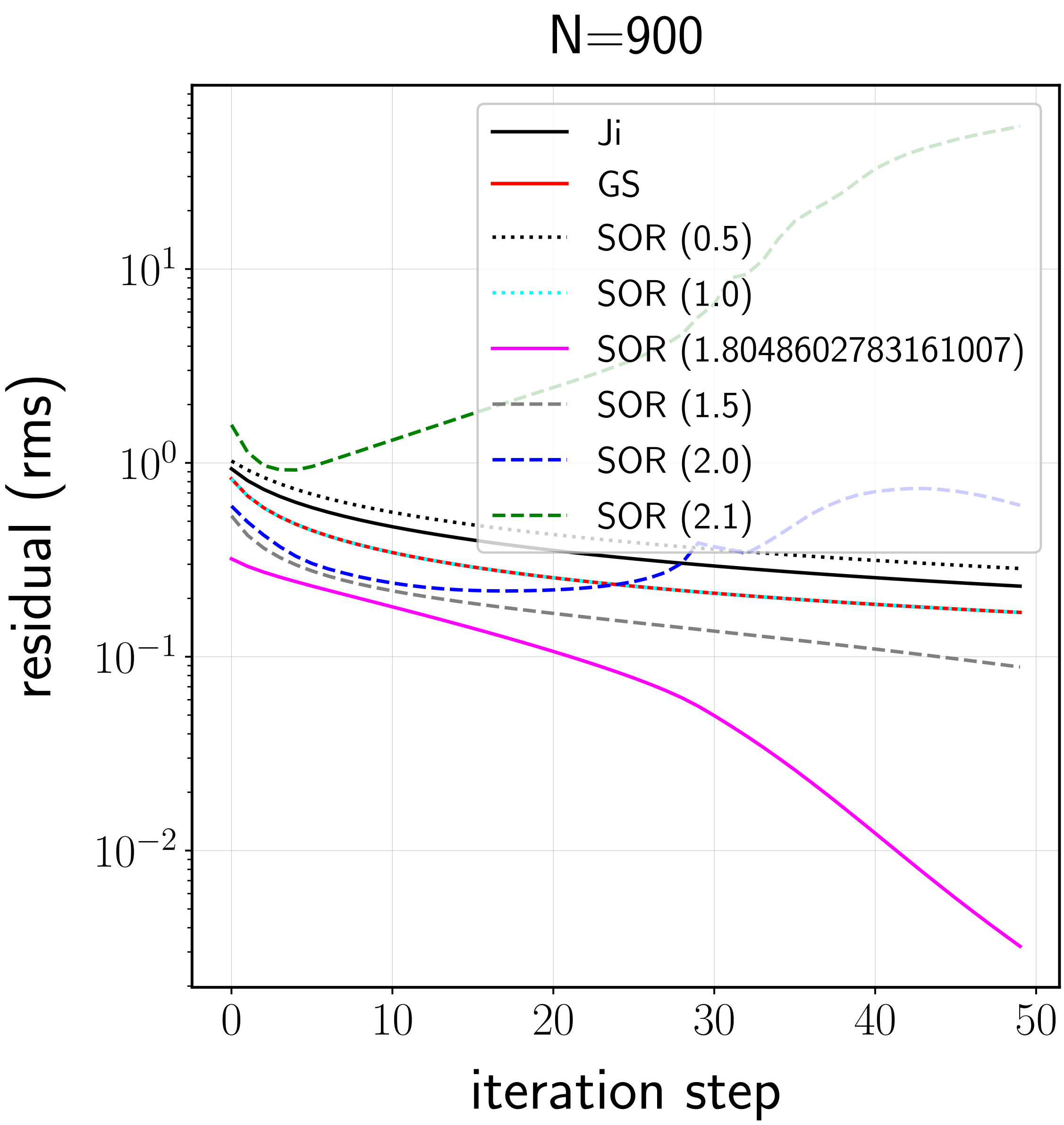

図 3.12: $n=30$ の場合の収束速度に関する結果．横軸はステップ k を表し，縦軸は残差 (3.19) 式を表す．Ji=Jacobi iteration, GS=Gauss-Seidel である．SOR の括弧の値は SOR パラメータの値であり，シアン色の SOR(1.0) は GS と同じとなる．マゼンタ色は最適化されたパラメータ値である．傾きの絶対値が大きいほど収束速度が速いので，SOR>GS>Ji の順に高速であることを示唆する．また図 3.11 よりも n が大きいため，SOR パラメータの値はより 2 に漸近していることも確認できる (図 2.1 を参照).

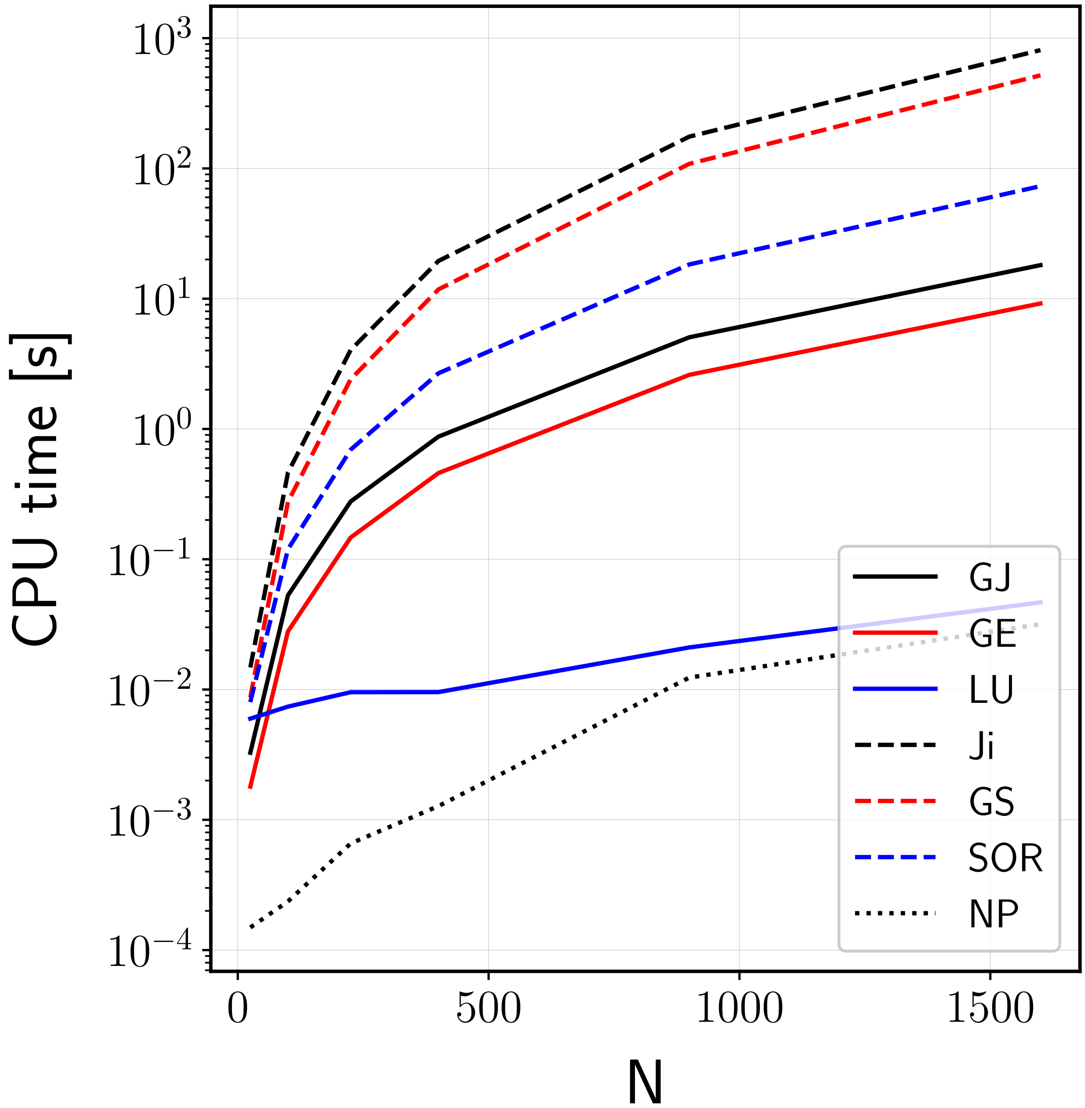

図 3.13: T_t=100, T_b=0, T_r=50, T_l=75 とした場合の計算コストの比較．GJ=Gauss-Jordan, GE=Gaussian elimination, LU=LU decomposition, Ji=Jacobi iteration, GS=Gauss-Seidel, SOR=SOR, np.linalg.solve=python の連立 1 次方程式を解くソルバー．Python のライブラリがダントツで速い．特に $N(=n^2)$ が大きくなるほど顕著になる．[この図の作り方] LDE.py の input の部分で，save_data の値を 0 として，ある $N(=n^2=\text{num\_grid})$ での CPU time を保存する．異なる複数の N に対して同様の計算を行う．plot_different_N.py を実行する．

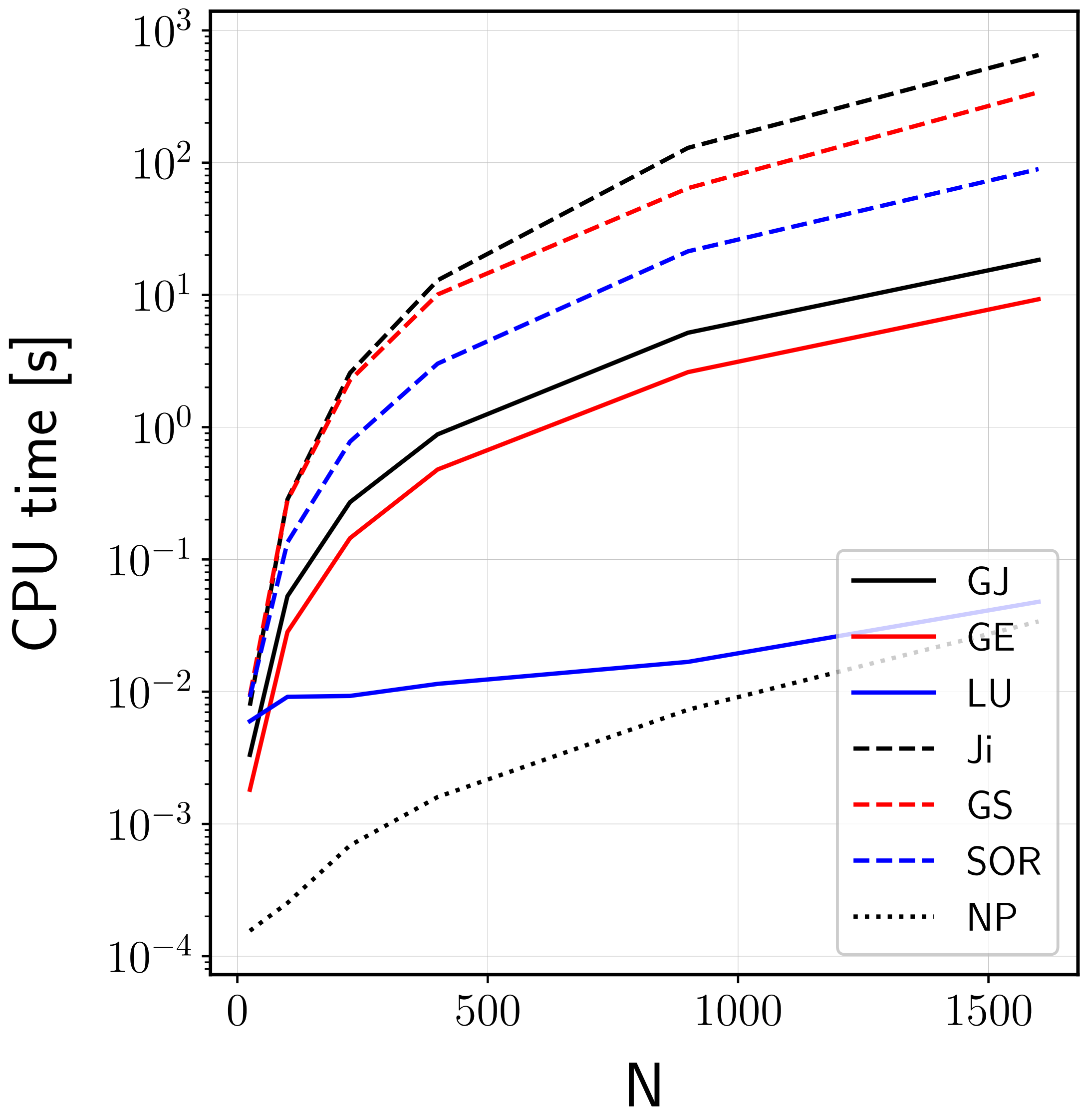

図 3.14: T_t=100, T_b=0, T_r=0, T_l=0 とした場合の計算コストの比較．GJ=Gauss-Jordan, GE=Gaussian elimination, LU=LU decomposition, Ji=Jacobi iteration, GS=Gauss-Seidel, SOR=SOR, np.linalg.solve=python の連立 1 次方程式を解くソルバー．Python のライブラリがダントツで速い．特に $N(=n^2)$ が大きくなるほど顕著になる．[この図の作り方] LDE.py の input の部分で，save_data の値を 0 として，ある $N(=n^2=$ num_grid$)$ での CPU time を保存する．異なる複数の N に対して同様の計算を行う．plot_different_N.py を実行する．

謝辞

本ノートは九大地球内部ダイナミクス研究室と地球深部物理研究室が主催する合同セミナーの資料として作成しました．指導教員である吉田 茂生准教授 (@本学理学研究院) には付録 A の式変形に関するアドバイスを頂きました．中島 涼輔さん (@本学理学研究院) にはプログラム実装に関するアドバイスを頂きました．また本ノートの作成にあたっては，川田 佳史さん (@JAMSTEC) の LaTeX テンプレートを使用させて頂きました．深く感謝申し上げます．

参考文献

■ 本

[1] Chapra, S. C., Canale, R. P. (2015) Numerical methods for engineers, McGraw-Hill Education, 7th edition

[2] Ferziger, J. H., Perić, M. 著 小林 敏雄, 大島 伸行, 坪倉 誠 訳 (2012) コンピュータによる流体力学, 丸善出版

[3] Lynch, D. R. (2005) Numerical Partial Differential Equations for Environmental Scientists and Engineers, Kluwer Academic Publishers

[4] 牛島 省 (2007) 数値計算のための Fortran90/95 プログラミング入門, 森北出版

[5] 薩摩 順吉, 長岡 洋介, 原 康夫 (1995) 物理の数学, 岩波書店

[6] 村上 正康, 佐藤 恒雄, 野沢 宗平, 稲葉 尚志 (2016) 教養の線形代数, 培風館, 6 訂版

[7] 森 正武 (2002) 数値解析, 共立出版

■ ネット上の資料

[8] 中島 研吾 (2014) 線形方程式の解法：反復法 (最終アクセス：2023/4/27)
http://nkl.cc.u-tokyo.ac.jp/13n/SolverIterative.pdf

[9] 山本 昌志 (2007) SOR 法 (最終アクセス：2023/4/27)
http://www.yamamo10.jp/yamamoto/lecture/2007/5E_comp_app/LinearEquations/relaxation_html/node6.html

[10] 渡辺 善隆 (1999) いつ反復計算をやめるべきか？ ～収束判定基準の設定方法～, 九州大学大型計算機センター広報, **32**, 1, 11–30 (最終アクセス：2023/4/28)
https://ri2t.kyushu-u.ac.jp/~watanabe/RESERCH/MANUSCRIPT/KOHO/CONVERGE/intro.html

付録 A　解析解の導出

解くべき式は (3.4) 式であった．

$$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}=0 \tag{3.4}$$

厳密解が求まるように，境界条件を次のように設定する．

$$\begin{aligned}&T(x,0)=0\\&T(x,L)=T_t\\&T(0,y)=0\\&T(L,y)=0\end{aligned} \tag{A.1}$$

つまり，上側境界だけ温度を与えて，それ以外の境界では 0 とする．この条件のもと，変数分離法で解くと

$$T(x,y)=2\sum_{n=1}^{\infty}C_n\sin\left(\frac{n\pi}{L}\,x\right)\sinh\left(\frac{n\pi}{L}\,y\right) \tag{A.2}$$

という級数の形で解を得る．係数 C_n を決めるために上側境界条件を用いる．まず

$$D_n=2C_n\sinh\left(\frac{n\pi}{L}\,y\right) \tag{A.3}$$

と置いて，$y=L$ とすると

$$T_t=\sum_{n=1}^{\infty}D_n\sin\left(\frac{n\pi}{L}\,x\right) \tag{A.4}$$

となる．これはフーリエ級数の形なので

$$D_n=\frac{2}{L}\int_0^L T_t\sin\left(\frac{n\pi}{L}\,x\right)dx \tag{A.5}$$

で D_n を求めることができる．これを解くことで

$$D_n=\begin{cases}\dfrac{4T_t}{n\pi} & (n:\text{奇数})\\ 0 & (n:\text{偶数})\end{cases} \tag{A.6}$$

$n=(2m-1)\ (m=1,2,\cdots)$ と書き直すと

$$C_m=\frac{D_m}{\sinh[(2m-1)\pi]}=\frac{2T_t}{(2m-1)\sinh[(2m-1)\pi]} \tag{A.7}$$

従って

$$T(x,y)=4\sum_{m}^{\infty}\frac{T_t}{(2m-1)\pi}\frac{\sinh\left(\frac{(2m-1)\pi}{L}\,y\right)}{\sinh[(2m-1)\pi]}\ \sin\left(\frac{(2m-1)\pi}{L}\,x\right) \tag{A.8}$$

を最終的に得る．しかし，これをコンピュータで表そうと思ったら次のような問題がある．

- 無限個の足し合わせが必要であるが，m が大きくなると sinh がとても大きくなってオーバーフローを起こす
- ギブス現象が生じるため，端の解の振る舞いが怪しい

実際に計算してみたところ，$m \sim 100$ でオーバーフローとなった．しかし，この級数は sin の部分で m が 1 増えるたびに正負の符号が逆転するので，収束が遅い．従って，$m > 100$ の計算を要する．$m > 100$ では sinh の分母分子の部分を

$$e^{\frac{(2m-1)}{L}(y-L)} \tag{A.9}$$

として計算することにし，$m = 10000$ までの和を計算した．結果は図 A.1 のようになった．

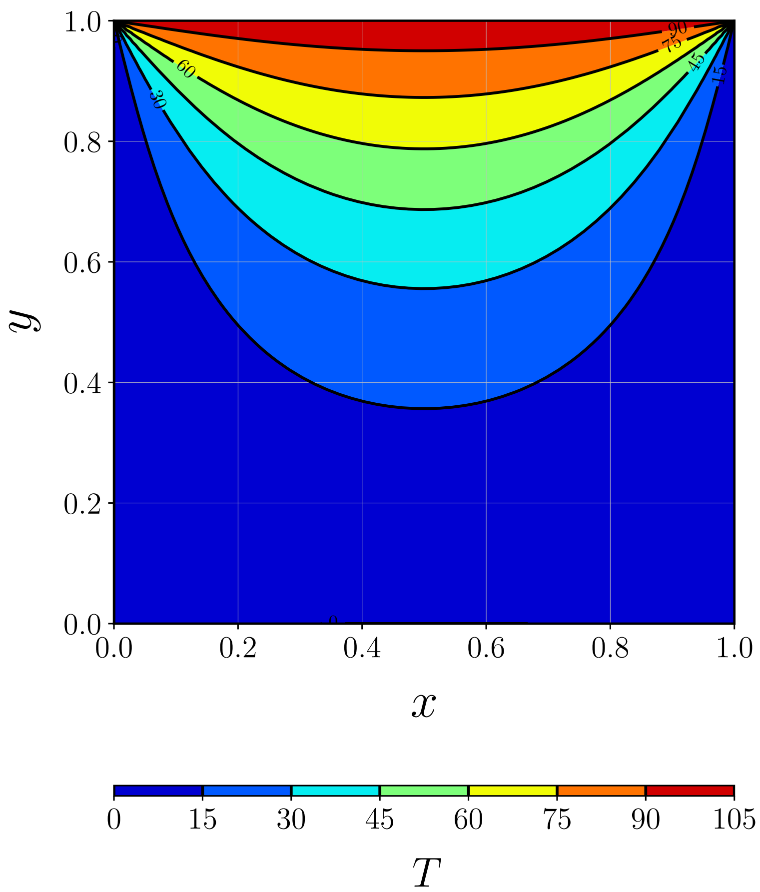

図 A.1: (A.8) 式において $m = 10000$ とした場合の解．カラーコンターが 105 となっているが，本来は 100 が上限である．従って，$m = 10000$ ではまだ収束しておらず，それ以上の足し合わせが必要となるがかなり計算が重くなる．

付録 B　計算プログラム

■ LDE.py

```
#=====================================================================================#
#                Soving Laplacian Difference Equation by numerical methods            #
#                                --  Main program  --                                 #
#-------------------------------------------------------------------------------------#
#                       Copyright by Kensuke Shobuzako (2023)                         #
#=====================================================================================#


#====================#
# charm
#====================#
import numpy as np
import matplotlib.pyplot as plt
import math
import glob
import sys
import os
import time
from scipy import integrate
from scipy.interpolate import interp1d
from scipy.linalg import lu_factor, lu_solve
from matplotlib import rc

#====================#
# INPUT
#====================#

# Tex user:0, NOT Tex user:1
Tex_user = 0
# file name
file_name = 'LDE_00'
# the number of grid points along an edge
num_grid = 20
# stopping criterion in iteration
stop_cri = 1E-4
# boundary condition (unit:C)
T_t = 100.0
T_b = 0.0
T_r = 50.0
T_l = 75.0
# switch (0:exe)
plot_fig  = 0 # make figures
save_data = 1 # make dat files for 'plot_different_N.py'
iteration = 1 # execute iteration check program
ana_sol   = 1 # find analytical solution for a specific case
# length (unit:m)
length = 1

#====================#
# program
#====================#
if (Tex_user == 0):
    rc('text', usetex=True)

program_start_time = time.perf_counter() # get time

#1. building mesh
def build_mesh(length, num_grid):
    global A_ij, x_ij, b_ij, wall, Delta_x
    # set matrix
    all_grid = num_grid**2
    A_ij = np.zeros((all_grid, all_grid)) # coefficient matrix
    x_ij = np.zeros((all_grid, 4))        # unknown variable: (name, (x,y,T_old,T_new))
    b_ij = np.zeros((all_grid, 1))        # right hand of Ax=b
    wall = np.zeros((num_grid+2, 3, 4))   # wall_grid: (name, (x,y,T), (right,left,
        bottom,top))
    # make grids
    Delta_x = float(length / int(num_grid + 1))
    x_ij[0, 0] = Delta_x
    x_ij[0, 1] = Delta_x
    # bottom grids
    for i in range(1, num_grid):
        x_ij[i, 0] = x_ij[i-1, 0] + Delta_x # x component
```

```
        x_ij[i, 1] = Delta_x                # y conponent
    # other grids
    for i in range(num_grid, num_grid**2):
        x_ij[i, 0] = x_ij[i-num_grid, 0]           # x component
        x_ij[i, 1] = x_ij[i-num_grid, 1] + Delta_x # y component
    # # right & left wall grid
    # wall[0, 0, 0] = float(num_grid+1) * Delta_x
    # wall[0, 1, 0] = 0.0
    # wall[0, 0, 1] = 0.0
    # wall[0, 1, 1] = 0.0
    # for i in range(1, len(wall[:,0,0])):
    #     wall[i, 0, 0] = float(num_grid+1) * Delta_x
    #     wall[i, 1, 0] = wall[i-1, 1, 0] + Delta_x
    #     wall[i, 0, 1] = 0.0
    #     wall[i, 1, 1] = wall[i-1, 1, 1] + Delta_x
    # # bottom & top wall grid
    # wall[0, 0, 2] = 0.0
    # wall[0, 1, 2] = 0.0
    # wall[0, 0, 3] = 0.0
    # wall[0, 1, 3] = float(num_grid+1) * Delta_x
    # for i in range(1, len(wall[:,0,0])):
    #     wall[i, 0, 2] = wall[i-1, 0, 2] + Delta_x
    #     wall[i, 1, 2] = 0.0
    #     wall[i, 0, 3] = wall[i-1, 0, 3] + Delta_x
    #     wall[i, 1, 3] = float(num_grid+1) * Delta_x
    # # imposing boundary condition
    # wall[:, 2, 0] = T_r
    # wall[:, 2, 1] = T_l
    # wall[:, 2, 2] = T_b
    # wall[:, 2, 3] = T_t

#2. formulating equation (Ax=b)
def formulate_equation(length, num_grid, Delta_x, A_ij, b_ij, x_ij):
    R_err = Delta_x * 0.5 # Rounding error
    for me in range(num_grid**2):
        you_l = me - 1
        you_r = me + 1
        you_t = me + num_grid
        you_b = me - num_grid
        A_ij[me, me] = -4.0
        # bottom judge
        if 0.0-R_err <= (x_ij[me, 1] - Delta_x) <= 0.0+R_err:
            b_ij[me] += -T_b
        else:
            A_ij[me, you_b] = 1.0
        # top judge
        if length-R_err <= (x_ij[me, 1] + Delta_x) <= length+R_err:
            b_ij[me] += -T_t
        else:
            A_ij[me, you_t] = 1.0
        # right judge
        if length-R_err <= (x_ij[me, 0] + Delta_x) <= length+R_err:
            b_ij[me] += -T_r
        else:
            A_ij[me, you_r] = 1.0
        # left judge
        if 0.0-R_err <= (x_ij[me, 0] - Delta_x) <= 0.0+R_err:
            b_ij[me] += -T_l
        else:
            A_ij[me, you_l] = 1.0

#3-1. Gauss-Jordan method
def Gauss_Jordan(num_grid, A_ij, b_ij, x_ij):
    for me in range(num_grid**2):
        # devided by pibot
        tmp = A_ij[me, me]
        A_ij[me, :] = A_ij[me, :] / tmp
        b_ij[me]    = b_ij[me]    / tmp
        # matrix calculation
        for you in range(num_grid**2):
            if (me != you):
                tmp = A_ij[you, me]
                A_ij[you, :] -= A_ij[me, :] * tmp
                b_ij[you]    -= b_ij[me]    * tmp
    for i in range(num_grid**2):
        x_ij[i, 3] = b_ij[i]

#3-2. Gaussian elimination method
def Gaussian_elimination(num_grid, A_ij, b_ij, x_ij):
    # forward elimination
    for me in range(num_grid**2):
```

```
        tmp = A_ij[me, me]
        A_ij[me, :] = A_ij[me, :] / tmp
        b_ij[me]    = b_ij[me]    / tmp
        for you in range(me+1, num_grid**2):
            tmp = A_ij[you, me]
            A_ij[you, :] -= A_ij[me, :] * tmp
            b_ij[you]    -= b_ij[me]    * tmp
    # backward substitution
    x_ij[-1, 3] = b_ij[-1]
    for me in range(num_grid**2-1, -1, -1):
        tmp = 0.0
        for you in range(me+1, num_grid**2):
            tmp += A_ij[me, you] * x_ij[you, 3]
        x_ij[me, 3] = b_ij[me] - tmp

#3-3. LU decomposition
def LU_decomposition(A_ij, b_ij, x_ij):
    X = lu_solve(lu_factor(A_ij), b_ij)
    for i in range(num_grid**2):
        x_ij[i, 3] = X[i]

#3-4. Jacobi method
def iteration_Jacobi(num_grid, A_ij, b_ij, x_ij, stop_cri, ite_max):
    global ite
    ite = 0
    x_ij[:, 2] = (T_b + T_t + T_b + T_r)/4.0 # initial value
    for m in range(int(ite_max)):
        ite += 1
        for me in range(num_grid**2):
            c_ij = 0.0
            d_ij = 0.0
            for you_l in range(me):
                c_ij += A_ij[me, you_l] * x_ij[you_l, 2]
            for you_r in range(me+1, num_grid**2):
                d_ij += A_ij[me, you_r] * x_ij[you_r, 2]
            x_ij[me, 3] = (b_ij[me] - c_ij - d_ij) / A_ij[me, me]
        #----- stopping criterion -----#
        #1. each grid
        #------------------------------#
        err = abs((x_ij[:, 3] - x_ij[:, 2]) / x_ij[:, 2])
        if np.all(err < stop_cri):
            break
        else:
            x_ij[:, 2] = x_ij[:, 3]
            x_ij[:, 3] = 0.0
        #------------------------------#
        #2. Residual
        #------------------------------#
        # matrix_Ax = np.dot(A_ij, x_ij[:, 3])
        # tmp_sum_bAx = 0.0
        # tmp_sum_b   = 0.0
        # for i in range(num_grid**2):
        #     tmp_sum_bAx += abs(b_ij[i] - matrix_Ax[i])**2
        #     tmp_sum_b   += abs(b_ij[i])**2
        # residual = math.sqrt(tmp_sum_bAx) / math.sqrt(tmp_sum_b)
        # if (residual < stop_cri):
        #     break
        # else:
        #     x_ij[:, 2] = x_ij[:, 3]
        #     x_ij[:, 3] = 0.0

#3-5. Gauss-Seidel method
def iteration_Gauss_Seidel(num_grid, A_ij, b_ij, x_ij, stop_cri, ite_max):
    global ite
    ite = 0
    x_ij[:, 2] = (T_b + T_t + T_b + T_r)/4.0 # initial value
    for m in range(int(ite_max)):
        ite += 1
        for me in range(num_grid**2):
            c_ij = 0.0
            d_ij = 0.0
            for you_l in range(me):
                c_ij += A_ij[me, you_l] * x_ij[you_l, 3]
            for you_r in range(me+1, num_grid**2):
                d_ij += A_ij[me, you_r] * x_ij[you_r, 2]
            x_ij[me, 3] = (b_ij[me] - c_ij - d_ij) / A_ij[me, me]
        #----- stopping criterion -----#
        #1. each grid
        #------------------------------#
        err = abs((x_ij[:, 3] - x_ij[:, 2]) / x_ij[:, 2])
        if np.all(err < stop_cri):
```

```
            break
        else:
            x_ij[:, 2] = x_ij[:, 3]
            x_ij[:, 3] = 0.0
        #------------------------------#
        #2. Residual
        #------------------------------#
        # matrix_Ax = np.dot(A_ij, x_ij[:, 3])
        # tmp_sum_bAx = 0.0
        # tmp_sum_b   = 0.0
        # for i in range(num_grid**2):
        #     tmp_sum_bAx += abs(b_ij[i] - matrix_Ax[i])**2
        #     tmp_sum_b   += abs(b_ij[i])**2
        # residual = math.sqrt(tmp_sum_bAx) / math.sqrt(tmp_sum_b)
        # if (residual < stop_cri):
        #     break
        # else:
        #     x_ij[:, 2] = x_ij[:, 3]
        #     x_ij[:, 3] = 0.0

#3-6. SOR method
def SOR_parameter(num_grid):
    global SOR_para
    a = 1.0 + math.sin(math.pi / (float(num_grid)-1.0))
    SOR_para = 2.0 / a

def iteration_SOR(num_grid, A_ij, b_ij, x_ij, stop_cri, SOR_para, ite_max):
    global ite
    ite = 0
    x_ij[:, 2] = (T_b + T_t + T_b + T_r)/4.0 # initial value
    for m in range(int(ite_max)):
        ite += 1
        for me in range(num_grid**2):
            c_ij = 0.0
            d_ij = 0.0
            for you_l in range(me):
                c_ij += A_ij[me, you_l] * x_ij[you_l, 3]
            for you_r in range(me+1, num_grid**2):
                d_ij += A_ij[me, you_r] * x_ij[you_r, 2]
            x_ij[me, 3] = x_ij[me, 2] \
                        + SOR_para * ((b_ij[me] - c_ij - d_ij) / A_ij[me, me] - x_ij[me,
                              2])
        #----- stopping criterion -----#
        #1. each grid
        #------------------------------#
        err = abs((x_ij[:, 3] - x_ij[:, 2]) / x_ij[:, 2])
        if np.all(err < stop_cri):
            break
        else:
            x_ij[:, 2] = x_ij[:, 3]
            x_ij[:, 3] = 0.0
        #------------------------------#
        #2. Residual
        #------------------------------#
        # matrix_Ax = np.dot(A_ij, x_ij[:, 3])
        # tmp_sum_bAx = 0.0
        # tmp_sum_b   = 0.0
        # for i in range(num_grid**2):
        #     tmp_sum_bAx += abs(b_ij[i] - matrix_Ax[i])**2
        #     tmp_sum_b   += abs(b_ij[i])**2
        # residual = math.sqrt(tmp_sum_bAx) / math.sqrt(tmp_sum_b)
        # if (residual < stop_cri):
        #     break
        # else:
        #     x_ij[:, 2] = x_ij[:, 3]
        #     x_ij[:, 3] = 0.0

#3-7. np.linalg.solve
def np_linalg_solve(A_ij, b_ij, x_ij):
    X = np.linalg.solve(A_ij, b_ij)
    for i in range(len(x_ij[:, 3])):
        x_ij[i, 3] = X[i]
    end_time = time.perf_counter() # get time

#4. analytical solution
def exe_ana_sol(length):
    start_time = time.perf_counter() # get time
    global X_ana, Y_ana, T_ana
    x_ana = np.linspace(0, length, 100)
    y_ana = np.linspace(0, length, 100)
    X_ana, Y_ana = np.meshgrid(x_ana, y_ana)
```

```
    T_ana = np.zeros((X_ana.shape[0], X_ana.shape[1]))
    for i in range(len(x_ana)):
        for j in range(len(y_ana)):
            for m in range(1, 10000):
                tmp = float((2*m-1))*math.pi
                if m <= 100:
                    C_n = T_t / (math.sinh(tmp) * tmp)
                    T_ana[i, j] += 4.0 * C_n * math.sin (tmp/length*X_ana[i, j]) \
                                                 * math.sinh(tmp/length*Y_ana[i, j])
                if m > 100:
                     T_ana[i, j] += 4.0 * T_t / tmp * math.exp(tmp/length*(Y_ana[i,j]-length)) \
                                                        * math.sin(tmp/length*X_ana[i,j])
                #if T_ana[i,j] > 100:
                #    print(T_ana[i,j], i,j )
    end_time = time.perf_counter() # get time
    print('[Message]␣analytical␣solution␣␣␣␣␣␣:␣{:.2f}␣[s]'.format(end_time-start_time))

#5. call functions
#5-1. Gauss-Jordan
build_mesh(length, num_grid)
formulate_equation(length, num_grid, Delta_x, A_ij, b_ij, x_ij)
start_time = time.perf_counter() # get time
Gauss_Jordan(num_grid, A_ij, b_ij, x_ij)
end_time = time.perf_counter()   # get time
time_GJ  = end_time-start_time
x_ij_GJ = x_ij
print('[Message␣1/7]␣Guass-Jordan␣␣␣␣␣␣␣␣␣␣:␣{:.2f}␣[s]'.format(end_time-start_time))

#5-2. Gaussian elimination
build_mesh(length, num_grid)
formulate_equation(length, num_grid, Delta_x, A_ij, b_ij, x_ij)
start_time = time.perf_counter() # get time
Gaussian_elimination(num_grid, A_ij, b_ij, x_ij)
end_time = time.perf_counter()   # get time
time_GE  = end_time-start_time
x_ij_GE = x_ij
print('[Message␣2/7]␣Guassian␣elimination␣␣:␣{:.2f}␣[s]'.format(end_time-start_time))

#5-3. LU decomposition
build_mesh(length, num_grid)
formulate_equation(length, num_grid, Delta_x, A_ij, b_ij, x_ij)
start_time = time.perf_counter() # get time
LU_decomposition(A_ij, b_ij, x_ij)
end_time = time.perf_counter() # get time
time_LU  = end_time-start_time
x_ij_LU = x_ij
print('[Message␣3/7]␣LU␣decomposition␣␣␣␣␣␣:␣{:.2f}␣[s]'.format(end_time-start_time))

#5-4. Jacobi iteration
build_mesh(length, num_grid)
formulate_equation(length, num_grid, Delta_x, A_ij, b_ij, x_ij)
start_time = time.perf_counter() # get time
iteration_Jacobi(num_grid, A_ij, b_ij, x_ij, stop_cri, 1E+10)
end_time = time.perf_counter()   # get time
time_Ji  = end_time-start_time
x_ij_Ji = x_ij
print('[Message␣4/7]␣Jacobi␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:␣{:.2f}␣[s]'.format(end_time-start_time))
print('␣␣␣␣␣␣␣␣␣␣␣␣␣␣Iteration␣␣␣␣␣␣␣␣␣␣␣␣␣:␣{:.1f}'.format(ite))

#5-5. Gauss-Seidel iteration
build_mesh(length, num_grid)
formulate_equation(length, num_grid, Delta_x, A_ij, b_ij, x_ij)
start_time = time.perf_counter() # get time
iteration_Gauss_Seidel(num_grid, A_ij, b_ij, x_ij, stop_cri, 1E+10)
end_time = time.perf_counter()   # get time
time_GS  = end_time-start_time
x_ij_GS = x_ij
print('[Message␣5/7]␣Gauss-Seidel␣␣␣␣␣␣␣␣␣␣:␣{:.2f}␣[s]'.format(end_time-start_time))
print('␣␣␣␣␣␣␣␣␣␣␣␣␣␣Iteration␣␣␣␣␣␣␣␣␣␣␣␣␣:␣{:.1f}'.format(ite))

#5-6. SOR iteration
SOR_parameter(num_grid)
build_mesh(length, num_grid)
formulate_equation(length, num_grid, Delta_x, A_ij, b_ij, x_ij)
start_time = time.perf_counter() # get time
iteration_SOR(num_grid, A_ij, b_ij, x_ij, stop_cri, SOR_para, 1E+10)
end_time = time.perf_counter()   # get time
time_SOR = end_time-start_time
x_ij_SOR = x_ij
print('[Message␣6/7]␣SOR␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:␣{:.2f}␣[s]'.format(end_time-start_time))
```

```
print('␣␣␣␣␣␣␣␣␣␣␣␣␣␣SOR␣parameter␣␣␣␣␣␣␣␣:{:.3f}'.format(SOR_para))
print('␣␣␣␣␣␣␣␣␣␣␣␣␣␣Iteration␣␣␣␣␣␣␣␣␣␣␣␣:␣{:.1f}'.format(ite))

#5-7. np.linalg.solve
build_mesh(length, num_grid)
formulate_equation(length, num_grid, Delta_x, A_ij, b_ij, x_ij)
start_time = time.perf_counter() # get time
np_linalg_solve(A_ij, b_ij, x_ij)
end_time = time.perf_counter()   # get time
time_NP  = end_time-start_time
x_ij_NP = x_ij
print('[Message␣7/7]␣np.linalg.solve␣␣␣␣␣␣:␣{:.2f}␣[s]'.format(end_time-start_time))

if (iteration == 0):
        ite_step = 100
        ite_process_check = int(ite_step / 3)

        #1. Jacobi method
        print('[Message]␣iteration␣check␣running␣Jacobi␣method␣␣␣␣␣␣␣...␣start')
        Jacobi_check = np.zeros((ite_step))
        for ite_check in range(1, ite_step+1):
            build_mesh(length, num_grid)
            formulate_equation(length, num_grid, Delta_x, A_ij, b_ij, x_ij)
            iteration_Jacobi(num_grid, A_ij, b_ij, x_ij, 1E-10, ite_check)
            tmp_sum = 0.0
            for p in range(num_grid**2):
                tmp_sum += ((x_ij_GJ[p, 3] - x_ij[p, 2]) / x_ij_GJ[p, 3])**2
            rms = math.sqrt(tmp_sum / num_grid**2)
            Jacobi_check[ite_check-1] = rms
            if (ite_check == ite_process_check):
                print('␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣...␣1/3␣completed')
            elif (ite_check == 2*ite_process_check):
                print('␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣...␣2/3␣completed')
        print('␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣...␣3/3␣completed')

        #2. Gauss-Seidel method
        print('[Message]␣iteration␣check␣running␣Gauss-Seidel␣method␣...␣start')
        Gauss_Seidel_check = np.zeros((ite_step))
        for ite_check in range(1, ite_step+1):
            build_mesh(length, num_grid)
            formulate_equation(length, num_grid, Delta_x, A_ij, b_ij, x_ij)
            iteration_Gauss_Seidel(num_grid, A_ij, b_ij, x_ij, 1E-10, ite_check)
            tmp_sum = 0.0
            for p in range(num_grid**2):
                tmp_sum += ((x_ij_GJ[p, 3] - x_ij[p, 2]) / x_ij_GJ[p, 3])**2
            rms = math.sqrt(tmp_sum / num_grid**2)
            Gauss_Seidel_check[ite_check-1] = rms
            if (ite_check == ite_process_check):
                print('␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣...␣1/3␣completed')
            elif (ite_check == 2*ite_process_check):
                print('␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣...␣2/3␣completed')
        print('␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣...␣3/3␣completed')

        #3. SOR method
        print('[Message]␣iteration␣check␣running␣SOR␣method␣␣␣␣␣␣␣␣␣␣...␣start')
        SOR_check = np.zeros((ite_step, 6)) # 6 is SOR parameter=[0.5, 1, ideal, 1.5, 2.0, 2.5]
        SOR_para_check = np.array((0.5, 1.0, SOR_para, 1.5, 2.0, 2.5))
        for q in range(6):
            for ite_check in range(1, ite_step+1):
                build_mesh(length, num_grid)
                formulate_equation(length, num_grid, Delta_x, A_ij, b_ij, x_ij)
                iteration_SOR(num_grid, A_ij, b_ij, x_ij, 1E-10, SOR_para_check[q], ite_check)
                tmp_sum = 0.0
                for p in range(num_grid**2):
                    tmp_sum += ((x_ij_GJ[p, 3] - x_ij[p, 2]) / x_ij_GJ[p, 3])**2
                rms = math.sqrt(tmp_sum / num_grid**2)
                SOR_check[ite_check-1, q] = rms
            print('␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣...␣{}/6␣completed'.format(q+1))

if (ana_sol == 0):
    exe_ana_sol(length)

if (save_data == 0):
    # np.savetxt('./data/{}_GJ_{}.dat'.format(file_name, num_grid), x_ij_GJ)
```

```python
    # np.savetxt('./data/{}_GE_{}.dat'.format(file_name, num_grid), x_ij_GE)
    # np.savetxt('./data/{}_LU_{}.dat'.format(file_name, num_grid), x_ij_LU)
    # np.savetxt('./data/{}_Ji_{}.dat'.format(file_name, num_grid), x_ij_Ji)
    # np.savetxt('./data/{}_GS_{}.dat'.format(file_name, num_grid), x_ij_GS)
    # np.savetxt('./data/{}_SOR_{}.dat'.format(file_name, num_grid), x_ij_SOR)
    # np.savetxt('./data/{}_NP_{}.dat'.format(file_name, num_grid), x_ij_NP)

    CPU_time = np.hstack((time_GJ, time_GE, time_LU, time_Ji, time_GS, time_SOR, time_NP))
    np.savetxt('./data/CPU_time/{}_CPUtime_{}.dat'.format(file_name, num_grid), CPU_time)

#====================#
# figure
#====================#
if (plot_fig == 0):
    print('[Message] producing figure...')

    #-----------------------------------
    # Fig1. temperature distribution
    #-----------------------------------
    # figure and axis environment
    fig, axs = plt.subplots(3, 3, figsize=(18, 22), facecolor='white', subplot_kw={'facecolor':'white'})
    # margin between figure
    plt.subplots_adjust(left=0.08, right=0.95, bottom=0.02, top=0.95, wspace=0.35, hspace=0.1)
    # Gauss-Jordan
    fig_1 = axs[0, 0].scatter(x_ij_GJ[:,0], x_ij_GJ[:,1], c=x_ij_GJ[:,3], cmap='jet', \
                              vmin=min(T_t, T_b, T_r, T_l), vmax=max(T_t, T_b, T_r, T_l))
    bar_1 = plt.colorbar(fig_1, aspect=60, ax=axs[0,0], orientation='horizontal', pad=0.18)
    if (Tex_user == 0):
        bar_1.set_label(r'$T$', size=24, labelpad=14)
    else:
        bar_1.set_label('T', size=24, labelpad=14)
    bar_1.ax.tick_params(direction='out', length=2.5, width=0.8, labelsize=18)
    # Gaussian elimination
    fig_2 = axs[0, 1].scatter(x_ij_GE[:,0], x_ij_GE[:,1], c=x_ij_GE[:,3], cmap='jet', \
                              vmin=min(T_t, T_b, T_r, T_l), vmax=max(T_t, T_b, T_r, T_l))
    bar_2 = plt.colorbar(fig_2, aspect=60, ax=axs[0,1], orientation='horizontal', pad=0.18)
    if (Tex_user == 0):
        bar_2.set_label(r'$T$', size=24, labelpad=14)
    else:
        bar_2.set_label('T', size=24, labelpad=14)
    bar_2.ax.tick_params(direction='out', length=2.5, width=0.8, labelsize=18)
    # LU decomposition
    fig_3 = axs[0, 2].scatter(x_ij_LU[:,0], x_ij_LU[:,1], c=x_ij_LU[:,3], cmap='jet', \
                              vmin=min(T_t, T_b, T_r, T_l), vmax=max(T_t, T_b, T_r, T_l))
    bar_3 = plt.colorbar(fig_3, aspect=60, ax=axs[0,2], orientation='horizontal', pad=0.18)
    if (Tex_user == 0):
        bar_3.set_label(r'$T$', size=24, labelpad=14)
    else:
        bar_3.set_label('T', size=24, labelpad=14)
    bar_3.ax.tick_params(direction='out', length=2.5, width=0.8, labelsize=18)
    # Jacobi iteration
    fig_4 = axs[1, 0].scatter(x_ij_Ji[:,0], x_ij_Ji[:,1], c=x_ij_Ji[:,3], cmap='jet', \
                              vmin=min(T_t, T_b, T_r, T_l), vmax=max(T_t, T_b, T_r, T_l))
    bar_4 = plt.colorbar(fig_4, aspect=60, ax=axs[1,0], orientation='horizontal', pad=0.18)
    if (Tex_user == 0):
        bar_4.set_label(r'$T$', size=24, labelpad=14)
    else:
        bar_4.set_label('T', size=24, labelpad=14)
    bar_4.ax.tick_params(direction='out', length=2.5, width=0.8, labelsize=18)
    # Gauss-Seidel
    fig_5 = axs[1, 1].scatter(x_ij_GS[:,0], x_ij_GS[:,1], c=x_ij_GS[:,3], cmap='jet', \
                              vmin=min(T_t, T_b, T_r, T_l), vmax=max(T_t, T_b, T_r, T_l))
    bar_5 = plt.colorbar(fig_5, aspect=60, ax=axs[1,1], orientation='horizontal', pad=0.18)
    if (Tex_user == 0):
        bar_5.set_label(r'$T$', size=24, labelpad=14)
    else:
        bar_5.set_label('T', size=24, labelpad=14)
```

```
    bar_5.ax.tick_params(direction='out', length=2.5, width=0.8, labelsize=18)
    # SOR
    fig_6 = axs[1, 2].scatter(x_ij_SOR[:,0], x_ij_SOR[:,1], c=x_ij_SOR[:,3], cmap='jet', \
                              vmin=min(T_t, T_b, T_r, T_l), vmax=max(T_t, T_b, T_r, T_l))
    bar_6 = plt.colorbar(fig_6, aspect=60, ax=axs[1,2], orientation='horizontal', pad=0.18)
    if (Tex_user == 0):
        bar_6.set_label(r'$T$', size=24, labelpad=14)
    else:
        bar_6.set_label('T', size=24, labelpad=14)
    bar_6.ax.tick_params(direction='out', length=2.5, width=0.8, labelsize=18)
    # np.linalg.solve
    fig_7 = axs[2, 0].scatter(x_ij_NP[:,0], x_ij_NP[:,1], c=x_ij_NP[:,3], cmap='jet', \
                              vmin=min(T_t, T_b, T_r, T_l), vmax=max(T_t, T_b, T_r, T_l))
    bar_7 = plt.colorbar(fig_7, aspect=60, ax=axs[2,0], orientation='horizontal', pad=0.18)
    if (Tex_user == 0):
        bar_7.set_label(r'$T$', size=24, labelpad=14)
    else:
        bar_7.set_label('T', size=24, labelpad=14)
    bar_7.ax.tick_params(direction='out', length=2.5, width=0.8, labelsize=18)
    # np.linalg.solve
    fig_8 = axs[2, 1].scatter(x_ij_NP[:,0], x_ij_NP[:,1], c=x_ij_NP[:,3], cmap='jet', \
                              vmin=min(T_t, T_b, T_r, T_l), vmax=max(T_t, T_b, T_r, T_l))
    bar_8 = plt.colorbar(fig_8, aspect=60, ax=axs[2,1], orientation='horizontal', pad=0.18)
    if (Tex_user == 0):
        bar_8.set_label(r'$T$', size=24, labelpad=14)
    else:
        bar_8.set_label('T', size=24, labelpad=14)
    bar_8.ax.tick_params(direction='out', length=2.5, width=0.8, labelsize=18)
    # np.linalg.solve
    fig_9 = axs[2, 2].scatter(x_ij_NP[:,0], x_ij_NP[:,1], c=x_ij_NP[:,3], cmap='jet', \
                              vmin=min(T_t, T_b, T_r, T_l), vmax=max(T_t, T_b, T_r, T_l))
    bar_9 = plt.colorbar(fig_9, aspect=60, ax=axs[2,2], orientation='horizontal', pad=0.18)
    if (Tex_user == 0):
        bar_9.set_label(r'$T$', size=24, labelpad=14)
    else:
        bar_9.set_label('T', size=24, labelpad=14)
    bar_9.ax.tick_params(direction='out', length=2.5, width=0.8, labelsize=18)
    # axis labels, ticks, outer frame, lines
    for j in range(3):
        for k in range(3):
                if (Tex_user == 0):
                    axs[j, k].set_xlabel(r'$x$', fontsize=24, labelpad=18)
                    axs[j, k].set_ylabel(r'$y$', fontsize=24, labelpad=18)
                else:
                    axs[j, k].set_xlabel('x', fontsize=24, labelpad=18)
                    axs[j, k].set_ylabel('y', fontsize=24, labelpad=18)
                axs[j, k].tick_params(axis='both', which='major', direction='out', length=3, width=0.8, labelsize=18)
                axs[j, k].spines["bottom"].set_linewidth(1.2)
                axs[j, k].spines["top"].set_linewidth(1.2)
                axs[j, k].spines["right"].set_linewidth(1.2)
                axs[j, k].spines["left"].set_linewidth(1.2)
                axs[j, k].axvline(x=0,      ymin=0, ymax=length, color='silver', linewidth=0.8, linestyle='--')
                axs[j, k].axvline(x=length, ymin=0, ymax=length, color='silver', linewidth=0.8, linestyle='--')
                axs[j, k].axhline(y=0,      xmin=0, xmax=length, color='silver', linewidth=0.8, linestyle='--')
                axs[j, k].axhline(y=length, xmin=0, xmax=length, color='silver', linewidth=0.8, linestyle='--')
    # title
    axs[0,0].set_title('GJ', fontsize=30, color='k')
    axs[0,1].set_title('GE', fontsize=30, color='k')
    axs[0,2].set_title('LU', fontsize=30, color='k')
    axs[1,0].set_title('Ji', fontsize=30, color='k')
    axs[1,1].set_title('GS', fontsize=30, color='k')
    axs[1,2].set_title('SOR', fontsize=30, color='k')
    axs[2,0].set_title('np.linalg.solve', fontsize=30, color='k')
    axs[2,1].set_title('--', fontsize=30, color='k')
    axs[2,2].set_title('--', fontsize=30, color='k')
    # save
    fig.savefig('./fig/{}_tem_{}.png'.format(file_name, num_grid), format='png', dpi
```

```
        =300, transparent=False)
    fig.savefig('./fig/{}_tem_{}.pdf'.format(file_name, num_grid), format='pdf', transparent=True)
    # close
    plt.close()

    #-----------------------------------
    # Fig2. iteraion
    #-----------------------------------
    if (iteration == 0):
        # figure and axis environment
        my_color = ['black', 'cyan', 'magenta', 'gray', 'blue', 'green']
        my_style = [':', ':', '-', '--', '--', '--']
        fig, axs = plt.subplots(1, 1, figsize=(7, 7), facecolor='white', subplot_kw={'facecolor':'white'})
        # margin between figures
        plt.subplots_adjust(left=0.19, right=0.93, bottom=0.14, top=0.91, wspace=0.4, hspace=0.3)
        # plot
        plt.plot(Jacobi_check[:],       label='Ji', color='black', linestyle='-')
        plt.plot(Gauss_Seidel_check[:], label='GS', color='red', linestyle='-')
        for i in range(6):
            plt.plot(SOR_check[:, i], label='SOR ({})'.format(SOR_para_check[i]), \
                    color=my_color[i], linestyle=my_style[i])
        # log scale
        axs.set_yscale('log')
        # legend
        axs.legend(fontsize=16, fancybox=True, edgecolor='silver')
        # axis labels
        axs.set_xlabel('iteration step', fontsize=26, labelpad=10)
        axs.set_ylabel('residual (rms)', fontsize=26, labelpad=12)
        # title
        axs.set_title('N={}'.format(num_grid**2), fontsize=24, color='k', y=1.02)
        # grid
        axs.grid(which='major', color='silver', linewidth=0.1)
        # direction and width of ticks
        axs.tick_params(axis='both', which='major', direction='out', length=3, width=0.8, labelsize=20)
        # width of outer frame
        axs.spines["bottom"].set_linewidth(1.2)
        axs.spines["top"].set_linewidth(1.2)
        axs.spines["right"].set_linewidth(1.2)
        axs.spines["left"].set_linewidth(1.2)
        # save
        fig.savefig('./fig/{}_ite_{}.png'.format(file_name, num_grid), format='png', dpi=300, transparent=False)
        fig.savefig('./fig/{}_ite_{}.pdf'.format(file_name, num_grid), format='pdf', transparent=True)
        # close
        plt.close()

    #-----------------------------------
    # Fig3. SOR parameter
    #-----------------------------------
    n_SOR = np.linspace(3**2, 100**2, 100)
    theory_SOR_para = np.zeros((100))
    for i in range(100):
        SOR_parameter(math.sqrt(n_SOR[i]))
        theory_SOR_para[i] = SOR_para

    # figure and axis environment
    fig, axs = plt.subplots(1, 1, figsize=(7, 7), facecolor='white', subplot_kw={'facecolor':'white'})
    # margin between figures
    plt.subplots_adjust(left=0.16, right=0.9, bottom=0.14, top=0.91, wspace=0.4, hspace=0.3)
    # plot
    plt.plot(n_SOR**(1.0/2.0), theory_SOR_para, color='black', linestyle='-')
    # axis labels
    axs.set_xlabel('n', fontsize=26, labelpad=10)
    if (Tex_user == 0):
        axs.set_ylabel(r'$\omega_{o}$', fontsize=26, labelpad=14)
    else:
        axs.set_ylabel('SOR parameter', fontsize=26, labelpad=14)
    # grid
    axs.grid(which='major', color='silver', linewidth=0.1)
    # direction and width of ticks
    axs.tick_params(axis='both', which='major', direction='out', length=3, width=0.8, labelsize=20)
    # width of outer frame
    axs.spines["bottom"].set_linewidth(1.2)
```

```
    axs.spines["top"].set_linewidth(1.2)
    axs.spines["right"].set_linewidth(1.2)
    axs.spines["left"].set_linewidth(1.2)
    # save
    fig.savefig('./fig/SORpara.png'.format(file_name, num_grid), format='png', dpi=300, transparent=False)
    fig.savefig('./fig/SORpara.pdf'.format(file_name, num_grid), format='pdf', transparent=True)
    # close
    plt.close()

    #----------------------------------
    # Fig4. analytical solution
    #----------------------------------
    if (ana_sol == 0):
        # figure and axis environment
        fig, axs = plt.subplots(1, 1, figsize=(6, 7.5), facecolor='white', subplot_kw={'facecolor':'white'})
        # margin between figures
        plt.subplots_adjust(left=0.15, right=0.9, bottom=0.05, top=0.92, wspace=0.4, hspace=0.3)
        # plot
        CS = axs.contour(X_ana, Y_ana, T_ana, colors='black')
        axs.clabel(CS, inline=True)
        CSf = axs.contourf(X_ana, Y_ana, T_ana, cmap='jet')
        cbar = plt.colorbar(CSf, aspect=60, ax=axs, orientation='horizontal', pad=0.18)
        if (Tex_user == 0):
            cbar.set_label(r'$T$', size=22, labelpad=12)
        else:
            cbar.set_label('T', size=22, labelpad=12)
        cbar.ax.tick_params(direction='out', length=2.5, width=0.8, labelsize=16)
        cbar.add_lines(CS)
        # axis labels
        if (Tex_user == 0):
            axs.set_xlabel(r'$x$', fontsize=26, labelpad=10)
            axs.set_ylabel(r'$y$', fontsize=26, labelpad=12)
        else:
            axs.set_xlabel('x', fontsize=26, labelpad=10)
            axs.set_ylabel('y', fontsize=26, labelpad=12)
        # grid
        axs.grid(which='major', color='silver', linewidth=0.1)
        # direction and width of ticks
        axs.tick_params(axis='both', which='major', direction='out', length=3, width=0.8, labelsize=16)
        # width of outer frame
        axs.spines["bottom"].set_linewidth(1.2)
        axs.spines["top"].set_linewidth(1.2)
        axs.spines["right"].set_linewidth(1.2)
        axs.spines["left"].set_linewidth(1.2)
        # save
        fig.savefig('./fig/{}_analytical_sol_{}.png'.format(file_name, num_grid), format='png', dpi=300, transparent=False)
        fig.savefig('./fig/{}_analytical_sol_{}.pdf'.format(file_name, num_grid), format='pdf', transparent=True)
        # close
        plt.close()


###
program_end_time = time.perf_counter() # get time
print('[Message]␣Program␣has␣finished␣!␣␣␣:␣{:.2f}␣[s]'.format(program_end_time-program_start_time))
```

■ plot_different_N.py

```python
#======================================================================================#
#                Soving Laplacian Difference Equation by numerical methods             #
#                                 --  for figure  --                                   #
#--------------------------------------------------------------------------------------#
#                        Copyright by Kensuke Shobuzako (2023)                         #
#======================================================================================#


#====================#
# charm
#====================#
import numpy as np
import matplotlib.pyplot as plt
import math
import glob
import sys
import os
import time
from scipy import integrate
from scipy.interpolate import interp1d
from scipy.linalg import lu_factor, lu_solve
from matplotlib import rc
rc('text', usetex=True)

#====================#
# read
#====================#
path_names  = glob.glob('./data/CPU_time/*.dat') # get file paths
file_size   = len(path_names)                     # count the number of files
cpu_time    = np.zeros((file_size, 7))            # (num_N, (GJ,GE,LU,Ji,GS,SOR,NP))
method_name = ['GJ', 'GE', 'LU', 'Ji', 'GS', 'SOR', 'NP']
print('[Message] Reading files below... ' )
j = 0
N = np.zeros((file_size)) # (N value...)
for i in path_names:
    cpu_time[j, :] = np.loadtxt(i)
    tmp_0 = i[31:]
    tmp_1 = tmp_0[:-4]
    N[j] = int(tmp_1)
    print('          ', i[16:])
    j += 1

N_sort = np.sort(N)
cpu_time_sort = np.sort(cpu_time, axis=0)

#====================#
# figure
#====================#
my_color = ['black', 'red', 'blue', 'black', 'red', 'blue', 'black']
my_style = ['-', '-', '-', '--', '--', '--', ':']
# figure and axis environment
fig, axs = plt.subplots(1, 1, figsize=(6, 5.9), facecolor='white', subplot_kw={'facecolor':'white'})
# margin between figures
plt.subplots_adjust(left=0.2, right=0.9, bottom=0.14, top=0.93, wspace=0.4, hspace=0.3)
# plot
j = 0
for i in method_name:
    plt.plot(N_sort**2, cpu_time_sort[:, j], label=i, color=my_color[j], linestyle=my_style[j])
    j += 1
# legend
axs.legend(fontsize=14, fancybox=True, edgecolor='silver')
# axis labels
axs.set_xlabel('N', fontsize=22, labelpad=10)
axs.set_ylabel('CPU time [s]', fontsize=22, labelpad=12)
# grid
axs.grid(which='major', color='silver', linewidth=0.1)
# direction and width of ticks
axs.tick_params(axis='both', which='major', direction='out', length=3, width=0.8, labelsize=16)
# width of outer frame
axs.spines["bottom"].set_linewidth(1.2)
axs.spines["top"].set_linewidth(1.2)
axs.spines["right"].set_linewidth(1.2)
axs.spines["left"].set_linewidth(1.2)
# save
fig.savefig('./fig/CPUtime.png', format='png', dpi=300, transparent=False)
```

```
fig.savefig('./fig/CPUtime.pdf', transparent=True)
# close
plt.close()
```