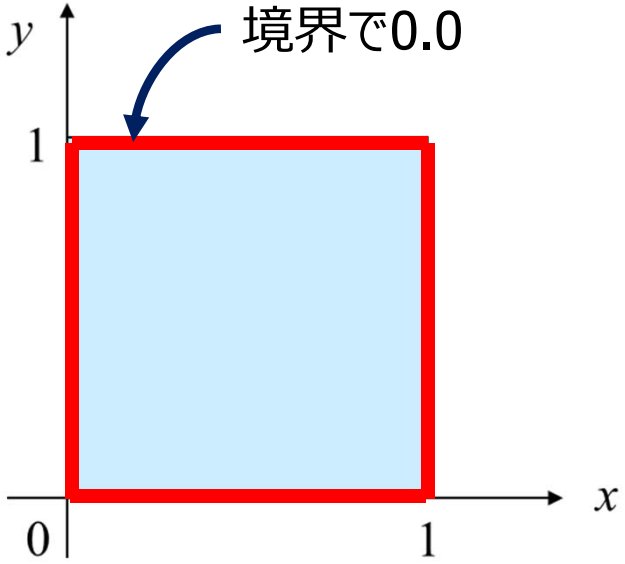
KOBE HPC サマースクール 2019（初級）

13. 熱伝導問題のハイブリッド並列計算

2次元定常熱伝導問題

- 2次元正方形領域 [0,1]×[0,1] に一定の熱を加え続けた時の領域の温度がどうなるか？

$$\Delta u + 10 = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + 10 = 0$$



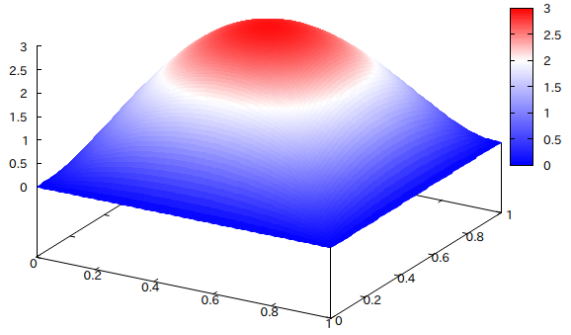
- 境界条件

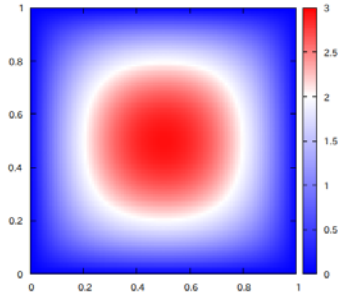
$$u(0, y) = 0.0 \quad (0 \le y \le 1)$$

$$u(1, y) = 0.0 \quad (0 \le y \le 1)$$

$$u(x, 0) = 0.0 \quad (0 \le x \le 1)$$

$$u(x, 1) = 0.0 \quad (0 \le x \le 1)$$

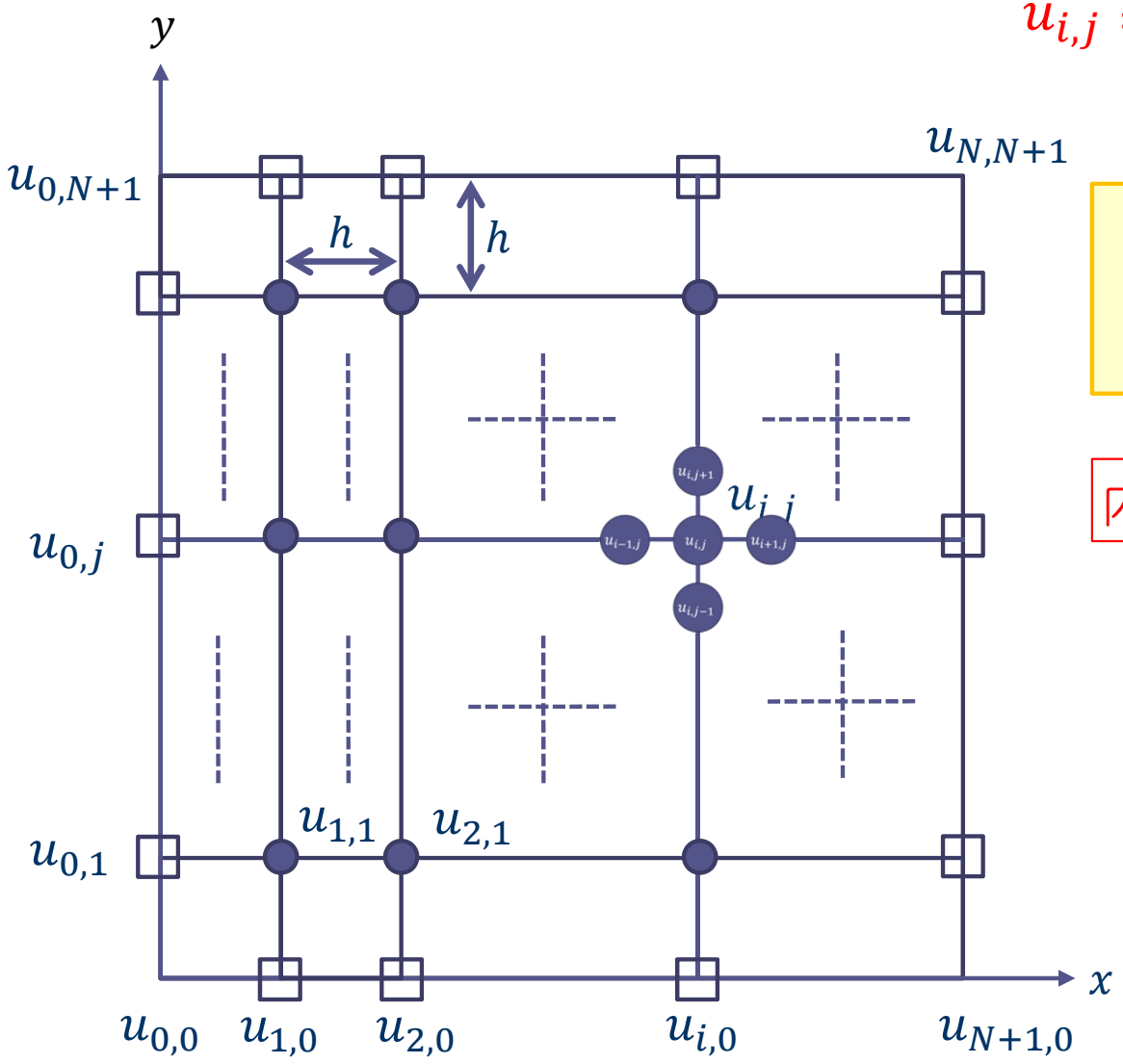




離散化格子と変数

x方向，y方向どちらも $(N+1)$ 等分する． $h = 1/(N+1)$

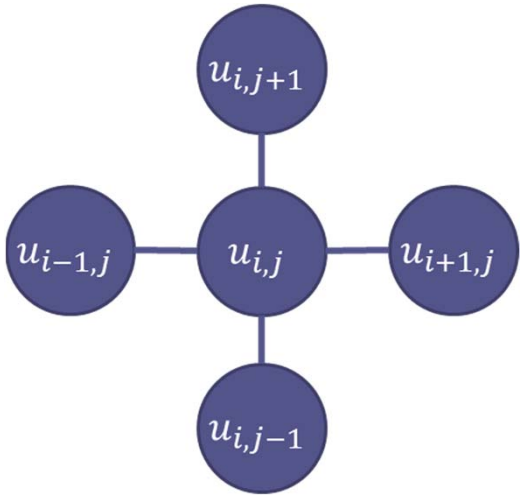
$u_{i,j} \triangleq u(ih, jh) \quad (i, j = 0, 1, 2, \ldots, N+1)$



□ 境界条件として既知の値

● 内部の格子点（未知数）

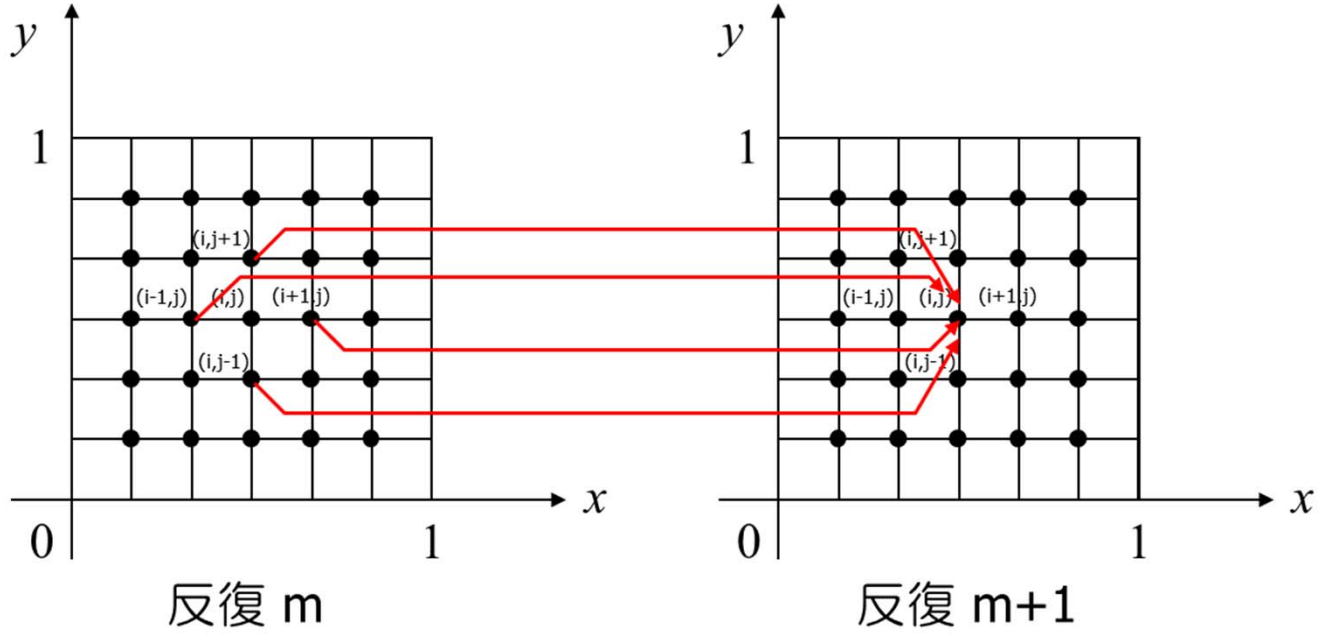
内部の格子点に関する関係式を作る．



熱伝導問題へのヤコビ法の適用

- 反復ベクトル生成のアルゴリズム

```
for( i=1; i<=N; i++ ) {
  for( j=1; j<=N; j++ ) {
    u_{i,j}^{(m+1)} = (u_{i-1,j}^{(m)}+u_{i+1,j}^{(m)}+u_{i,j-1}^{(m)}+u_{i,j+1}^{(m)})*0.25 + 10.0*dx*dx;
  };
};
```



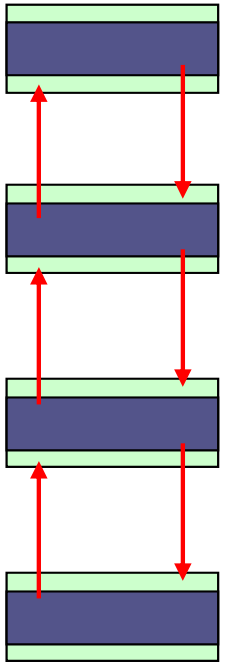
演習13-4 ：プログラム heat2d.c の実行

- 2次元定常熱伝導問題の逐次プログラム heat2d.c をコンパイルして実行せよ.

```
$ cp  /tmp/Summer/M-4/heat2d.c ./
$ icc -O3 heat2d.c
$ ./a.out
```

  - プログラムでは $\mathrm{N} = 128$ の時に，反復ベクトルの差のノルム $\left\| u_{i,j}^{\mathrm{m}+1} - u_{i,j}^{\mathrm{m}} \right\|_2$ がある閾値$\epsilon = 1.0 \times 10^{-3}$ より小さくなったら反復を停止している.

heat2d.c のMPIによる並列化



並列化のヒント

- 2次元配列 u，u0 をブロック行分割
- u の計算をする前に，u[is:ie][]をu0にコピーし，
  - 上のプロセスのu0の ie 行を下のプロセスの (is-1) 行へ，
  - 下のプロセスのu0の is 行を上のプロセスの (ie+1) 行へ，

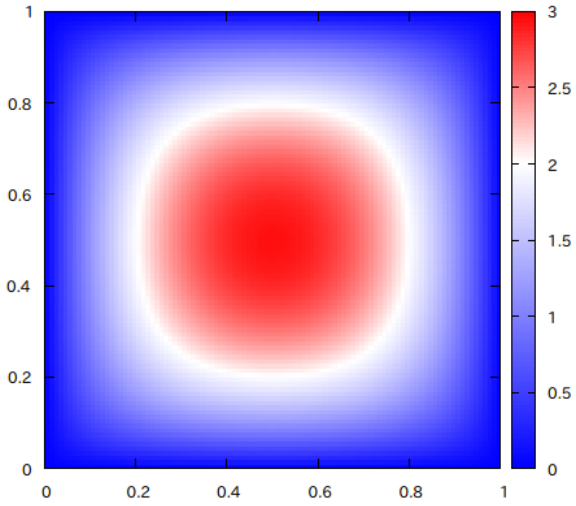
  送る.
  - 並列化sr_matrix.c と同様に，mpi_sendrecv を用いて送受信

上下のプロセスから1行を受信
（受信用の領域を確保しておく）

- u の is ～ ie 行の計算を行う.
- 反復ベクトルの差のノルムの部分和を計算し，MPI_Allgather で総和を取り，収束の判定を行うようにする.
  - ローカルで残差の部分和を計算し，各プロセスでノルムを計算する.

演習13-1 heat2d.cの並列化

- heat2d.cをMPIを用いて並列化せよ.
- N=128とし，プロセス数1, 2, 4, 8 として，反復開始から終了までの計算時間を計測し，速度向上率を求めよ．また，それをグラフにせよ.
- 計算結果を gnuplotで図示せよ.
  - 計算結果の出力は，プロセス0にデータを集め，ファイルに出力する.



演習13-2 ハイブリッド並列化

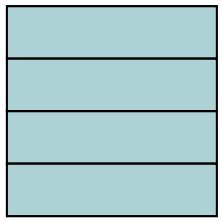
- MPIを用いて並列化したheat2d.cに，さらにopenMPのディレクティブを挿入し，タスク並列とプロセス並列によりさらに実行時間が削減されることを確認せよ.
  - #pragma omp parallel を利用する.
  - omp.h をインクルードするのを忘れずに.
- コンパイル
  - `icc␣-qopenmp␣xxxxx.c␣-lmpi`

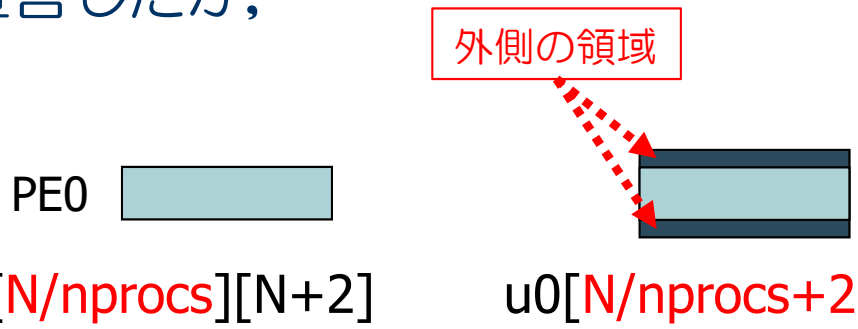
【サンプル】Hybrid並列プログラムの実行シェル

```
#!/bin/bash

#PBS -N hybrid                                  ジョブ名の指定
#PBS -q S
#PBS -j oe
#PBS -l select=2:ncpus=4:mpiprocs=2             select:    計算ノード数
                                                ncpus :    1計算ノード内のコア数
source /etc/profile.d/modules.sh                mpiprocs:  1計算ノード内のMPIプロセス数
module load intel
module load mpt

export KMP_AFFINITY=disabled
export OMP_NUM_THREADS=2

cd ${PBS_O_WORKDIR}
mpiexec_mpt omplace -nt ${OMP_NUM_THREADS} ./a.out
```

上級者になるために．．．

- 大規模問題を解く場合には，1プロセスのメモリ使用容量を少なくする必要が出てくる．
- 今回のプログラムでは，どのプロセスも u[N+2,N+2], u0[N+2,N+2] として，計算全領域の変数を宣言したが，





各プロセスは，これだけの大きさをもてば良いはず．

- 実行前から並列数（nprocs）が固定されていれば，上記のように，プログラムの先頭で小さい領域を宣言することも出来るが，汎用のプログラムとしては，nprocsを実行開始時に決めたい．
- 配列を動的に確保する必要がある（malloc関数）．
  - サンプルプログラム /tmp/Summer/M-4/alloc_mpi_heat2d.c

補足：2次元定常熱伝導問題の離散化

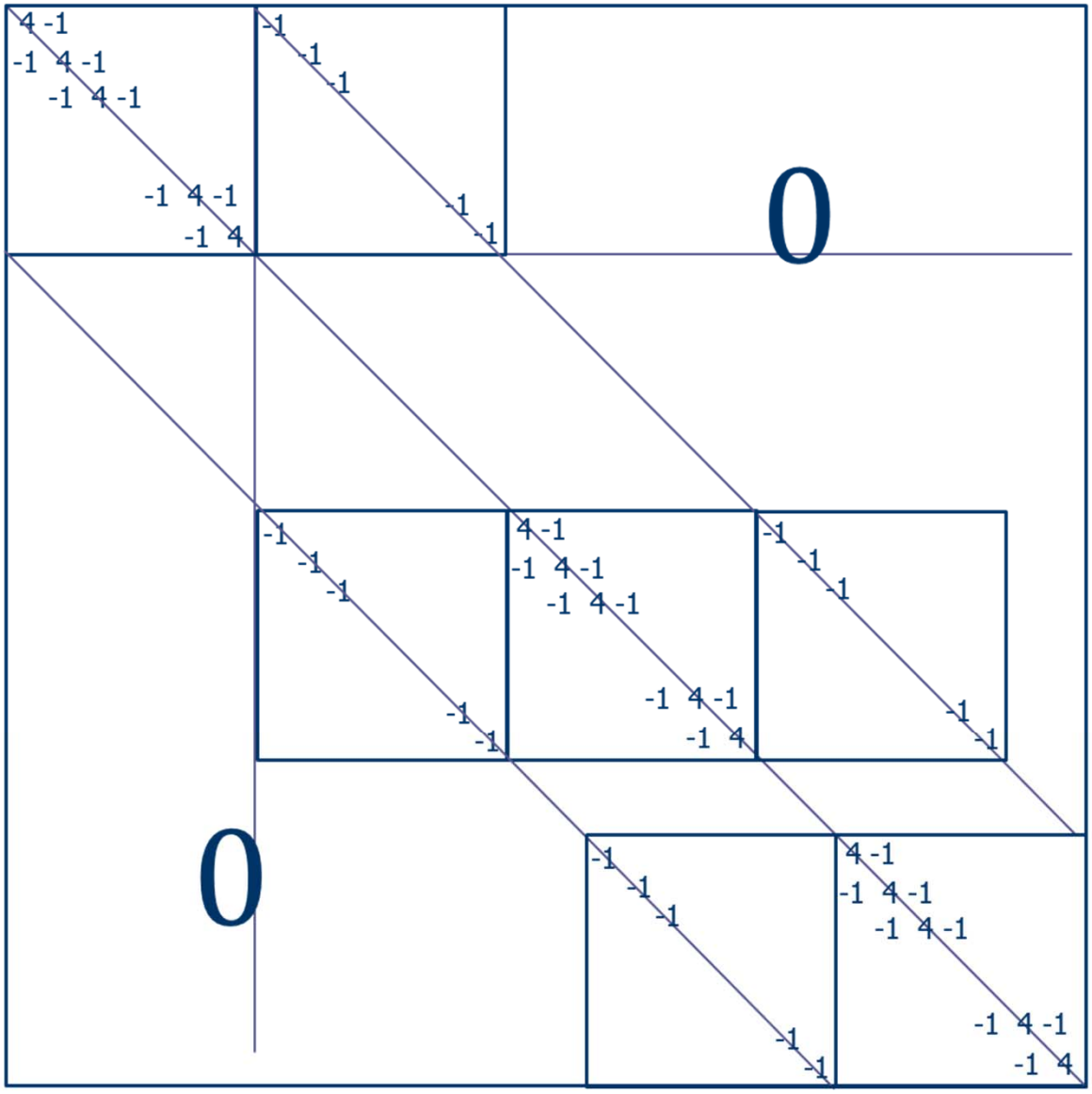
$u_{i,j}$を辞書式順序で並べたベクトルを$\boldsymbol{u}$とすると，点(i,j)に関する離散式は

$$u_{i,j-1} + u_{i-1,j} - 4u_{i,j} + u_{i+1,j} + u_{i,j+1} = -10\,h^2 \quad (i,j = 1,2,\ldots,N)$$

$$\boldsymbol{u} = \begin{bmatrix} u_{1,1} \\ u_{2,1} \\ u_{3,1} \\ \vdots \\ u_{N,1} \\ u_{1,2} \\ u_{2,2} \\ u_{3,2} \\ \vdots \\ u_{N,2} \\ \vdots \\ \vdots \\ u_{1,j} \\ \vdots \\ u_{i,j} \\ \vdots \\ u_{N,j} \\ \vdots \\ \vdots \\ u_{1,N} \\ u_{2,N} \\ u_{3,N} \\ \vdots \\ u_{N,N} \end{bmatrix}$$

$$[0 \quad \cdots \quad \underbrace{1 \quad \cdots \quad 1}_{\text{N離れている}} \quad -4 \quad \underbrace{1 \quad \cdots \quad 1}_{\text{N離れている}} \quad 0 \quad \cdots \quad 0] \begin{bmatrix} u_{1,1} \\ u_{2,1} \\ u_{3,1} \\ \vdots \\ u_{N,1} \\ u_{1,2} \\ u_{2,2} \\ u_{3,2} \\ \vdots \\ u_{N,2} \\ \vdots \\ \vdots \\ u_{1,j} \\ \vdots \\ u_{i,j} \\ \vdots \\ u_{N,j} \\ \vdots \\ \vdots \\ u_{1,N} \\ u_{2,N} \\ u_{3,N} \\ \vdots \\ u_{N,N} \end{bmatrix} = \begin{bmatrix} b_{1,1} \\ b_{2,1} \\ b_{3,1} \\ \vdots \\ b_{N,1} \\ b_{1,2} \\ b_{2,2} \\ b_{3,2} \\ \vdots \\ b_{N,2} \\ \vdots \\ \vdots \\ b_{1,j} \\ \vdots \\ b_{i,j} \\ \vdots \\ b_{N,j} \\ \vdots \\ \vdots \\ b_{1,N} \\ b_{2,N} \\ b_{3,N} \\ \vdots \\ b_{N,N} \end{bmatrix}$$

行列を用いた表現



$$\begin{bmatrix} u_{1,1} \\ u_{2,1} \\ u_{3,1} \\ \vdots \\ u_{N,1} \\ u_{1,2} \\ u_{2,2} \\ u_{3,2} \\ \vdots \\ u_{N,2} \\ \vdots \\ u_{1,j} \\ \vdots \\ u_{i,j} \\ \vdots \\ u_{N,j} \\ \vdots \\ u_{1,N} \\ u_{2,N} \\ u_{3,N} \\ \vdots \\ u_{N,N} \end{bmatrix} = \begin{bmatrix} b_{1,1} \\ b_{2,1} \\ b_{3,1} \\ \vdots \\ b_{N,1} \\ b_{1,2} \\ b_{2,2} \\ b_{3,2} \\ \vdots \\ b_{N,2} \\ \vdots \\ b_{1,j} \\ \vdots \\ b_{i,j} \\ \vdots \\ b_{N,j} \\ \vdots \\ b_{1,N} \\ b_{2,N} \\ b_{3,N} \\ \vdots \\ b_{N,N} \end{bmatrix}$$

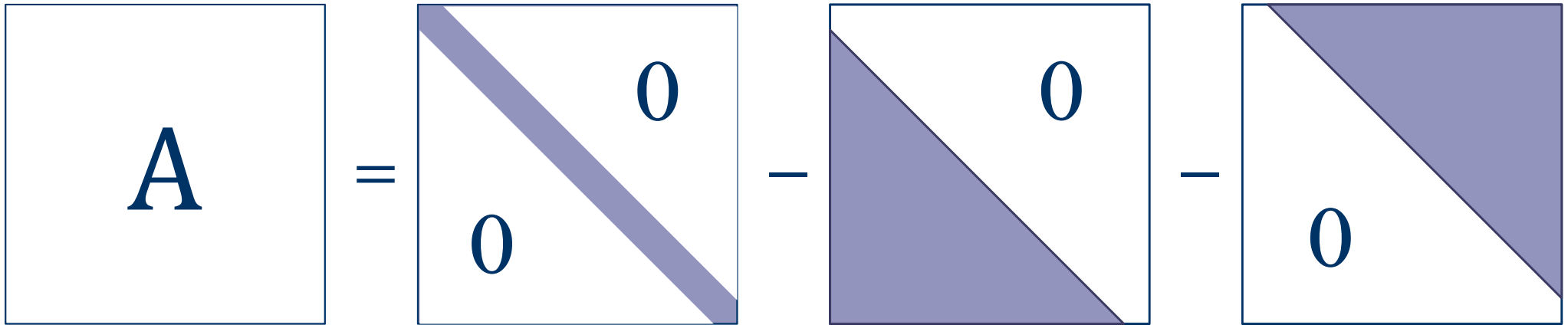
連立一次方程式の反復解法

- 反復解法
  - $\boldsymbol{x}^{(0)}$を真の解$\boldsymbol{x}^{*}$の近似値とし，反復ベクトルの系列 $\boldsymbol{x}^{(0)} \rightarrow \boldsymbol{x}^{(1)} \rightarrow \boldsymbol{x}^{(2)} \rightarrow \cdots \rightarrow \boldsymbol{x}^{(m)}$により，真の解に近づける方法
  - 反復ベクトルの系列の作り方によりいろいろな解法がある．

  - 定常反復法
    - ヤコビ法（Jacobi法）
    - ガウス・ザイデル法（Gauss-Seidel法）
    - 逐次過大緩和法（Successive Over-Relaxation：SOR法）
    - 交互方向法（Alternating-Direction Implicit Iterative Method: ADI法）など

  - 非定常反復法
    - 共役勾配法（Conjugate Gradient Method，CG法）
    - 双共役勾配法（Bi-CG法）
    - 一般化最小残差法（GMRES法）など

行列分離



行列分離： $\mathrm{A} = \mathrm{D} - \mathrm{E} - \mathrm{F}$

$\mathrm{D} = \mathrm{diag}(a_{11}, a_{22}, \cdots, a_{nn})$：対角行列

E：狭義下三角行列

F：狭義上三角行列

$$\mathrm{A}\boldsymbol{x} = \boldsymbol{b} \quad \Rightarrow \quad (\mathrm{D} - \mathrm{E} - \mathrm{F})\boldsymbol{x} = \boldsymbol{b}$$

ヤコビ法（その1）

$$(\mathrm{D}-\mathrm{E}-\mathrm{F})\boldsymbol{x}=\boldsymbol{b} \quad \Rightarrow \quad \mathrm{D}\boldsymbol{x}=(\mathrm{E}+\mathrm{F})\boldsymbol{x}+\boldsymbol{b}$$

$$\boldsymbol{x}^{(m+1)} = \underbrace{\mathrm{D}^{-1}(\mathrm{E}+\mathrm{F})}_{\text{ヤコビ行列}}\boldsymbol{x}^{(m)} + \mathrm{D}^{-1}\boldsymbol{b} \quad (m \geq 0)$$

で，反復ベクトル系列を生成する.

$$x_i^{(m+1)} = \left(-\sum_{j=1, j\neq i}^{n} a_{ij} x_j^{(m)} + b_i \right) / a_{ii}$$

$$(1 \leq i \leq n, m \geq 0)$$

$$\begin{matrix} x_1^{(m)} \\ x_2^{(m)} \\ x_3^{(m)} \\ \vdots \\ \vdots \\ x_{n-1}^{(m)} \\ x_n^{(m)} \end{matrix} \Rightarrow \begin{matrix} x_1^{(m+1)} \\ x_2^{(m+1)} \\ x_3^{(m+1)} \\ \vdots \\ \vdots \\ x_{n-1}^{(m+1)} \\ x_n^{(m+1)} \end{matrix}$$