

有限要素法による  
一次元定常熱伝導解析プログラム  
C言語編

中島 研吾

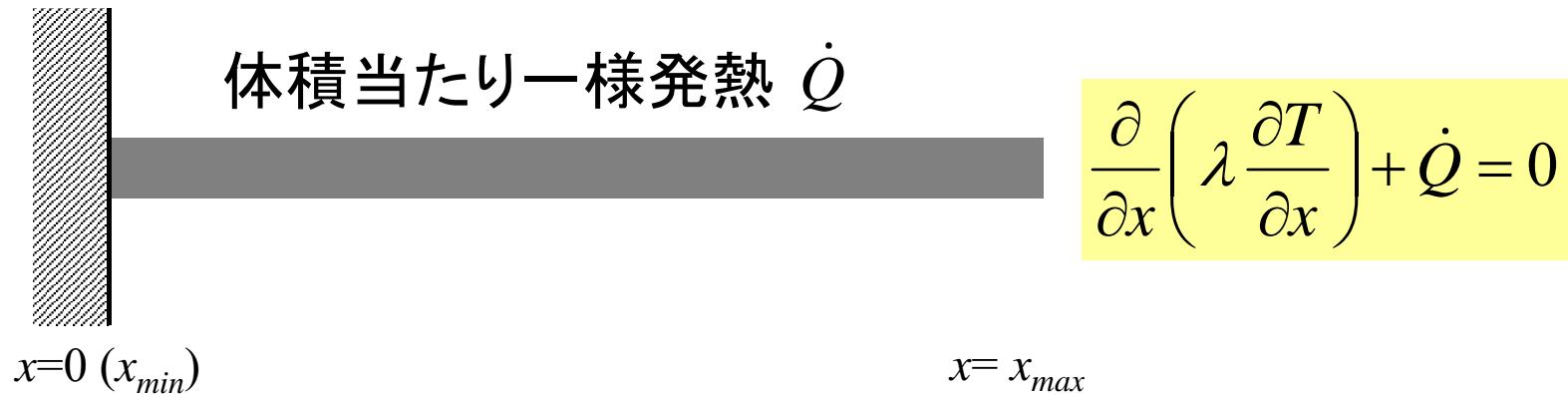
東京大学情報基盤センター

- ガラーキン法による一次元熱伝導問題の解法
- 連立一次方程式の解法
  - 共役勾配法
  - 前処理手法
- 疎行列格納法
- プログラムの内容

# キーワード

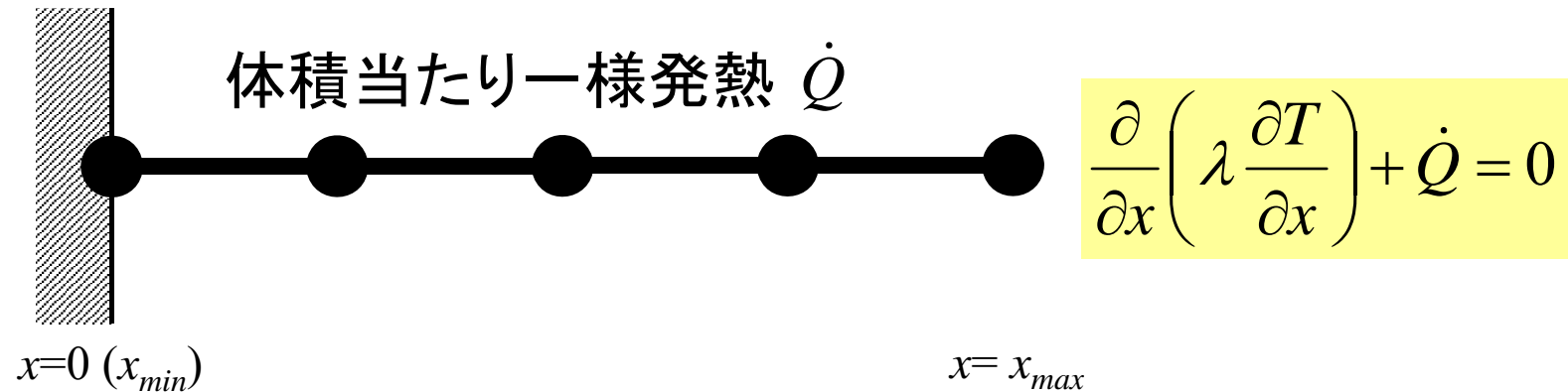
- 一次元熱伝導問題
- ガラーキン法
- 線形一次要素
- 前処理付共役勾配法

# 対象とする問題：一次元熱伝導問題



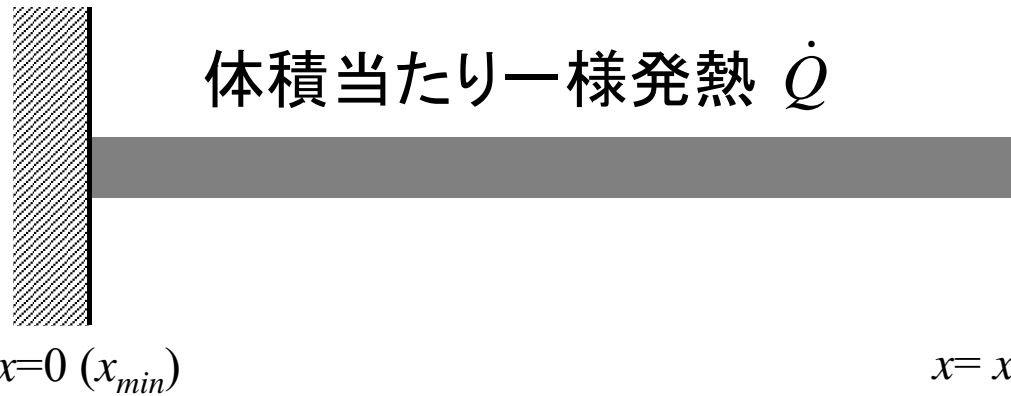
- 一様な：断面積 $A$ ，熱伝導率 $\lambda$
- 体積当たり一様発熱（時間当たり） $[QL^{-3}T^{-1}]$   $\dot{Q}$
- 境界条件
  - $x=0$  :  $T=0$ （固定）
  - $x=x_{max}$  :  $\frac{\partial T}{\partial x} = 0$ （断熱）

# 対象とする問題：一次元熱伝導問題



- 一様な：断面積 $A$ ，熱伝導率 $\lambda$
- 体積当たり一様発熱（時間当たり） $[QL^{-3}T^{-1}]$   $\dot{Q}$
- 境界条件
  - $x=0$  :  $T=0$ （固定）
  - $x=x_{max}$  :  $\frac{\partial T}{\partial x} = 0$ （断熱）

# 解析解



$$\frac{\partial}{\partial x} \left( \lambda \frac{\partial T}{\partial x} \right) + \dot{Q} = 0$$

$$T = 0 @ x = 0$$

$$\frac{\partial T}{\partial x} = 0 @ x = x_{max}$$

$$\lambda T'' = -\dot{Q}$$

$$\lambda T' = -\dot{Q}x + C_1 \Rightarrow C_1 = \dot{Q}x_{max}, \quad T' = 0 @ x = x_{max}$$

$$\lambda T = -\frac{1}{2}\dot{Q}x^2 + C_1x + C_2 \Rightarrow C_2 = 0, \quad T = 0 @ x = 0$$

$$\therefore T = -\frac{1}{2\lambda}\dot{Q}x^2 + \frac{\dot{Q}x_{max}}{\lambda}x$$

# 一次元線形要素 (1/4)

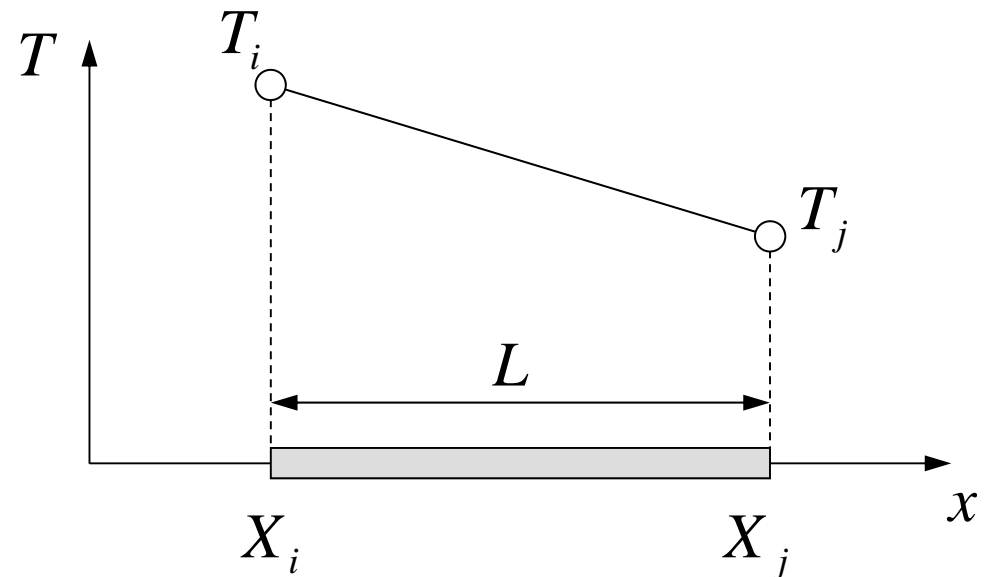
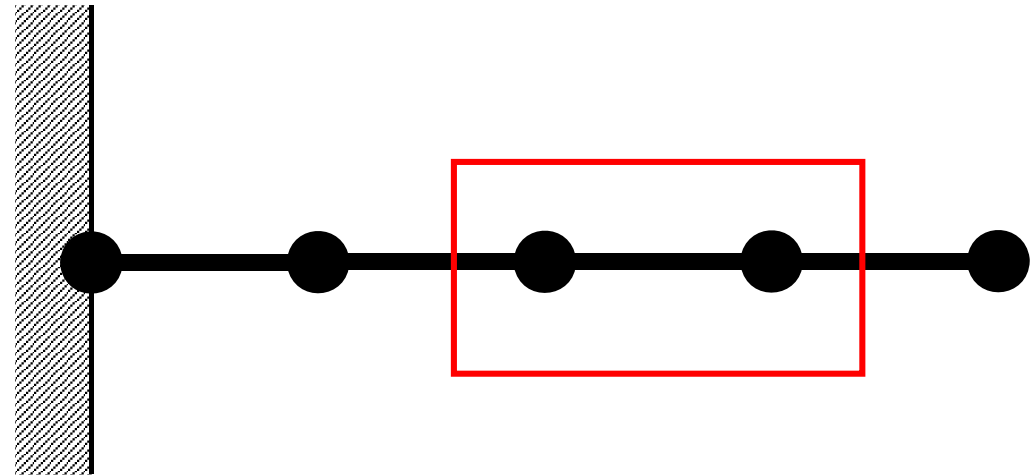
- 一次元線形要素

- 長さ $L$ の両端に節点 (node) を持つ線分

- 節点 : node
- 要素 : element

- 節点  $i, j$  における温度を  $T_i, T_j$
- 要素内での温度 $T$ は以下のように表される (座標 $x$ の一次関数, Piecewise Linear) :

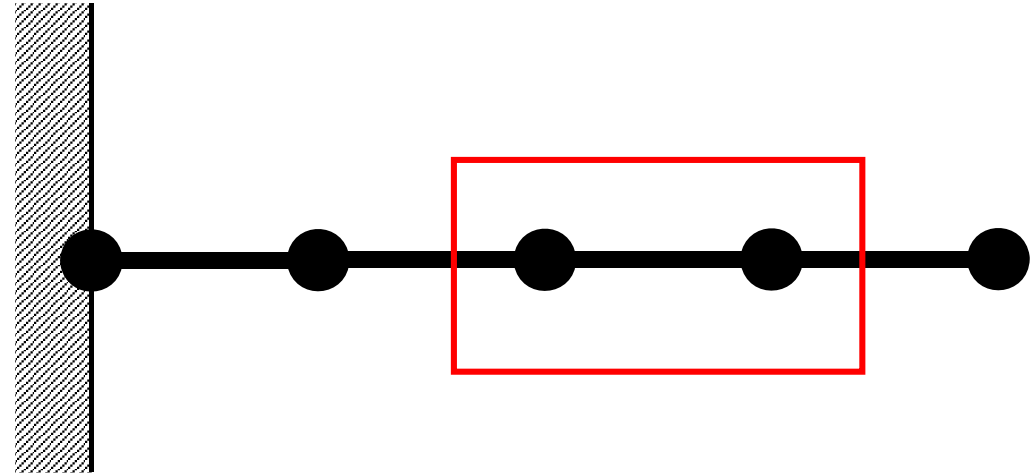
$$T = \alpha_1 + \alpha_2 x$$



# 一次元線形要素 (1/4)

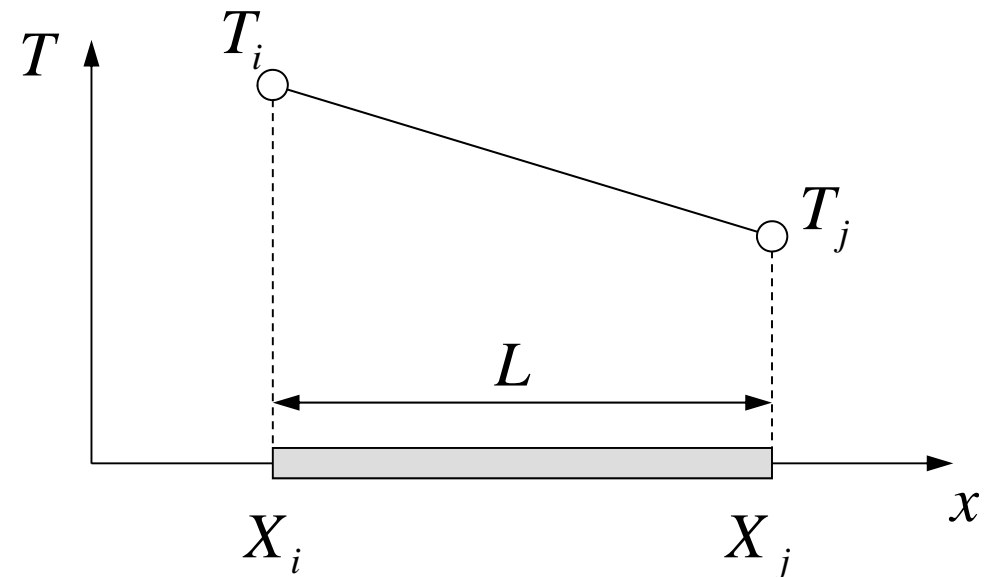
## 一次元線形要素

- 長さ $L$ の両端に節点 (node) を持つ線分
  - 節点 : node
  - 要素 : element



- 節点  $i, j$  における温度を  $\phi_i, \phi_j$
- 要素内での温度  $T$  は以下のように表される (座標  $x$  の一次関数, Piecewise Linear) :

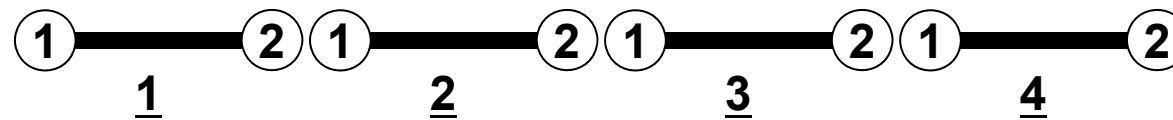
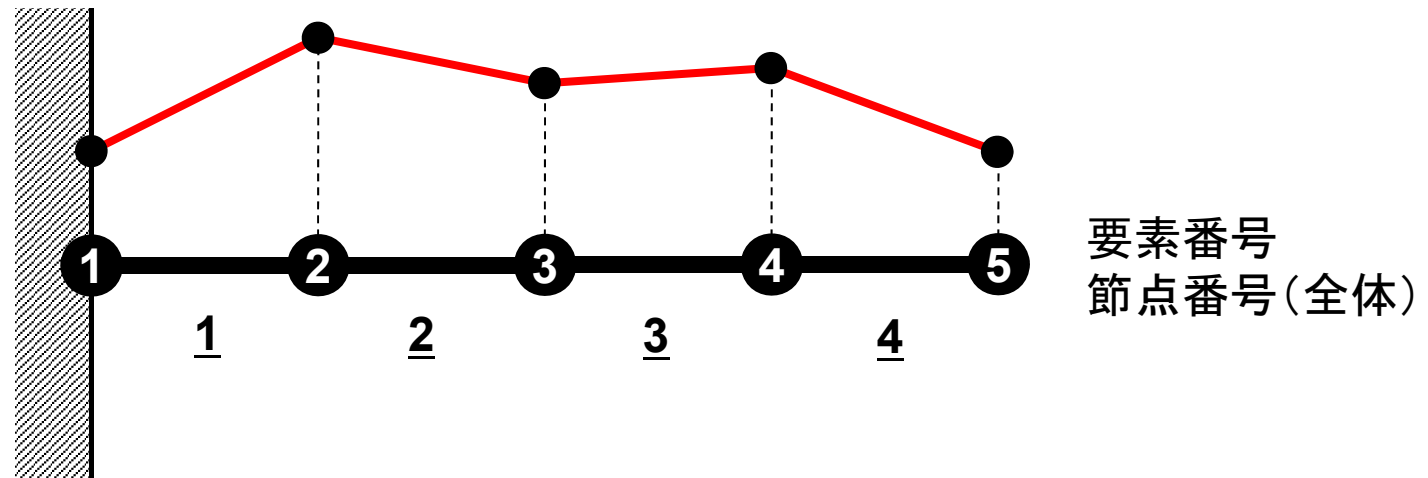
$$T = \alpha_1 + \alpha_2 x$$





# Piecewise Linear

## 各要素内で「温度 $T$ の分布」が線形



各要素における  
「局所」節点番号

温度勾配は要素内で一定  
(節点で不連続となる可能性あり)

# 一次元線形要素：形状関数（2/4）

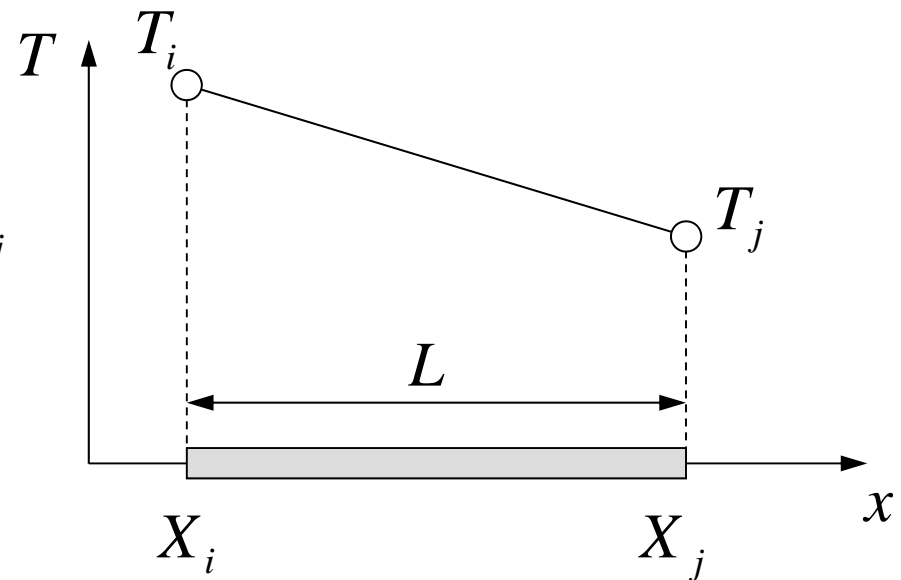
- 節点での条件から，係数は以下のように求められる：

$$T = T_i @ x = X_i, \quad T = T_j @ x = X_j$$

$$T_i = \alpha_1 + \alpha_2 X_i, \quad T_j = \alpha_1 + \alpha_2 X_j$$

- 従って：

$$\alpha_1 = \frac{T_i X_j - T_j X_i}{L}, \quad \alpha_2 = \frac{T_j - T_i}{L}$$



- 元の式に代入して，書き直すと以下のようなになる

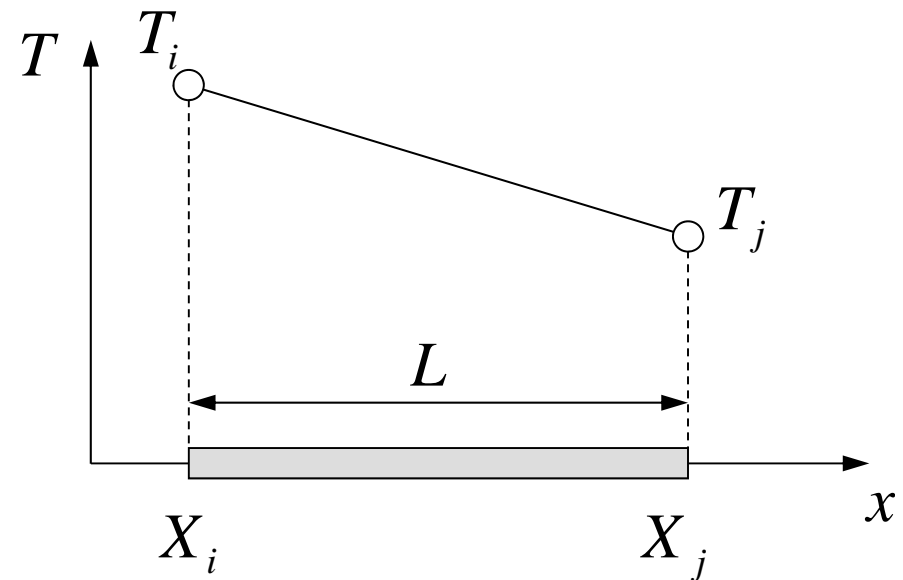
$$T = \underbrace{\left( \frac{X_j - x}{L} \right)}_{N_i} T_i + \underbrace{\left( \frac{x - X_i}{L} \right)}_{N_j} T_j$$

これらのxに関する一次式を形状関数 (shape function) または内挿関数 (interpolation function) と呼ぶ ( $N_i$ ,  $N_j$ と表す)

# 一次元線形要素：形状関数（3/4）

- 形状関数 $N_k$ は要素を構成する節点数と同じ数だけ存在する：
  - 位置座標のみの関数である
  - 「試行関数」の一種

$$N_i = \left( \frac{X_j - x}{L} \right), \quad N_j = \left( \frac{x - X_i}{L} \right)$$



- 形状関数の一次結合により要素内の温度を表す
  - 係数（=未知数）が節点における温度

$$T = N_i T_i + N_j T_j \longleftrightarrow$$

$$T_M = \sum_{i=1}^M a_i \Psi_i$$

$\Psi_i$  領域, 境界において定義される, 位置座標のみ既知関数, 互いに独立である: 試行関数 (trial/test function) と呼ばれる。線形代数における基底 (basis) に相当する

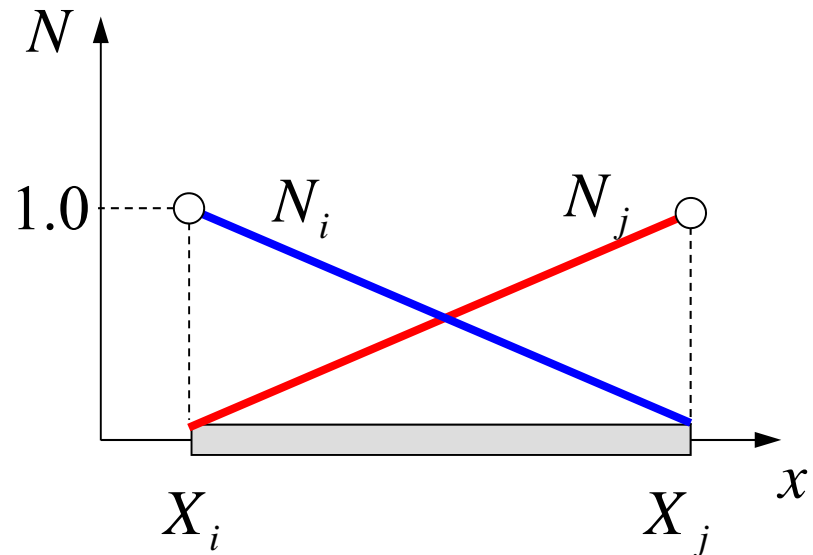
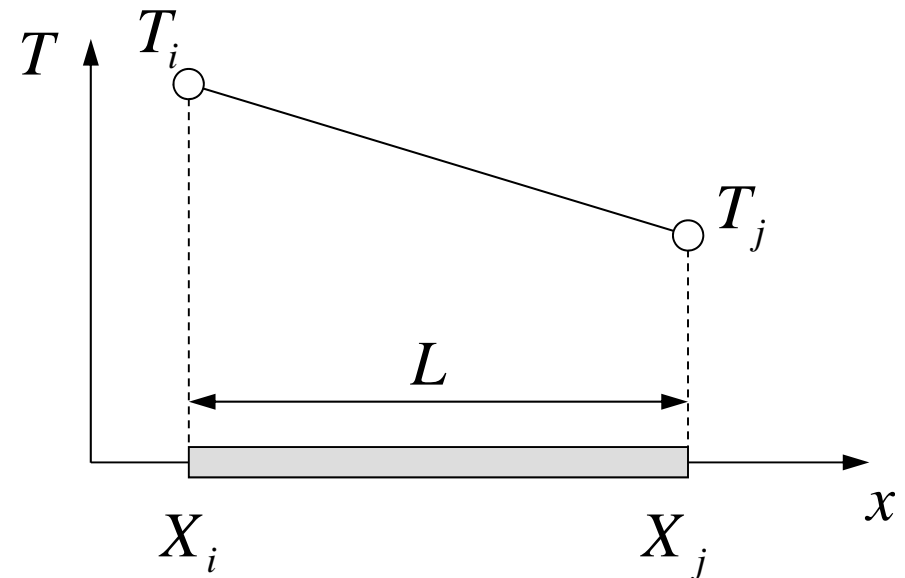
$a_i$  係数 (未知数)

# 一次元線形要素：形状関数（4/4）

- 形状関数はある節点で1の値をとり，他の節点では必ず0の値をとる：

$$N_i = \left( \frac{X_j - x}{L} \right), \quad N_j = \left( \frac{x - X_i}{L} \right)$$

確認してみよう



# ガラーキン法の適用 (1/4)

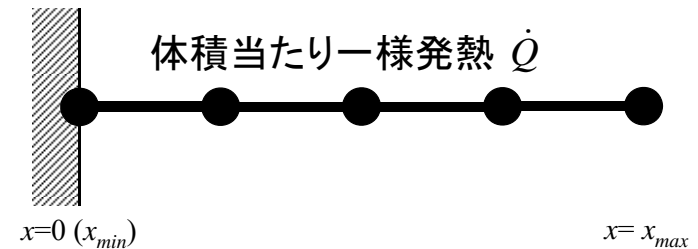
- 以下のような一次元熱伝導方程式を考慮する（熱伝導率一定）：

$$\lambda \left( \frac{d^2 T}{dx^2} \right) + \dot{Q} = 0$$

$T = [N]\{\phi\}$  要素内の温度分布  
 (マトリクス形式), 節点における温度を  $\phi$  としてある。

- ガラーキン法に従い, 重み関数を  $[N]$  とすると, 各要素において以下の積分方程式が得られる:

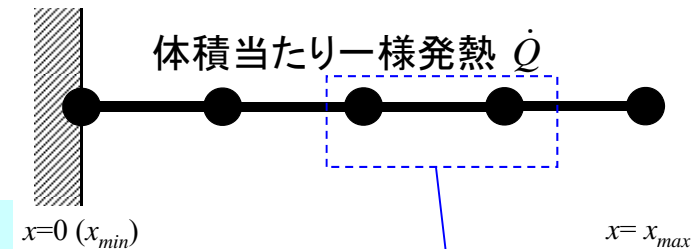
$$\int_V [N]^T \left\{ \lambda \left( \frac{d^2 T}{dx^2} \right) + \dot{Q} \right\} dV = 0$$



# ガラーキン法の適用 (2/4)

- 一次元のグリーンの定理

$$\int_V A \left( \frac{d^2 B}{dx^2} \right) dV = \int_S A \frac{dB}{dx} dS - \int_V \left( \frac{dA}{dx} \frac{dB}{dx} \right) dV$$

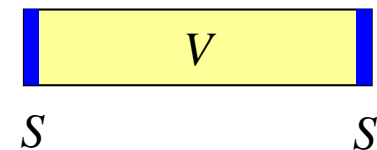


- これを前式の2階微分の部分に適用すると :

$$\int_V \lambda [N]^T \left( \frac{d^2 T}{dx^2} \right) dV = - \int_V \lambda \left( \frac{d[N]^T}{dx} \frac{dT}{dx} \right) dV + \int_S \lambda [N]^T \frac{dT}{dx} dS$$

- これに以下を代入する :

$$T = [N] \{ \phi \}, \quad \frac{dT}{dx} = \frac{d[N]}{dx} \{ \phi \} \quad \bar{q} = -\lambda \frac{dT}{dx}$$



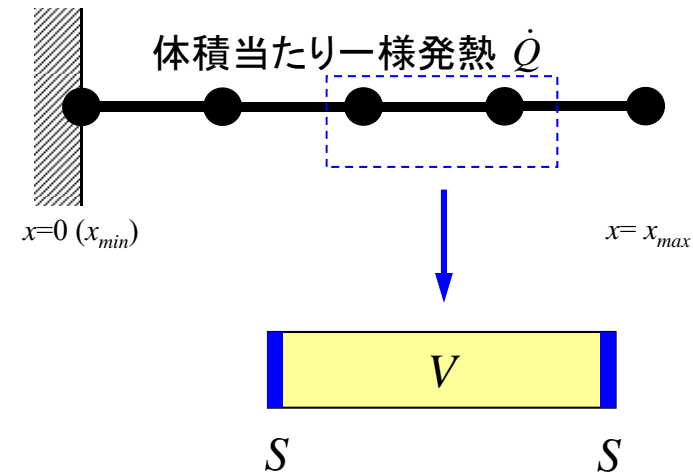
: 要素表面熱流量 [QL<sup>-2</sup>T<sup>-1</sup>]

# ガラーキン法の適用 (3/4)

- 更に体積あたり発熱量の項  $\dot{Q}$  を加えて次式が得られる：

$$-\int_V \lambda \left( \frac{d[N]^T}{dx} \frac{d[N]}{dx} \right) dV \cdot \{\phi\}$$

$$-\int_S \bar{q}[N]^T dS + \int_V Q[N]^T dV = 0$$

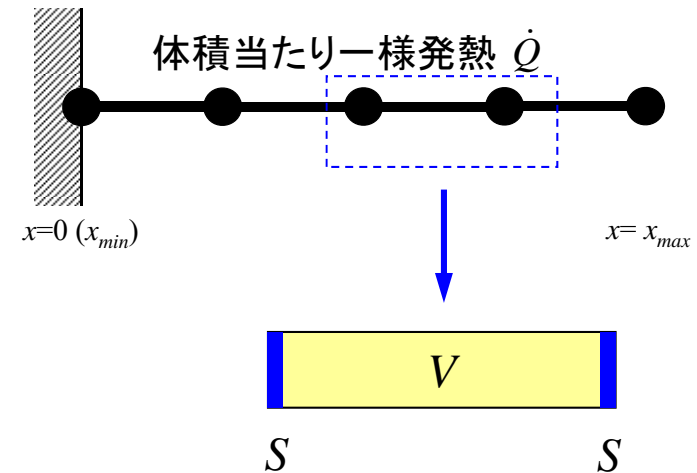


- この式を弱形式 (weak form) と呼ぶ。元の微分方程式では2階の微分が含まれていたが、上式では、グリーンの定理によって1階微分に低減されている。
  - 弱形式によって近似関数 (形状関数, 内挿関数) に対する要求が弱くなっている：すなわち線形関数で2階微分の効果を記述できる。

# ガラーキン法の適用 (4/4)

$$-\int_V \lambda \left( \frac{d[N]^T}{dx} \frac{d[N]}{dx} \right) dV \cdot \{\phi\}$$

$$\boxed{-\int_S \bar{q}[N]^T dS} + \int_V \dot{Q}[N]^T dV = 0$$

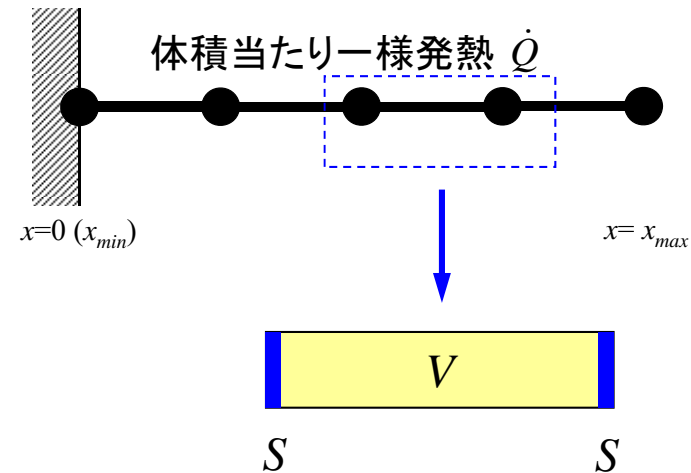


- この項は要素境界で相殺するため、領域境界における項のみが残る。



# 弱形式と境界条件

- 未知数の値が直接与えられる (Dirichlet)
  - 重み関数=0となる
  - 第一種境界条件
  - 基本境界条件
    - essential boundary condition
- 未知数の導関数が与えられる (Neumann)
  - 弱形式中で自然に考慮される
  - 第二種境界条件
  - 自然境界条件
    - natural boundary condition



$$-\int_V \lambda \left( \frac{d[N]^T}{dx} \frac{d[N]}{dx} \right) dV \cdot \{\phi\}$$

$$-\int_S \bar{q} [N]^T dS + \int_V \dot{Q} [N]^T dV = 0$$

$$\bar{q} = -\lambda \frac{dT}{dx} \text{ から得られる}$$

# 境界条件を考慮した弱形式：各要素

$$[k]^{(e)} \{\phi\}^{(e)} = \{f\}^{(e)}$$

$$[k]^{(e)} = \int_V \lambda \left( \frac{d[N]^T}{dx} \frac{d[N]}{dx} \right) dV$$

$$\{f\}^{(e)} = \int_V \dot{Q} [N]^T dV - \int_S \bar{q} [N]^T dS$$

# 要素単位での積分：[k]

$$N_i = \left( \frac{X_j - x}{L} \right), \quad N_j = \left( \frac{x - X_i}{L} \right)$$

$$\frac{dN_i}{dx} = \left( \frac{-1}{L} \right), \quad \frac{dN_j}{dx} = \left( \frac{1}{L} \right)$$

$$\int_V \lambda \left( \frac{d[N]^T}{dx} \frac{d[N]}{dx} \right) dV$$

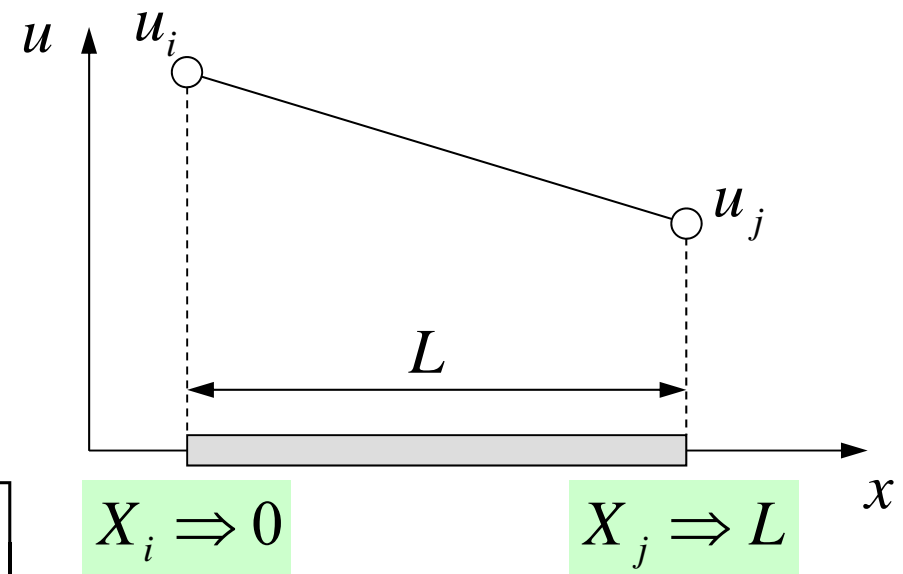
$$= \lambda \int_0^L \begin{bmatrix} -1/L \\ 1/L \end{bmatrix} [-1/L, 1/L] A dx$$

2x1 matrix

1x2 matrix

$$= \frac{\lambda A}{L^2} \int_0^L \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} dx = \frac{\lambda A}{L} \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}$$

A: 断面積, L: 要素長さ



$$N_i = \left( 1 - \frac{x}{L} \right), \quad N_j = \left( \frac{x}{L} \right)$$

# 要素単位での積分： $\{f\}$ (1/2)

$$N_i = \left( \frac{X_j - x}{L} \right), \quad N_j = \left( \frac{x - X_i}{L} \right) \quad \frac{dN_i}{dx} = \left( \frac{-1}{L} \right), \quad \frac{dN_j}{dx} = \left( \frac{1}{L} \right)$$

$$N_i = \left( 1 - \frac{x}{L} \right), \quad N_j = \left( \frac{x}{L} \right)$$

$$\int_V \dot{Q} [N]^T dV = \dot{Q} A \int_0^L \begin{bmatrix} 1 - x/L \\ x/L \end{bmatrix} dx = \frac{\dot{Q} A L}{2} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$$

体積当たり発熱



$A$ : 断面積,  $L$ : 要素長さ

# 要素単位での積分： $\{f\}$ (2/2)

$$N_i = \left( \frac{X_j - x}{L} \right), \quad N_j = \left( \frac{x - X_i}{L} \right) \quad \frac{dN_i}{dx} = \left( \frac{-1}{L} \right), \quad \frac{dN_j}{dx} = \left( \frac{1}{L} \right)$$

$$\int_V \dot{Q} [N]^T dV = \dot{Q} A \int_0^L \begin{bmatrix} 1 - x/L \\ x/L \end{bmatrix} dx = \frac{\dot{Q} A L}{2} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$$

体積当たり発熱

$$\int_S \bar{q} [N]^T dS = \bar{q} A \Big|_{x=L} = \bar{q} A \begin{Bmatrix} 0 \\ 1 \end{Bmatrix}, \quad \bar{q} = -\lambda \frac{dT}{dx}$$

表面熱流束

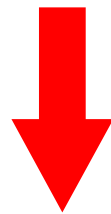


表面熱流束がこの断面のみに作用しているとする

# 全体方程式

- 要素単位の方程式を全体で足し合わせ,

$$[k]^{(e)} \{\phi\}^{(e)} = \{f\}^{(e)} \quad \text{要素マトリクス, 要素方程式}$$



$$[K] \cdot \{\Phi\} = \{F\} \quad \text{全体マトリクス, 全体方程式}$$

$$[K] = \sum [k], \quad \{F\} = \sum \{f\}$$

$$\{\Phi\}: \text{global vector of } \{\phi\}$$

この連立一次方程式(全体方程式)  
を解いてやればよい

# ファイル準備 on PC

コピー, 展開

<http://nkl.cc.u-tokyo.ac.jp/files/fem-c.tar>

<http://nkl.cc.u-tokyo.ac.jp/files/fem-f.tar>

```
>$ cd <$CUR>
```

```
>$ tar xvf fem-c.tar
```

```
>$ tar xvf fem-f.tar
```

```
>$ cd fem-c or cd fem-f
```

以下のディレクトリが出来ていることを確認

1D fem3D

これらを以降 `<$P-TOP>/1d`, `<$P-TOP>/fem3D`

Your PC

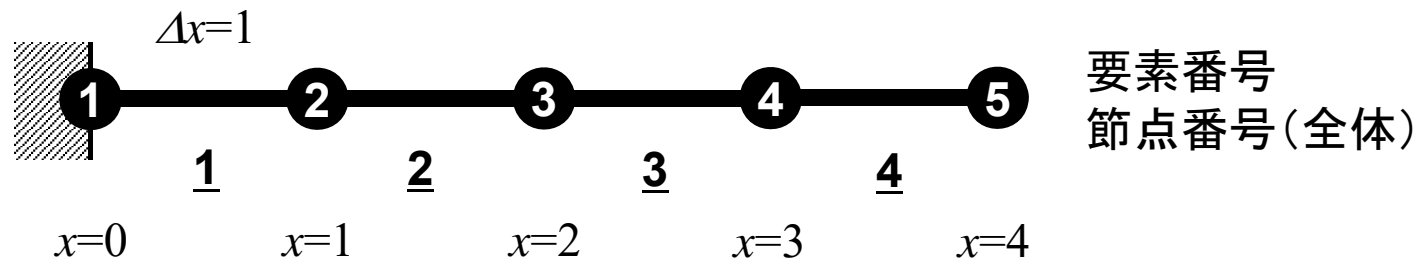
FX10

# 実行 (Cygwinではa.exe)

```
>$ cd <${P-TOP}>/1d
>$ cc -O 1d.c          (or gfotran -O 1d.f)
>$ ./a.out
```

制御ファイル input.dat

4	NE (要素数)
1.0 1.0 1.0 1.0	$\Delta x$ (要素長さL), Q, A, $\lambda$
100	反復回数 (CG法後述)
1.e-8	CG法の反復打切誤差





# 結果

```
>$ ./a.out
```

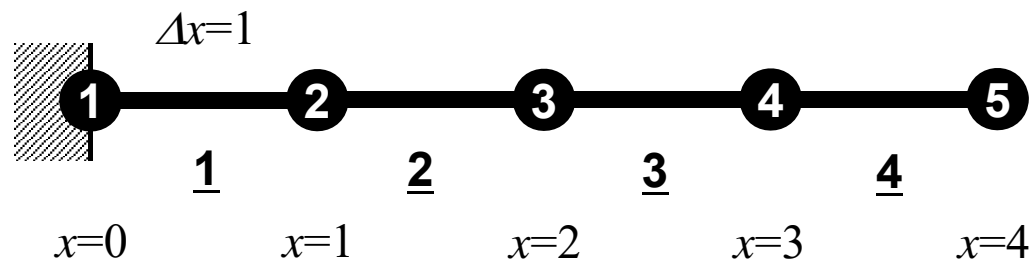
```
4 iters, RESID= 4.154074e-17
```

```
### TEMPERATURE
```

1	0.000000E+00	0.000000E+00
2	3.500000E+00	3.500000E+00
3	6.000000E+00	6.000000E+00
4	7.500000E+00	7.500000E+00
5	8.000000E+00	8.000000E+00

計算結果

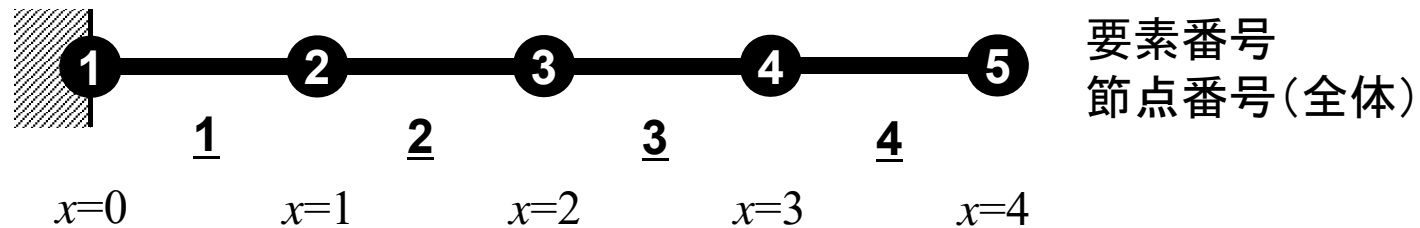
解析解



要素番号  
節点番号(全体)

# 要素方程式とその重ね合わせ (1/3)

- 4要素, 5節点の例題



- 要素1の $[k], \{f\}$  は以下のようなになる :

$$[k]^{(1)} = \frac{\lambda A}{L} \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \quad \{f\}^{(1)} = \frac{\dot{Q}AL}{2} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$$

- 要素4については :

$$[k]^{(4)} = \frac{\lambda A}{L} \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \quad \{f\}^{(4)} = \frac{\dot{Q}AL}{2} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$$

# 要素方程式とその重ね合わせ (2/3)

- これを順番に足していけばよい

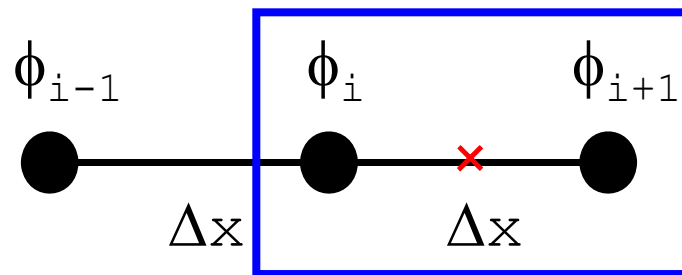
$$[K] = \sum_{e=1}^4 [k]^{(e)} =$$

$$\{F\} = \sum_{e=1}^4 \{f\}^{(e)} =$$



# 差分法における二階微分係数

- $\times$  ( $i$ と $i+1$ の中点)における微分係数



$$\left(\frac{d\phi}{dx}\right)_{i+1/2} \approx \frac{\phi_{i+1} - \phi_i}{\Delta x}$$

$\Delta x \rightarrow 0$  となると微分係数の定義そのもの

- $i$ における二階微分係数

$$\left(\frac{d^2\phi}{dx^2}\right)_i \approx \frac{\left(\frac{d\phi}{dx}\right)_{i+1/2} - \left(\frac{d\phi}{dx}\right)_{i-1/2}}{\Delta x} = \frac{\frac{\phi_{i+1} - \phi_i}{\Delta x} - \frac{\phi_i - \phi_{i-1}}{\Delta x}}{\Delta x} = \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2}$$

要素ごとに物性，寸法が  
異なっても簡単に対応が可能

$$[k]^{(e)} = \frac{\lambda^{(e)} A^{(e)}}{L^{(e)}} \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}$$

$$[K] = \sum_{e=1}^4 [k^{(e)}] =$$

$$\begin{array}{c} \begin{array}{cc} +1 & -1 \\ -1 & +1 \end{array} \times \frac{\lambda^{(1)} A^{(1)}}{L^{(1)}} + \begin{array}{cc} +1 & -1 \\ -1 & +1 \end{array} \times \frac{\lambda^{(2)} A^{(2)}}{L^{(2)}} \\ \\ \begin{array}{cc} +1 & -1 \\ -1 & +1 \end{array} \times \frac{\lambda^{(3)} A^{(3)}}{L^{(3)}} + \begin{array}{cc} +1 & -1 \\ -1 & +1 \end{array} \times \frac{\lambda^{(4)} A^{(4)}}{L^{(4)}} \end{array}$$







- ガラーキン法による一次元弾性問題の解法
- 連立一次方程式の解法
  - 共役勾配法
  - 前処理手法
- 疎行列格納法
- プログラムの内容

# あとは出てきた全体方程式 (連立一次方程式)を解けばよい

- 多くの科学技術計算は、最終的に大規模線形方程式  $Ax=b$  を解くことに帰着される。
- 様々な手法が提案されている
  - 疎行列 (sparse), 密行列 (dense)
  - 直接法 (direct), 反復法 (iterative)
- 密行列 (dense)
  - 境界要素法, スペクトル法など
- 疎行列 (sparse): 0の部分が多い
  - FEM, FDMなど

# 直接法 (Direct Method)

- Gaussの消去法, 完全LU分解
  - 逆行列 $A^{-1}$ を直接求める
- 利点
  - 安定, 幅広いアプリケーションに適用可能
    - Partial Pivoting
  - 疎行列, 密行列いずれにも適用可能
- 欠点
  - 反復法よりもメモリ, 計算時間を必要とする
    - 密行列の場合,  $O(N^3)$ の計算量
  - 大規模な計算向けではない
    - $O(N^2)$ の記憶容量,  $O(N^3)$ の計算量

# 反復法 (Iterative Method) とは?

Linear Equations  
連立一次方程式

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

**A**                      **x**                      **b**

Initial Solution  
初期解

$$\mathbf{x}^{(0)} = \begin{pmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \vdots \\ x_n^{(0)} \end{pmatrix}$$

Starting from a initial vector  $\mathbf{x}^{(0)}$ , iterative method obtains the final converged solutions by iterations

$$\mathbf{X}^{(1)}, \mathbf{X}^{(2)}, \dots$$

# 反復法 (Iterative Method)

- 定常 (stationary) 法

- 反復計算中, 解ベクトル以外の変数は変化せず
- SOR, Gauss-Seidel, Jacobiなど
- 概して遅い

$$\mathbf{Ax} = \mathbf{b} \Rightarrow$$

$$\mathbf{x}^{(k+1)} = \mathbf{M}\mathbf{x}^{(k)} + \mathbf{N}\mathbf{b}$$

- 非定常 (nonstationary) 法

- 拘束, 最適化条件が加わる
- Krylov部分空間 (subspace) への写像を基底として使用するため, Krylov部分空間法とも呼ばれる
- CG (Conjugate Gradient: 共役勾配法)
- BiCGSTAB (Bi-Conjugate Gradient Stabilized)
- GMRES (Generalized Minimal Residual)

# 反復法 (Iterative Method) (続き)

- 利点
  - 直接法と比較して、メモリ使用量、計算量が少ない。
  - 並列計算には適している。
- 欠点
  - 収束性が、アプリケーション、境界条件の影響を受けやすい。
  - 前処理 (preconditioning) が重要。

# 非定常反復法: クリロフ部分空間法 (1/2)

## Krylov Subspace Method

$$\mathbf{Ax} = \mathbf{b} \Rightarrow \mathbf{x} = \mathbf{b} + (\mathbf{I} - \mathbf{A})\mathbf{x}$$

以下の反復式を導入し  $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k$  を求める:

$$\begin{aligned} \mathbf{x}_k &= \mathbf{b} + (\mathbf{I} - \mathbf{A})\mathbf{x}_{k-1} \\ &= (\mathbf{b} - \mathbf{Ax}_{k-1}) + \mathbf{x}_{k-1} \\ &= \mathbf{r}_{k-1} + \mathbf{x}_{k-1} \end{aligned}$$

where  $\mathbf{r}_k = \mathbf{b} - \mathbf{Ax}_k$ : 残差ベクトル  
(residual)



$$\mathbf{x}_k = \mathbf{x}_0 + \sum_{i=0}^{k-1} \mathbf{r}_i$$

$$\begin{aligned} \mathbf{r}_k &= \mathbf{b} - \mathbf{Ax}_k = \mathbf{b} - \mathbf{A}(\mathbf{r}_{k-1} + \mathbf{x}_{k-1}) \\ &= (\mathbf{b} - \mathbf{Ax}_{k-1}) - \mathbf{Ar}_{k-1} = \mathbf{r}_{k-1} - \mathbf{Ar}_{k-1} = (\mathbf{I} - \mathbf{A})\mathbf{r}_{k-1} \end{aligned}$$

# 非定常反復法: クリロフ部分空間法 (2/2)

## Krylov Subspace Method

$$\mathbf{x}_k = \mathbf{x}_0 + \sum_{i=0}^{k-1} \mathbf{r}_i = \mathbf{x}_0 + \mathbf{r}_0 + \sum_{i=0}^{k-2} (\mathbf{I} - \mathbf{A})\mathbf{r}_i = \mathbf{x}_0 + \mathbf{r}_0 + \sum_{i=1}^{k-1} (\mathbf{I} - \mathbf{A})^i \mathbf{r}_0$$

$$\mathbf{z}_k = \mathbf{r}_0 + \sum_{i=1}^{k-1} (\mathbf{I} - \mathbf{A})^i \mathbf{r}_0 = \left[ \mathbf{I} + \sum_{i=1}^{k-1} (\mathbf{I} - \mathbf{A})^i \right] \mathbf{r}_0$$



$\mathbf{z}_k$ はk次のクリロフ部分空間(Krylov Subspace)に属するベクトル, 問題はクリロフ部分空間からどのようにして解の近似ベクトル $\mathbf{x}_k$ を求めるかにある:

$$\left[ \mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \dots, \mathbf{A}^{k-1}\mathbf{r}_0 \right]$$



# 代表的な反復法：共役勾配法

- Conjugate Gradient法, 略して「CG」法
  - 最も代表的な「非定常」反復法
- 対称正定値行列 (Symmetric Positive Definite: SPD)
  - 任意のベクトル  $\{x\}$  に対して  $\{x\}^T[A]\{x\} > 0$
  - 全対角成分  $> 0$ , 全固有値  $> 0$ , 全部分行列式  $> 0$  と同値
  - (ガラーキソ法) 熱伝導, 弾性, ねじり: 本コードの場合も SPD
- アルゴリズム
  - 最急降下法 (Steepest Descent Method) の変種
  - $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
    - $x^{(i)}$ : 反復解,  $p^{(i)}$ : 探索方向,  $\alpha_i$ : 定数
  - 厳密解を  $y$  とするとき  $\{x-y\}^T[A]\{x-y\}$  を最小とするような  $\{x\}$  を求める。
  - 詳細は参考文献参照
    - 例えば: 森正武「数値解析(第2版)」(共立出版)

$$\det \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & \cdots & a_{3n} \\ a_{41} & a_{42} & a_{43} & a_{44} & \cdots & a_{4n} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{nm} \end{bmatrix}$$

# 共役勾配法のアルゴリズム

```
Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
   $z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} \cdot z^{(i-1)}$ 
  if  $i = 1$ 
     $p^{(1)} = z^{(0)}$ 
  else
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
  endif
   $q^{(i)} = [A]p^{(i)}$ 
   $\alpha_i = \rho_{i-1} / p^{(i)} \cdot q^{(i)}$ 
   $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
   $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
  check convergence  $|r|$ 
end
```

- 行列ベクトル積
- ベクトル内積
- ベクトル定数倍の加減

$x^{(i)}$  : ベクトル

$\alpha_i$  : スカラー

# 共役勾配法のアルゴリズム

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
   $z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} \cdot z^{(i-1)}$ 
  if  $i = 1$ 
     $p^{(1)} = z^{(0)}$ 
  else
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
  endif
   $q^{(i)} = [A]p^{(i)}$ 
   $\alpha_i = \rho_{i-1} / p^{(i)} \cdot q^{(i)}$ 
   $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
   $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
  check convergence  $|r|$ 
end

```

- 行列ベクトル積
- ベクトル内積
- ベクトル定数倍の加減

$x^{(i)}$  : ベクトル

$\alpha_i$  : スカラー

# 共役勾配法のアルゴリズム

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
   $z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
  if  $i=1$ 
     $p^{(1)} = z^{(0)}$ 
  else
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
  endif
   $q^{(i)} = [A]p^{(i)}$ 
   $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$ 
   $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
   $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
  check convergence  $|r|$ 
end

```

- 行列ベクトル積
- ベクトル内積
- ベクトル定数倍の加減

$x^{(i)}$  : ベクトル

$\alpha_i$  : スカラー

# 共役勾配法のアルゴリズム

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
   $z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} \cdot z^{(i-1)}$ 
  if  $i = 1$ 
     $p^{(1)} = z^{(0)}$ 
  else
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
  endif
   $q^{(i)} = [A]p^{(i)}$ 
   $\alpha_i = \rho_{i-1} / p^{(i)} \cdot q^{(i)}$ 
   $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
   $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
  check convergence  $|r|$ 
end

```

- 行列ベクトル積
- ベクトル内積
- ベクトル定数倍の加減

$x^{(i)}$  : ベクトル

$\alpha_i$  : スカラー

# 共役勾配法のアルゴリズム

```
Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
   $z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} \cdot z^{(i-1)}$ 
  if  $i = 1$ 
     $p^{(1)} = z^{(0)}$ 
  else
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
  endif
   $q^{(i)} = [A]p^{(i)}$ 
   $\alpha_i = \rho_{i-1} / p^{(i)} \cdot q^{(i)}$ 
   $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
   $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
  check convergence  $|r|$ 
end
```

$x^{(i)}$  : ベクトル  
 $\alpha_i$  : スカラー

# CG法アルゴリズムの導出(1/5)

$y$ を厳密解 ( $Ay=b$ ) とするとき, 下式を最小にする  $x$  を求める:

$$(x-y)^T [A](x-y)$$

$$\begin{aligned} (x-y)^T [A](x-y) &= (x, Ax) - (y, Ax) - (x, Ay) + (y, Ay) \\ &= (x, Ax) - 2(x, Ay) + (y, Ay) = (x, Ax) - 2(x, b) + \underline{(y, b)} \quad \text{定数} \end{aligned}$$

従って, 下記  $f(x)$  を最小にする  $x$  を求めればよい:

$$f(x) = \frac{1}{2}(x, Ax) - (x, b)$$

$$f(x+h) = f(x) + (h, Ax-b) + \frac{1}{2}(h, Ah)$$

任意のベクトル  $h$

$$f(x) = \frac{1}{2}(x, Ax) - (x, b)$$

$$f(x+h) = f(x) + (h, Ax - b) + \frac{1}{2}(h, Ah)$$

•任意のベクトル $h$

$$\begin{aligned} f(x+h) &= \frac{1}{2}(x+h, A(x+h)) - (x+h, b) \\ &= \frac{1}{2}(x+h, Ax) + \frac{1}{2}(x+h, Ah) - (x, b) - (h, b) \\ &= \frac{1}{2}(x, Ax) + \frac{1}{2}(h, Ax) + \frac{1}{2}(x, Ah) + \frac{1}{2}(h, Ah) - (x, b) - (h, b) \\ &= \frac{1}{2}(x, Ax) - (x, b) + (h, Ax) - (h, b) + \frac{1}{2}(h, Ah) \\ &= f(x) + (h, Ax - b) + \frac{1}{2}(h, Ah) \end{aligned}$$



# CG法アルゴリズムの導出(2/5)

CG法は任意の  $x^{(0)}$  から始めて,  $f(x)$  の最小値を逐次探索する。  
今,  $k$  番目の近似値  $x^{(k)}$  と探索方向  $p^{(k)}$  が決まったとすると:

$$x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}$$

$f(x^{(k+1)})$  を最小にするためには:

$$f(x^{(k)} + \alpha_k p^{(k)}) = \frac{1}{2} \alpha_k^2 (p^{(k)}, Ap^{(k)}) - \alpha_k (p^{(k)}, b - Ax^{(k)}) + f(x^{(k)})$$
$$\frac{\partial f(x^{(k)} + \alpha_k p^{(k)})}{\partial \alpha_k} = 0 \Rightarrow \alpha_k = \frac{(p^{(k)}, b - Ax^{(k)})}{(p^{(k)}, Ap^{(k)})} = \frac{(p^{(k)}, r^{(k)})}{(p^{(k)}, Ap^{(k)})} \quad (1)$$

$r^{(k)} = b - Ax^{(k)}$  は第  $k$  近似に対する残差

# CG法アルゴリズムの導出(3/5)

残差  $r^{(k)}$  も以下の式によって計算できる  $r^{(k+1)} = b - Ax^{(k+1)}, r^{(k)} = b - Ax^{(k)}$

$$r^{(k+1)} = r^{(k)} - \alpha_k Ap^{(k)} \quad (2) \qquad r^{(k+1)} - r^{(k)} = Ax^{(k+1)} - Ax^{(k)} = \alpha_k Ap^{(k)}$$

探索方向を以下の漸化式によって求める:

$$p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)}, r^{(0)} = p^{(0)} \quad (3)$$

本当のところは下記のように  $(k+1)$  回目に厳密解  $y$  が求まれば良いのであるが、解がわかっていない場合は困難...

$$y = x^{(k+1)} + \alpha_{k+1} p^{(k+1)}$$

# CG法アルゴリズムの導出(4/5)

ところで、下式のような都合の良い直交関係がある:

$$\left( Ap^{(k)}, y - x^{(k+1)} \right) = 0$$

$$\begin{aligned} \left( Ap^{(k)}, y - x^{(k+1)} \right) &= \left( p^{(k)}, Ay - Ax^{(k+1)} \right) = \left( p^{(k)}, b - Ax^{(k+1)} \right) \\ &= \left( p^{(k)}, b - A[x^{(k)} + \alpha_k p^{(k)}] \right) = \left( p^{(k)}, b - Ax^{(k)} - \alpha_k Ap^{(k)} \right) \\ &= \left( p^{(k)}, r^{(k)} - \alpha_k Ap^{(k)} \right) = \left( p^{(k)}, r^{(k)} \right) - \alpha_k \left( p^{(k)}, Ap^{(k)} \right) = 0 \end{aligned}$$

$$\therefore \alpha_k = \frac{\left( p^{(k)}, r^{(k)} \right)}{\left( p^{(k)}, Ap^{(k)} \right)}$$

従って以下が成立する:

$$\left( Ap^{(k)}, y - x^{(k+1)} \right) = \left( Ap^{(k)}, \alpha_{k+1} p^{(k+1)} \right) = 0 \Rightarrow \left( p^{(k+1)}, Ap^{(k)} \right) = 0$$

# CG法アルゴリズムの導出(5/5)

$$\begin{aligned} (p^{(k+1)}, Ap^{(k)}) &= (r^{(k+1)} + \beta_k p^{(k)}, Ap^{(k)}) = (r^{(k+1)}, Ap^{(k)}) + \beta_k (p^{(k)}, Ap^{(k)}) = 0 \\ \Rightarrow \beta_k &= \frac{-(r^{(k+1)}, Ap^{(k)})}{(p^{(k)}, Ap^{(k)})} \quad (4) \end{aligned}$$

$(p^{(k+1)}, Ap^{(k)}) = 0$   $p^{(k)}$  と  $p^{(k+1)}$  が行列Aに関して共役 (conjugate)

```

Compute  $p^{(0)} = r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
  calc.  $\alpha_{i-1}$ 
   $x^{(i)} = x^{(i-1)} + \alpha_{i-1}p^{(i-1)}$ 
   $r^{(i)} = r^{(i-1)} - \alpha_{i-1}[A]p^{(i-1)}$ 

  check convergence  $|r|$ 
  (if not converged)
  calc.  $\beta_{i-1}$ 
   $p^{(i)} = r^{(i)} + \beta_{i-1}p^{(i-1)}$ 
end

```

$$\alpha_{i-1} = \frac{(p^{(i-1)}, r^{(i-1)})}{(p^{(i-1)}, Ap^{(i-1)})}$$

$$\beta_{i-1} = \frac{-(r^{(i)}, Ap^{(i-1)})}{(p^{(i-1)}, Ap^{(i-1)})}$$

# CG法アルゴリズム

任意の $(i,j)$ に対して以下の共役関係が得られる:

$$(p^{(i)}, Ap^{(j)}) = 0 \quad (i \neq j)$$

探索方向 $p^{(k)}$ , 残差ベクトル $r^{(k)}$ についても以下の関係が成立する:

$$(r^{(i)}, r^{(j)}) = 0 \quad (i \neq j), \quad (p^{(k)}, r^{(k)}) = (r^{(k)}, r^{(k)})$$

N次元空間で互いに直交で一次独立な残差ベクトル $r^{(k)}$ はN個しか存在しない, 従って共役勾配法は未知数がN個のときにN回以内に収束する  $\Rightarrow$  実際は丸め誤差の影響がある(条件数が大きい場合)

## Top 10 Algorithms in the 20<sup>th</sup> Century (SIAM)

<http://www.siam.org/news/news.php?id=637>

モンテカルロ法, シンプレックス法, **クリロフ部分空間法**, 行列分解法, 最適化Fortranコンパイラ, QR法, クイックソート, FFT, 整数関係アルゴリズム, FMM(高速多重極法)

# Proof (1/3)

## Mathematical Induction

### 数学的帰納法

$$\begin{aligned} (r^{(i)}, r^{(j)}) &= 0 \quad (i \neq j) && \text{直交性} \\ (p^{(i)}, Ap^{(j)}) &= 0 \quad (i \neq j) && \text{共役性} \end{aligned}$$

$$(1) \quad \alpha_k = \frac{(p^{(k)}, r^{(k)})}{(p^{(k)}, Ap^{(k)})}$$

$$(2) \quad r^{(k+1)} = r^{(k)} - \alpha_k Ap^{(k)}$$

$$(3) \quad p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)}, \quad r^{(0)} = p^{(0)}$$

$$(4) \quad \beta_k = \frac{-(r^{(k+1)}, Ap^{(k)})}{(p^{(k)}, Ap^{(k)})}$$

# Proof (2/3)

## Mathematical Induction

### 数学的帰納法

$$\begin{aligned} (r^{(i)}, r^{(j)}) &= 0 \quad (i \neq j) \\ (p^{(i)}, Ap^{(j)}) &= 0 \quad (i \neq j) \end{aligned} \quad (*)$$

(\*) is satisfied for  $i \leq k, j \leq k$  where  $i \neq j$

$$\begin{aligned} \text{if } i < k \quad (r^{(k+1)}, r^{(i)}) &= (r^{(i)}, r^{(k+1)}) \stackrel{(2)}{=} (r^{(i)}, r^{(k)} - \alpha_k Ap^{(k)}) \\ &\stackrel{(*)}{=} -\alpha_k (r^{(i)}, Ap^{(k)}) \stackrel{(4)}{=} -\alpha_k (p^{(i)} - \beta_{i-1} p^{(i-1)}, Ap^{(k)}) \\ &= -\alpha_k (p^{(i)}, Ap^{(k)}) + \alpha_k \beta_{i-1} (p^{(i-1)}, Ap^{(k)}) \stackrel{(*)}{=} 0 \end{aligned}$$

$$\begin{aligned} \text{if } i = k \quad (r^{(k+1)}, r^{(k)}) &\stackrel{(2)}{=} (r^{(k)}, r^{(k)}) - (r^{(k)}, \alpha_k Ap^{(k)}) \\ &\stackrel{(3)}{=} (r^{(k)}, r^{(k)}) - (p^{(k)} - \beta_{k-1} p^{(k-1)}, \alpha_k Ap^{(k)}) \\ &\stackrel{(*)}{=} (r^{(k)}, r^{(k)}) - \alpha_k (p^{(k)}, Ap^{(k)}) \stackrel{(1)}{=} (r^{(k)}, r^{(k)}) - (p^{(k)}, r^{(k)}) \\ &\stackrel{(3)}{=} (r^{(k)}, r^{(k)}) - (\beta_{k-1} p^{(k-1)} + r^{(k)}, r^{(k)}) \\ &= -\beta_{k-1} (p^{(k-1)}, r^{(k)}) \stackrel{(2)}{=} -\beta_{k-1} (p^{(k-1)}, r^{(k-1)} - \alpha_{k-1} Ap^{(k-1)}) \\ &= -\beta_{k-1} \left\{ (p^{(k-1)}, r^{(k-1)}) - \alpha_{k-1} (p^{(k-1)}, Ap^{(k-1)}) \right\} \stackrel{(1)}{=} 0 \end{aligned}$$

$$(1) \alpha_k = \frac{(p^{(k)}, r^{(k)})}{(p^{(k)}, Ap^{(k)})}$$

$$(2) r^{(k+1)} = r^{(k)} - \alpha_k Ap^{(k)}$$

$$(3) p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)}$$

$$(4) \beta_k = \frac{-(r^{(k+1)}, Ap^{(k)})}{(p^{(k)}, Ap^{(k)})}$$

# Proof (3/3)

## Mathematical Induction

### 数学的帰納法

$$\begin{aligned} (r^{(i)}, r^{(j)}) &= 0 \quad (i \neq j) \\ (p^{(i)}, Ap^{(j)}) &= 0 \quad (i \neq j) \end{aligned} \quad (*)$$

(\*) is satisfied for  $i \leq k, j \leq k$  where  $i \neq j$

$$\begin{aligned} \text{if } i < k \quad (p^{(k+1)}, Ap^{(i)}) &\stackrel{(3)}{=} (r^{(k+1)} + \beta_k p^{(k)}, Ap^{(i)}) \\ &\stackrel{(*)}{=} (r^{(k+1)}, Ap^{(i)}) \end{aligned}$$

$$\stackrel{(2)}{=} \frac{1}{\alpha_k} (r^{(k+1)}, r^{(i)} - r^{(i-1)}) = 0$$

$$\begin{aligned} \text{if } i = k \quad (p^{(k+1)}, Ap^{(k)}) &\stackrel{(3)}{=} (r^{(k+1)}, Ap^{(k)}) + \beta_k (p^{(k)}, Ap^{(k)}) \\ &\stackrel{(4)}{=} 0 \end{aligned}$$

$$(1) \alpha_k = \frac{(p^{(k)}, r^{(k)})}{(p^{(k)}, Ap^{(k)})}$$

$$(2) r^{(k+1)} = r^{(k)} - \alpha_k Ap^{(k)}$$

$$(3) p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)}$$

$$(4) \beta_k = \frac{-(r^{(k+1)}, Ap^{(k)})}{(p^{(k)}, Ap^{(k)})}$$



$$\begin{aligned}
(r^{(k+1)}, r^{(k)}) &= 0 \\
(r^{(k+1)}, r^{(k)}) &\stackrel{(2)}{=} (r^{(k)}, r^{(k)}) - (r^{(k)}, \alpha_k A p^{(k)}) \\
&\stackrel{(3)}{=} (r^{(k)}, r^{(k)}) - (p^{(k)} - \beta_{k-1} p^{(k-1)}, \alpha_k A p^{(k)}) \\
&\stackrel{(*)}{=} (r^{(k)}, r^{(k)}) - \alpha_k (p^{(k)}, A p^{(k)}) \stackrel{(1)}{=} (r^{(k)}, r^{(k)}) - (p^{(k)}, r^{(k)}) = 0
\end{aligned}$$

$$\therefore (r^{(k)}, r^{(k)}) = (p^{(k)}, r^{(k)})$$

$$(1) \alpha_k = \frac{(p^{(k)}, r^{(k)})}{(p^{(k)}, A p^{(k)})}$$

$$(2) r^{(k+1)} = r^{(k)} - \alpha_k A p^{(k)}$$

$$(3) p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)}$$

$$(4) \beta_k = \frac{-(r^{(k+1)}, A p^{(k)})}{(p^{(k)}, A p^{(k)})}$$

# $\alpha_k, \beta_k$

実際は  $\alpha_k, \beta_k$  はもうちょっと簡単な形に変形できる:

$$\alpha_k = \frac{(p^{(k)}, b - Ax^{(k)})}{(p^{(k)}, Ap^{(k)})} = \frac{(p^{(k)}, r^{(k)})}{(p^{(k)}, Ap^{(k)})} = \frac{(r^{(k)}, r^{(k)})}{(p^{(k)}, Ap^{(k)})}$$

$$\because (p^{(k)}, r^{(k)}) = (r^{(k)}, r^{(k)})$$

$$\beta_k = \frac{-(r^{(k+1)}, Ap^{(k)})}{(p^{(k)}, Ap^{(k)})} = \frac{(r^{(k+1)}, r^{(k+1)})}{(r^{(k)}, r^{(k)})}$$

$$\because (r^{(k+1)}, Ap^{(k)}) = \frac{(r^{(k+1)}, r^{(k)} - r^{(k+1)})}{\alpha_k} = -\frac{(r^{(k+1)}, r^{(k+1)})}{\alpha_k}$$

# 共役勾配法 (CG法) のアルゴリズム

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
   $\rho_{i-1} = r^{(i-1)} \cdot r^{(i-1)}$ 
  if  $i = 1$ 
     $p^{(1)} = r^{(0)}$ 
  else
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $p^{(i)} = r^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
  endif
   $q^{(i)} = [A]p^{(i)}$ 
   $\alpha_i = \rho_{i-1} / p^{(i)} \cdot q^{(i)}$ 
   $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
   $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
  check convergence  $|r|$ 
end

```

$x^{(i)}$  : Vector

$\alpha_i$  : Scalar

$$\beta_{i-1} = \frac{(r^{(i-1)}, r^{(i-1)})}{(r^{(i-2)}, r^{(i-2)})} \quad (= \rho_{i-1})$$

$$\alpha_i = \frac{(r^{(i-1)}, r^{(i-1)})}{(p^{(i)}, Ap^{(i)})} \quad (= \rho_{i-1})$$

# 前処理 (preconditioning) とは？

- 反復法の収束は係数行列の固有値分布に依存
  - 固有値分布が少なく, かつ1に近いほど収束が早い(単位行列)
  - 条件数(condition number)(対称正定) = 最大最小固有値比
    - 条件数が1に近いほど収束しやすい
- もとの係数行列  $[A]$  に良く似た前処理行列  $[M]$  を適用することによって固有値分布を改善する。
  - 前処理行列  $[M]$  によって元の方程式  $[A] \{x\} = \{b\}$  を  $[A'] \{x\} = \{b'\}$  へと変換する。ここで  $[A'] = [M]^{-1} [A]$ ,  $\{b'\} = [M]^{-1} \{b\}$  である。
  - $[A'] = [M]^{-1} [A]$  が単位行列に近ければ良い
  - より一般的には  $[A'] \{x'\} = \{b'\}$  ( $[A'] = [M_L]^{-1} [A] [M_R]^{-1}$ ,  $\{b'\} = [M_L]^{-1} \{b\}$ ,  $\{x'\} = [M_R] \{x\}$ )
  - $[M_L] / [M_R]$  : 左／右前処理(left/right preconditioning)

# 前処理付き共役勾配法のアルゴリズム

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
  solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} \cdot z^{(i-1)}$ 
  if  $i = 1$ 
     $p^{(1)} = z^{(0)}$ 
  else
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
  endif
   $q^{(i)} = [A]p^{(i)}$ 
   $\alpha_i = \rho_{i-1} / p^{(i)} \cdot q^{(i)}$ 
   $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
   $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
  check convergence  $|r|$ 
end

```

$$[M] = [M_1][M_2]$$

$$[A']x' = b'$$

$$[A'] = [M_1]^{-1}[A][M_2]^{-1}$$

$$x' = [M_2]x, \quad b' = [M_1]^{-1}b$$

$$p' \Rightarrow [M_2]p, \quad r' \Rightarrow [M_1]^{-1}r$$

$$p'^x(i) = r'^{(i-1)} + \beta'_{i-1} p'^{(i-1)}$$

$$[M_2]p^{(i)} = [M_1]^{-1}r^{(i-1)} + \beta'_{i-1} [M_2]p^{(i-1)}$$

$$p^{(i)} = [M_2]^{-1}[M_1]^{-1}r^{(i-1)} + \beta'_{i-1} p^{(i-1)}$$

$$p^{(i)} = [M]^{-1}r^{(i-1)} + \beta'_{i-1} p^{(i-1)}$$

$$\beta'_{i-1} = \frac{([M]^{-1}r^{(i-1)}, r^{(i-1)})}{([M]^{-1}r^{(i-2)}, r^{(i-2)})}$$

$$\alpha'_{i-1} = \frac{([M]^{-1}r^{(i-1)}, r^{(i-1)})}{(p^{(i-1)}, [A]p^{(i-1)})}$$

# ILU(0), IC(0)

- 最もよく使用されている前処理(疎行列用)
  - 不完全LU分解
    - Incomplete LU Factorization
  - 不完全コレスキー分解
    - Incomplete Cholesky Factorization(対称行列)
- 不完全な直接法
  - もとの行列が疎でも, 逆行列は疎とは限らない。
  - fill-in
  - もとの行列と同じ非ゼロパターン(fill-in無し)を持っているのがILU(0), IC(0)

# 対角スケーリング, 点ヤコビ前処理

- 前処理行列として, もとの行列の対角成分のみを取り出した行列を前処理行列  $[M]$  とする。
  - 対角スケーリング, 点ヤコビ (point-Jacobi) 前処理

$$[M] = \begin{bmatrix} D_1 & 0 & \dots & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ \dots & & \dots & & \dots \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & \dots & 0 & D_N \end{bmatrix}$$

- **solve  $[M]z^{(i-1)} = r^{(i-1)}$**  という場合に逆行列を簡単に求めることができる。
- 簡単な問題では収束する。
- 1d.f, 1d.cはこの手法を使用している

- ガラーキン法による一次元弾性問題の解法
- 連立一次方程式の解法
  - 共役勾配法
  - 前処理手法
- 疎行列格納法
- プログラムの内容





# 1d.f, 1d.cにおけるマトリクス関連変数

変数名	型	サイズ	内容
N	I	-	未知数総数
NPLU	I	-	連立一次方程式係数マトリクス非対角成分総数
Diag(:)	R	N	連立一次方程式係数マトリクス対角成分
PHI(:)	R	N	連立一次方程式未知数ベクトル
Rhs(:)	R	N	連立一次方程式右辺ベクトル
Index(:)	I	0:N N+1	係数マトリクス非対角成分要素番号用一次元圧縮配列(非対角成分数)
Item(:)	I	NPLU	係数マトリクス非対角成分要素番号用一次元圧縮配列(非対角成分要素(列)番号)
AMat(:)	R	NPLU	係数マトリクス非対角成分要素番号用一次元圧縮配列(非対角成分)

非零非対角成分のみを格納する  
Compressed Row Storage法を使用している。

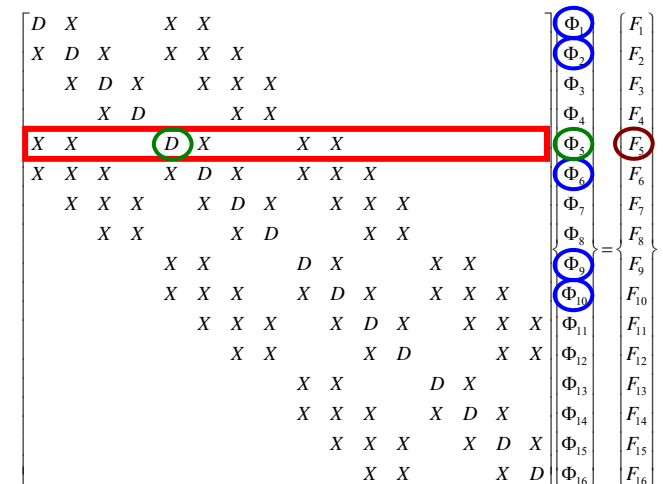
# 行列ベクトル積への適用

(非零)非対角成分のみを格納, 疎行列向け方法  
Compressed Row Storage (CRS)

Diag [i] 対角成分(実数, [N])  
 Index[i] 非対角成分に関する一次元配列  
 (通し番号)(整数, [N+1])  
 Item[k] 非対角成分の要素(列)番号  
 (整数, [Index[N]])  
 AMat[k] 非対角成分  
 (実数, [Index[N]])

$$\{Y\} = [A] \{X\}$$

```
for (i=0; i<N; i++) {
    Y[i] = Diag[i] * X[i];
    for (k=Index[i]; k<Index[i+1]; k++) {
        Y[i] += AMat[k]*X[Item[k]];
    }
}
```



# 行列ベクトル積：密行列⇒とても簡単

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1,N-1} & a_{1,N} \\ a_{21} & a_{22} & & a_{2,N-1} & a_{2,N} \\ \cdots & & \cdots & & \cdots \\ a_{N-1,1} & a_{N-1,2} & & a_{N-1,N-1} & a_{N-1,N} \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N-1} & a_{N,N} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{Bmatrix} = \begin{Bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \\ y_N \end{Bmatrix}$$

$$\{Y\} = [A] \{X\}$$

```
for (i=0; i<N; i++) {  
    Y[i] = 0.0;  
    for (j=0; j<N; j++) {  
        Y[i] = Y[i] + A[i][j]*X[j];  
    }  
}
```

# Compressed Row Storage (CRS)

	①	②	③	④	⑤	⑥	⑦	⑧
①	1.1	2.4	0	0	3.2	0	0	0
②	4.3	3.6	0	2.5	0	3.7	0	9.1
③	0	0	5.7	0	1.5	0	3.1	0
④	0	4.1	0	9.8	2.5	2.7	0	0
⑤	3.1	9.5	10.4	0	11.5	0	4.3	0
⑥	0	0	6.5	0	0	12.4	9.5	0
⑦	0	6.4	2.5	0	0	1.4	23.1	13.1
⑧	0	9.5	1.3	9.6	0	3.1	0	51.3

# Compressed Row Storage (CRS)

## Cでは0番から番号付け

	0	1	2	3	4	5	6	7
0	1.1 ◎ ①	2.4 ①			3.2 ④			
1	4.3 ◎ ①	3.6 ①		2.5 ③		3.7 ⑤		9.1 ⑦
2			5.7 ②		1.5 ④		3.1 ⑥	
3		4.1 ①		9.8 ③	2.5 ④	2.7 ⑤		
4	3.1 ◎ ①	9.5 ①	10.4 ②		11.5 ④		4.3 ⑥	
5			6.5 ②			12.4 ⑤	9.5 ⑥	
6		6.4 ①	2.5 ②			1.4 ⑤	23.1 ⑥	13.1 ⑦
7		9.5 ①	1.3 ②	9.6 ③		3.1 ⑤		51.3 ⑦

N= 8

対角成分

Diag[0]= 1.1  
 Diag[1]= 3.6  
 Diag[2]= 5.7  
 Diag[3]= 9.8  
 Diag[4]= 11.5  
 Diag[5]= 12.4  
 Diag[6]= 23.1  
 Diag[7]= 51.3

# Compressed Row Storage (CRS)

	0	1	2	3	4	5	6	7
0	1.1 ⊙ ④		2.4 ①			3.2 ④		
1	3.6 ①	4.3 ⊙			2.5 ③		3.7 ⑤	9.1 ⑦
2	5.7 ②				1.5 ④		3.1 ⑥	
3	9.8 ③		4.1 ①		2.5 ④	2.7 ⑤		
4	11.5 ④	3.1 ⊙	9.5 ①	10.4 ②			4.3 ⑥	
5	12.4 ⑤			6.5 ②			9.5 ⑥	
6	23.1 ⑥		6.4 ①	2.5 ②			1.4 ⑤	13.1 ⑦
7	51.3 ⑦		9.5 ①	1.3 ②	9.6 ③		3.1 ⑤	

# Compressed Row Storage (CRS)

	非対角成分数						
0	1.1 ◎	2.4 ①	3.2 ④			2	Index[0] = 0
1	3.6 ①	4.3 ◎	2.5 ③	3.7 ⑤	9.1 ⑦	4	Index[1] = 2
2	5.7 ②	1.5 ④	3.1 ⑥			2	Index[2] = 6
3	9.8 ③	4.1 ①	2.5 ④	2.7 ⑤		3	Index[3] = 8
4	11.5 ④	3.1 ◎	9.5 ①	10.4 ②	4.3 ⑥	4	Index[4] = 11
5	12.4 ⑤	6.5 ②	9.5 ⑥			2	Index[5] = 15
6	23.1 ⑥	6.4 ①	2.5 ②	1.4 ⑤	13.1 ⑦	4	Index[6] = 17
7	51.3 ⑦	9.5 ①	1.3 ②	9.6 ③	3.1 ⑤	4	Index[7] = 21
							Index[8] = 25

**NPLU = 25**  
**(=Index[N])**

Index[i] ~ Index[i+1] - 1番目がi行目の非対角成分



# Compressed Row Storage (CRS)

	非対角成分数						
0	1.1 ⊙	2.4 ①	3.2 ④			2	Index[0] = 0
1	3.6 ①	4.3 ⊙	2.5 ③	3.7 ⑤	9.1 ⑦	4	Index[1] = 2
2	5.7 ②	1.5 ④	3.1 ⑥			2	<u>Index[2] = 6</u>
3	9.8 ③	<b>4.1 ①</b>	<b>2.5 ④</b>	<b>2.7 ⑤</b>		3	<u>Index[3] = 8</u>
4	11.5 ④	3.1 ⊙	9.5 ①	10.4 ②	4.3 ⑥	4	<u>Index[4] = 11</u>
5	12.4 ⑤	6.5 ②	9.5 ⑥			2	Index[5] = 15
6	23.1 ⑥	6.4 ①	2.5 ②	1.4 ⑤	13.1 ⑦	4	Index[6] = 17
7	51.3 ⑦	9.5 ①	1.3 ②	9.6 ③	3.1 ⑤	4	Index[7] = 21
							Index[8] = 25

**NPLU = 25**  
(=Index[N])

Index[i] ~ Index[i+1] - 1番目がi行目の非対角成分

# Compressed Row Storage (CRS)

0	1.1 ⊙	2.4 ①	3.2 ④		
1	3.6 ①	4.3 ⊙	2.5 ③	3.7 ⑤	9.1 ⑦
2	5.7 ②	<b>1.5</b> ④	3.1 ⑥		
3	9.8 ③	4.1 ①	2.5 ④	2.7 ⑤	
4	11.5 ④	3.1 ⊙	9.5 ①	10.4 ②	4.3 ⑥
5	12.4 ⑤	6.5 ②	9.5 ⑥		
6	23.1 ⑥	6.4 ①	<b>2.5</b> ②	1.4 ⑤	13.1 ⑦
7	51.3 ⑦	9.5 ①	1.3 ②	9.6 ③	3.1 ⑤

例:

Item[ 6]= 4, AMat[ 6]= 1.5

Item[18]= 2, AMat[18]= 2.5

# Compressed Row Storage (CRS)

0	1.1 ⊙	2.4 ①,0	3.2 ④,1		
1	3.6 ①	4.3 ⊙,2	2.5 ③,3	3.7 ⑤,4	9.1 ⑦,5
2	5.7 ②	1.5 ④,6	3.1 ⑥,7		
3	9.8 ③	4.1 ①,8	2.5 ④,9	2.7 ⑤,10	
4	11.5 ④	3.1 ⊙,11	9.5 ①,12	10.4 ②,13	4.3 ⑥,14
5	12.4 ⑤	6.5 ②,15	9.5 ⑥,16		
6	23.1 ⑥	6.4 ①,17	2.5 ②,18	1.4 ⑤,19	13.1 ⑦,20
7	51.3 ⑦	9.5 ①,21	1.3 ②,22	9.6 ③,23	3.1 ⑤,24

Diag [i] 対角成分(実数, [N])  
 Index[i] 非対角成分に関する一次元配列  
 (通し番号)(整数, [N+1])  
 Item[k] 非対角成分の要素(列)番号  
 (整数, [Index[N]])  
 AMat[k] 非対角成分  
 (実数, [Index[N]])

$\{Y\} = [A] \{X\}$

```

for (i=0; i<N; i++) {
    Y[i] = Diag[i] * X[i];
    for (k=Index[i]; k<Index[i+1]; k++) {
        Y[i] += AMat[k]*X[Item[k]];
    }
}
  
```

# Compressed Row Storage (CRS): C

0	1.1 ⊙	2.4 ①,0	3.2 ④,1		
1	3.6 ①	4.3 ⊙,2	2.5 ③,3	3.7 ⑤,4	9.1 ⑦,5
2	5.7 ②	1.5 ④,6	3.1 ⑥,7		
3	9.8 ③	4.1 ①,8	2.5 ④,9	2.7 ⑤,10	
4	11.5 ④	3.1 ⊙,11	9.5 ①,12	10.4 ②,13	4.3 ⑥,14
5	12.4 ⑤	6.5 ②,15	9.5 ⑥,16		
6	23.1 ⑥	6.4 ①,17	2.5 ②,18	1.4 ⑤,19	13.1 ⑦,20
7	51.3 ⑦	9.5 ①,21	1.3 ②,22	9.6 ③,23	3.1 ⑤,24

Diag [i] 対角成分(実数, [N])  
 Index [i] 非対角成分に関する一次元配列  
 (通し番号)(整数, [N+1])  
 Item [k] 非対角成分の要素(列)番号  
 (整数, [Index[N]])  
 AMat [k] 非対角成分  
 (実数, [Index[N]])

$\{Y\} = [A] \{X\}$

```

for (i=0; i<N; i++) {
    Y[i] = Diag[i] * X[i];
    for (k=Index[i]; k<Index[i+1]; k++) {
        Y[i] += AMat[k]*X[Item[k]];
    }
}
  
```

- ガラーキン法による一次元弾性問題の解法
- 連立一次方程式の解法
  - 共役勾配法
  - 前処理手法
- 疎行列格納法
- プログラムの内容

# 有限要素法の処理: プログラム

- 初期化
  - 制御変数読み込み
  - 座標読み込み⇒要素生成(N:節点数, NE:要素数)
  - 配列初期化(全体マトリクス, 要素マトリクス)
  - 要素⇒全体マトリクスマッピング(Index, Item)
- マトリクス生成
  - 要素単位の処理(do icel= 1, NE)
    - 要素マトリクス計算
    - 全体マトリクスへの重ね合わせ
  - 境界条件の処理
- 連立一次方程式
  - 共役勾配法(CG)

# プログラム: 1d.c (1/6)

## 諸変数

```
/*  
// 1D Steady-State Heat Transfer  
// FEM with Piece-wise Linear Elements  
// CG (Conjugate Gradient) Method  
//  
//  $d/dx(CdT/dx) + Q = 0$   
//  $T=0@x=0$   
*/  
#include <stdio.h>  
#include <stdlib.h>  
#include <math.h>  
#include <assert.h>  
  
int main() {  
    int NE, N, NPLU, IterMax;  
    int R, Z, Q, P, DD;  
  
    double dX, Resid, Eps, Area, QV, COND;  
    double X1, X2, U1, U2, DL, Strain, Sigma, Ck;  
    double QN, XL, C2, Xi, PHIA;  
    double *PHI, *Rhs, *X;  
    double *Diag, *AMat;  
    double **W;  
    int *Index, *Item, *Icelnod;  
    double Kmat[2][2], Emat[2][2];  
    int i, j, in1, in2, k, icel, k1, k2, js;  
    int iter;  
    FILE *fp;  
    double BNorm2, Rho, Rho1=0.0, C1, Alpha, DNorm2;  
    int ierr = 1;  
    int errno = 0;
```

# 変数表(1/2)

変数名	種別	サイズ	I/O	内 容
NE	I		I	要素数
N	I		O	節点数
NPLU	I		O	非零非対角成分数
IterMax	I		I	最大反復回数
errno	I		O	エラー戻り値
R, Z, Q, P, DD	I		O	CG法ベクトル名
dX	R		I	要素長さ
Resid	R		O	CG法残差
Eps	R		I	CG法反復打ち切り残差
Area	R		I	要素断面積 $Q$
QV	R		I	体積当たり発熱量
COND	R		I	熱伝導率



# 変数表 (2/2)

変数名	種別	サイズ	I/O	内 容
X	R	N	O	節点座標
U	R	N	O	節点変位
Rhs	R	N	O	右辺ベクトル
Diag	R	N	O	全体マトリクス：対角成分
W	R	[4] [N]	O	CG法のwork配列
Amat	R	NPLU	O	全体マトリクス：非零非対角成分
Index	I	N+1	O	全体マトリクス：各行の非零非対角成分数
Item	I	NPLU	O	全体マトリクス：列番号
Icelnod	I	2*NE	O	各要素節点番号
Kmat	R	[2] [2]	O	要素マトリクス[k]
Emat	R	[2] [2]	O	要素マトリクス

# プログラム: 1d.c (2/6)

## 初期設定, 配列宣言

```

/*
// +-----+
// |  INIT.  |
// +-----+
*/
fp = fopen("input.dat", "r");
assert(fp != NULL);
fscanf(fp, "%d", &NE);
fscanf(fp, "%lf %lf %lf %lf", &dX, &QV, &Area, &COND);
fscanf(fp, "%d", &IterMax);
fscanf(fp, "%lf", &Eps);
fclose(fp);

N= NE + 1;

PHI = calloc(N, sizeof(double));
X    = calloc(N, sizeof(double));
Diag = calloc(N, sizeof(double));

AMat = calloc(2*N-2, sizeof(double));

Rhs = calloc(N, sizeof(double));

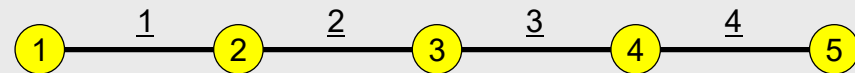
Index= calloc(N+1, sizeof(int));
Item  = calloc(2*N-2, sizeof(int));

Icelnod= calloc(2*NE, sizeof(int));

```

制御ファイル input.dat

4	NE (要素数)
1.0 1.0 1.0 1.0	$\Delta x$ (要素長さL) Q, A, COND
100	反復回数
1.e-8	CG法の反復打切誤差



NE : 要素数  
N : 節点数 (=NE+1)

# プログラム: 1d.c (2/6)

## 初期設定, 配列宣言

```
/*
// +-----+
// | INIT. |
// +-----+
*/
fp = fopen("input.dat", "r");
assert(fp != NULL);
fscanf(fp, "%d", &NE);
fscanf(fp, "%lf %lf %lf %lf", &dX, &QV, &Area, &COND);
fscanf(fp, "%d", &IterMax);
fscanf(fp, "%lf", &Eps);
fclose(fp);

N= NE + 1;

PHI = calloc(N, sizeof(double));
X    = calloc(N, sizeof(double));
Diag = calloc(N, sizeof(double));

AMat = calloc(2*N-2, sizeof(double));
Rhs = calloc(N, sizeof(double));

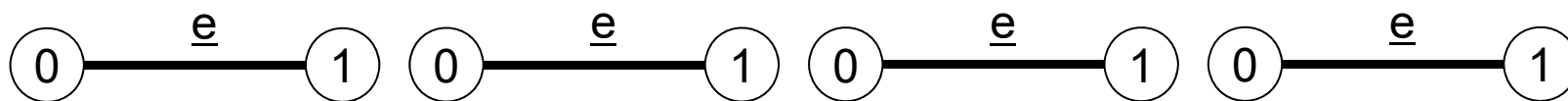
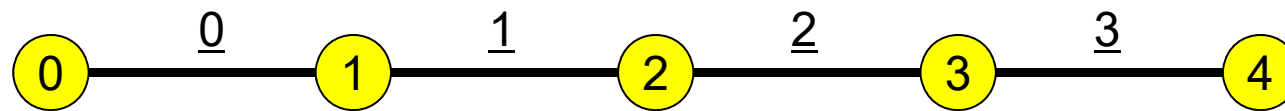
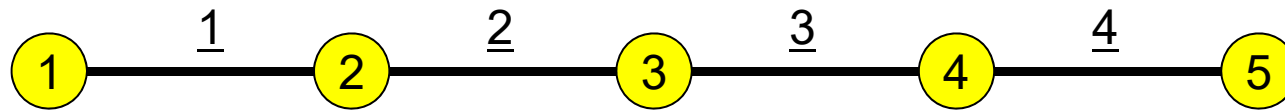
Index= calloc(N+1, sizeof(int));
Item = calloc(2*N-2, sizeof(int));

Icelnod= calloc(2*NE, sizeof(int));
```

**Amat :** 非零非対角成分  
**Item :** 対応する列番号



注意：プログラムの中では節点・要素番号は0からふられている（C言語）



# プログラム: 1d.c (2/6)

## 初期設定, 配列宣言

```

/*
// +-----+
// | INIT. |
// +-----+
*/
fp = fopen("input.dat", "r");
assert(fp != NULL);
fscanf(fp, "%d", &NE);
fscanf(fp, "%lf %lf %lf %lf", &dX, &QV, &Area, &COND);
fscanf(fp, "%d", &IterMax);
fscanf(fp, "%lf", &Eps);
fclose(fp);

```

```
N = NE + 1;
```

```

PHI = calloc(N, sizeof(double));
X    = calloc(N, sizeof(double));
Diag = calloc(N, sizeof(double));

```

```
AMat = calloc(2*N-2, sizeof(double));
```

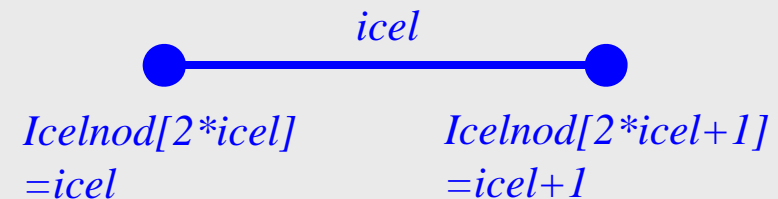
```
Rhs = calloc(N, sizeof(double));
```

```

Index = calloc(N+1, sizeof(int));
Item = calloc(2*N-2, sizeof(int));

```

```
Icelnod = calloc(2*NE, sizeof(int));
```



Amat : 非零非対角成分

Item : 対応する列番号

各節点の非零非対角成分数は「2」  
(ただし両端では「1」)

総数 :  $2*(N-2)+1+1 = 2*N-2$

# プログラム: 1d.c (3/6)

## 配列宣言(続き), 初期化

```
W = (double **)malloc(sizeof(double *)*4);
if(W == NULL) {
    fprintf(stderr, "Error: %s\n", strerror(errno));
    return -1;
}
for(i=0; i<4; i++) {
    W[i] = (double *)malloc(sizeof(double)*N);
    if(W[i] == NULL) {
        fprintf(stderr, "Error: %s\n", strerror(errno));
        return -1;
    }
}
```

```
for(i=0; i<N; i++) PHI[i] = 0.0;
for(i=0; i<N; i++) Diag[i] = 0.0;
for(i=0; i<N; i++) Rhs[i] = 0.0;
for(k=0; k<2*N-2; k++) AMat[k] = 0.0;
```

```
for(i=0; i<N; i++) X[i]= i*dX;
```

X : 各節点の座標

```
for(icel=0; icel<NE; icel++) {
    Icelnod[2*icel] = icel;
    Icelnod[2*icel+1] = icel+1;
}
```

```
Kmat[0][0]= +1.0;
Kmat[0][1]= -1.0;
Kmat[1][0]= -1.0;
Kmat[1][1]= +1.0;
```

# プログラム: 1d.c (3/6)

## 配列宣言(続き), 初期化

```

W = (double **)malloc(sizeof(double *)*4);
  if(W == NULL) {
    fprintf(stderr, "Error: %s\n", strerror(errno));
    return -1;
  }
  for(i=0; i<4; i++) {
    W[i] = (double *)malloc(sizeof(double)*N);
    if(W[i] == NULL) {
      fprintf(stderr, "Error: %s\n", strerror(errno));
      return -1;
    }
  }

```

```

for(i=0; i<N; i++) PHI[i] = 0.0;
for(i=0; i<N; i++) Diag[i] = 0.0;
for(i=0; i<N; i++) Rhs[i] = 0.0;
for(k=0; k<2*N-2; k++) AMat[k] = 0.0;

```

```

for(i=0; i<N; i++) X[i]= i*dX;

```

```

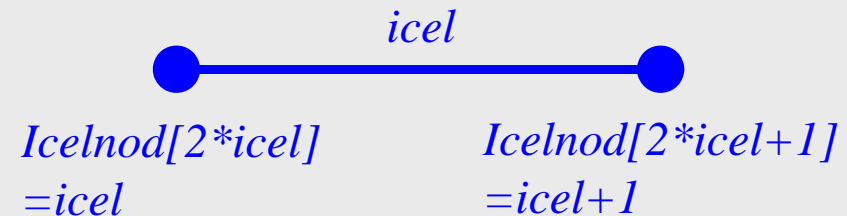
for(icel=0; icel<NE; icel++) {
  Icelnod[2*icel] = icel;
  Icelnod[2*icel+1] = icel+1;
}

```

```

Kmat[0][0]= +1.0;
Kmat[0][1]= -1.0;
Kmat[1][0]= -1.0;
Kmat[1][1]= +1.0;

```





# プログラム: 1d.c (3/6)

## 配列宣言(続き), 初期化

```

W = (double **)malloc(sizeof(double *)*4);
if(W == NULL) {
    fprintf(stderr, "Error: %s\n", strerror(errno));
    return -1;
}
for(i=0; i<4; i++) {
    W[i] = (double *)malloc(sizeof(double)*N);
    if(W[i] == NULL) {
        fprintf(stderr, "Error: %s\n", strerror(errno));
        return -1;
    }
}

```

```

for(i=0; i<N; i++) PHI[i] = 0.0;
for(i=0; i<N; i++) Diag[i] = 0.0;
for(i=0; i<N; i++) Rhs[i] = 0.0;
for(k=0; k<2*N-2; k++) AMat[k] = 0.0;

```

```

for(i=0; i<N; i++) X[i] = i*dX;

```

```

for(icel=0; icel<NE; icel++) {
    Icelnod[2*icel] = icel;
    Icelnod[2*icel+1] = icel+1;
}

```

```

Kmat[0][0] = +1.0;
Kmat[0][1] = -1.0;
Kmat[1][0] = -1.0;
Kmat[1][1] = +1.0;

```

$$[k]^{(e)} = \int_V \lambda \left( \frac{d[N]^T}{dx} \frac{d[N]}{dx} \right) dV = \frac{\lambda A}{L} \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}$$

[Kmat]

# プログラム: 1d.c(4/6)

## 全体マトリクス: 非零非対角成分に対応する列番号

```

/*
// +-----+
// | CONNECTIVITY |
// +-----+
*/
for (i=0; i<N+1; i++) Index[i] = 2;
Index[0] = 0;
Index[1] = 1;
Index[N] = 1;

for (i=0; i<N; i++) {
    Index[i+1] = Index[i+1] + Index[i];
}

NPLU = Index[N];

for (i=0; i<N; i++) {
    jS = Index[i];
    if (i == 0) {
        Item[jS] = i+1;
    } else if (i == N-1) {
        Item[jS] = i-1;
    } else {
        Item[jS] = i-1;
        Item[jS+1] = i+1;
    }
}

```

各節点の非零非対角成分数は「2」  
(ただし両端では「1」)

総数:  $2*(N-2)+1+1 = 2*N-2$   
 $Index[N] = 2*N-2 = NPLU$

	非対角成分数	Index
0	2	Index[0] = 0
1	4	Index[1] = 2
2	2	Index[2] = 6
3	3	Index[3] = 8
4	4	Index[4] = 11
5	2	Index[5] = 15
6	4	Index[6] = 17
7	4	Index[7] = 21
		Index[8] = 25

0	1.1 ◎	2.4 ①	3.2 ④		
1	3.6 ①	4.3 ◎	2.5 ③	3.7 ⑤	9.1 ⑦
2	5.7 ②	1.5 ④	3.1 ⑥		
3	9.8 ③	4.1 ①	2.5 ④	2.7 ⑤	
4	11.5 ④	3.1 ◎	9.5 ①	10.4 ②	4.3 ⑥
5	12.4 ⑤	6.5 ②	9.5 ⑥		
6	23.1 ⑥	6.4 ①	2.5 ②	1.4 ⑤	13.1 ⑦
7	51.3 ⑦	9.5 ①	1.3 ②	9.6 ③	3.1 ⑤

Index[i]~Index[i+1]-1番目がi行目の非対角成分

# プログラム: 1d.c(4/6)

## 全体マトリクス: 非零非対角成分に対応する列番号

```

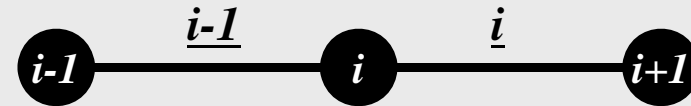
/*
// +-----+
// | CONNECTIVITY |
// +-----+
*/
for (i=0; i<N+1; i++) Index[i] = 2;
Index[0]= 0;
Index[1]= 1;
Index[N]= 1;

for (i=0; i<N; i++) {
    Index[i+1]= Index[i+1] + Index[i];
}

NPLU= Index[N];

for (i=0; i<N; i++) {
    jS = Index[i];
    if (i == 0) {
        Item[jS] = i+1;
    } else if (i == N-1) {
        Item[jS] = i-1;
    } else {
        Item[jS] = i-1;
        Item[jS+1] = i+1;
    }
}

```



行番号	非対角成分数	Index
0	2	Index[0] = 0
1	4	Index[1] = 2
2	2	Index[2] = 6
3	3	Index[3] = 8
4	4	Index[4] = 11
5	2	Index[5] = 15
6	4	Index[6] = 17
7	4	Index[7] = 21
8	4	Index[8] = 25

0	1.1 ◎	2.4 ①	3.2 ④		
1	3.6 ①	4.3 ◎	2.5 ③	3.7 ⑤	9.1 ⑦
2	5.7 ②	1.5 ④	3.1 ⑥		
3	9.8 ③	4.1 ①	2.5 ④	2.7 ⑤	
4	11.5 ④	3.1 ◎	9.5 ①	10.4 ②	4.3 ⑥
5	12.4 ⑤	6.5 ②	9.5 ⑥		
6	23.1 ⑥	6.4 ①	2.5 ②	1.4 ⑤	13.1 ⑦
7	51.3 ⑦	9.5 ①	1.3 ②	9.6 ③	3.1 ⑤

Index[i]~Index[i+1]-1番目がi行目の非対角成分

# プログラム: 1d.c (5/6)

全体マトリクス生成: 要素マトリクス ⇒ 全体マトリクス

```

/*
// |-----|
// | MATRIX assemble |
// |-----|
*/
for (icel=0; icel<NE; icel++) {
    in1= Icelnod[2*icel];
    in2= Icelnod[2*icel+1];
    X1 = X[in1];
    X2 = X[in2];
    DL = fabs(X2-X1);

    Ck= Area*COND/DL;
    Emat[0][0]= Ck*Kmat[0][0];
    Emat[0][1]= Ck*Kmat[0][1];
    Emat[1][0]= Ck*Kmat[1][0];
    Emat[1][1]= Ck*Kmat[1][1];

    Diag[in1]= Diag[in1] + Emat[0][0];
    Diag[in2]= Diag[in2] + Emat[1][1];

    if (icel==0) {k1=Index[in1];
    } else {k1=Index[in1]+1;}
    k2=Index[in2];

    AMat[k1]= AMat[k1] + Emat[0][1];
    AMat[k2]= AMat[k2] + Emat[1][0];

    QN= 0.5*QV*Area*dX;
    Rhs[in1]= Rhs[in1] + QN;
    Rhs[in2]= Rhs[in2] + QN;
}

```



# プログラム: 1d.c (5/6)

全体マトリクス生成: 要素マトリクス ⇒ 全体マトリクス

```

/*
// |-----|
// | MATRIX assemble |
// |-----|
*/
for (icel=0; icel<NE; icel++) {
  in1= Icelnod[2*icel];
  in2= Icelnod[2*icel+1];
  X1 = X[in1];
  X2 = X[in2];
  DL = fabs(X2-X1);

  Ck= Area*COND/DL;
  Emat[0][0]= Ck*Kmat[0][0];
  Emat[0][1]= Ck*Kmat[0][1];
  Emat[1][0]= Ck*Kmat[1][0];
  Emat[1][1]= Ck*Kmat[1][1];

  Diag[in1]= Diag[in1] + Emat[0][0];
  Diag[in2]= Diag[in2] + Emat[1][1];

  if (icel==0) {k1=Index[in1];
  }else {k1=Index[in1]+1;}
  k2=Index[in2];

  AMat[k1]= AMat[k1] + Emat[0][1];
  AMat[k2]= AMat[k2] + Emat[1][0];

  QN= 0.5*QV*Area*dX;
  Rhs[in1]= Rhs[in1] + QN;
  Rhs[in2]= Rhs[in2] + QN;
}

```



$$[Emat] = [k]^{(e)} = \frac{\lambda A}{L} \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} = \frac{\lambda A}{L} [Kmat]$$

# プログラム: 1d.c (5/6)

全体マトリクス生成: 要素マトリクス ⇒ 全体マトリクス

```

/*
// -----
// | MATRIX assemble |
// -----
*/
for (icel=0; icel<NE; icel++) {
    in1= Icelnod[2*icel];
    in2= Icelnod[2*icel+1];
    X1 = X[in1];
    X2 = X[in2];
    DL = fabs(X2-X1);

    Ck= Area*COND/DL;
    Emat[0][0]= Ck*Kmat[0][0];
    Emat[0][1]= Ck*Kmat[0][1];
    Emat[1][0]= Ck*Kmat[1][0];
    Emat[1][1]= Ck*Kmat[1][1];

    Diag[in1]= Diag[in1] + Emat[0][0];
    Diag[in2]= Diag[in2] + Emat[1][1];

    if (icel==0) {k1=Index[in1];
    } else {k1=Index[in1]+1;}
    k2=Index[in2];

    AMat[k1]= AMat[k1] + Emat[0][1];
    AMat[k2]= AMat[k2] + Emat[1][0];

    QN= 0.5*QV*Area*dX;
    Rhs[in1]= Rhs[in1] + QN;
    Rhs[in2]= Rhs[in2] + QN;
}

```



$$[Emat] = [k]^{(e)} = \frac{EA}{L} \begin{bmatrix} \textcircled{+1} & -1 \\ -1 & \textcircled{+1} \end{bmatrix}$$

# プログラム: 1d.c (5/6)

全体マトリクス生成: 要素マトリクス ⇒ 全体マトリクス

```

/*
// |-----|
// | MATRIX assemble |
// |-----|
*/

for (icel=0; icel<NE; icel++) {
    in1= Icelnod[2*icel];
    in2= Icelnod[2*icel+1];
    X1 = X[in1];
    X2 = X[in2];
    DL = fabs(X2-X1);

    Ck= Area*COND/DL;
    Emat[0][0]= Ck*Kmat[0][0];
    Emat[0][1]= Ck*Kmat[0][1];
    Emat[1][0]= Ck*Kmat[1][0];
    Emat[1][1]= Ck*Kmat[1][1];

    Diag[in1]= Diag[in1] + Emat[0][0];
    Diag[in2]= Diag[in2] + Emat[1][1];

    if (icel==0) {k1=Index[in1];
    } else {k1=Index[in1]+1;}
    k2=Index[in2];

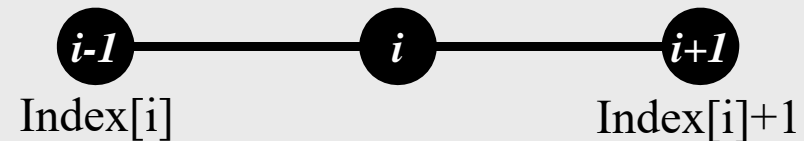
    AMat[k1]= AMat[k1] + Emat[0][1];
    AMat[k2]= AMat[k2] + Emat[1][0];

    QN= 0.5*QV*Area*dX;
    Rhs[in1]= Rhs[in1] + QN;
    Rhs[in2]= Rhs[in2] + QN;
}

```



「i」 行の非対角成分：  
Index[i], Index[i]+1



$$[Emat] = [k]^{(e)} = \frac{\lambda A}{L} \begin{bmatrix} +1 & \ominus -1 \\ \ominus -1 & +1 \end{bmatrix} \begin{matrix} k1 \\ k2 \end{matrix}$$

k2

# 通常の要素:k1

## in1の非対角成分としてのin2

```

/*
// |-----|
// | MATRIX assemble |
// |-----|
*/
for (icel=0; icel<NE; icel++) {
  in1= Icelnod[2*icel];
  in2= Icelnod[2*icel+1];
  X1 = X[in1];
  X2 = X[in2];
  DL = fabs(X2-X1);

  Ck= Area*COND/DL;
  Emat[0][0]= Ck*Kmat[0][0];
  Emat[0][1]= Ck*Kmat[0][1];
  Emat[1][0]= Ck*Kmat[1][0];
  Emat[1][1]= Ck*Kmat[1][1];

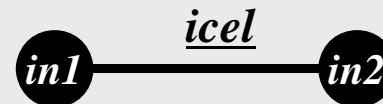
  Diag[in1]= Diag[in1] + Emat[0][0];
  Diag[in2]= Diag[in2] + Emat[1][1];

  if (icel==0) {k1=Index[in1];
  } else {k1=Index[in1]+1;}
  k2=Index[in2];

  AMat[k1]= AMat[k1] + Emat[0][1];
  AMat[k2]= AMat[k2] + Emat[1][0];

  QN= 0.5*QV*Area*dX;
  Rhs[in1]= Rhs[in1] + QN;
  Rhs[in2]= Rhs[in2] + QN;
}

```



「i」 行の非対角成分：  
Index[i], Index[i]+1



$$[Emat] = [k]^{(e)} = \frac{\lambda A}{L} \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \quad k1$$



# 通常の要素:k2

## in2の非対角成分としてのin1

```

/*
// |-----|
// | MATRIX assemble |
// |-----|
*/
for (icel=0; icel<NE; icel++) {
  in1= Icelnod[2*icel];
  in2= Icelnod[2*icel+1];
  X1 = X[in1];
  X2 = X[in2];
  DL = fabs(X2-X1);

  Ck= Area*COND/DL;
  Emat[0][0]= Ck*Kmat[0][0];
  Emat[0][1]= Ck*Kmat[0][1];
  Emat[1][0]= Ck*Kmat[1][0];
  Emat[1][1]= Ck*Kmat[1][1];

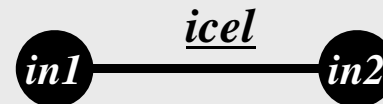
  Diag[in1]= Diag[in1] + Emat[0][0];
  Diag[in2]= Diag[in2] + Emat[1][1];

  if (icel==0) {k1=Index[in1];
  } else {k1=Index[in1]+1;}
  k2=Index[in2];

  AMat[k1]= AMat[k1] + Emat[0][1];
  AMat[k2]= AMat[k2] + Emat[1][0];

  QN= 0.5*QV*Area*dX;
  Rhs[in1]= Rhs[in1] + QN;
  Rhs[in2]= Rhs[in2] + QN;
}

```



「i」 行の非対角成分：  
Index[i], Index[i]+1



$$[Emat] = [k]^{(e)} = \frac{\lambda A}{L} \begin{bmatrix} +1 & -1 \\ \ominus 1 & +1 \end{bmatrix}$$

k2

# 0番要素(左端): k1

## in1の非対角成分としてのin2

```

/*
// |-----|
// | MATRIX assemble |
// |-----|
*/
for (icel=0; icel<NE; icel++) {
  in1= Icelnod[2*icel];
  in2= Icelnod[2*icel+1];
  X1 = X[in1];
  X2 = X[in2];
  DL = fabs(X2-X1);

  Ck= Area*COND/DL;
  Emat[0][0]= Ck*Kmat[0][0];
  Emat[0][1]= Ck*Kmat[0][1];
  Emat[1][0]= Ck*Kmat[1][0];
  Emat[1][1]= Ck*Kmat[1][1];

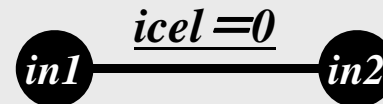
  Diag[in1]= Diag[in1] + Emat[0][0];
  Diag[in2]= Diag[in2] + Emat[1][1];

  if (icel==0) {k1=Index[in1];
  } else {k1=Index[in1]+1;}
  k2=Index[in2];

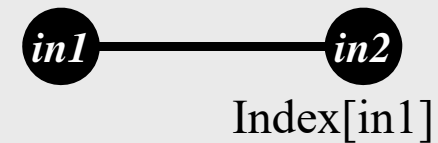
  AMat[k1]= AMat[k1] + Emat[0][1];
  AMat[k2]= AMat[k2] + Emat[1][0];

  QN= 0.5*QV*Area*dX;
  Rhs[in1]= Rhs[in1] + QN;
  Rhs[in2]= Rhs[in2] + QN;
}

```



「i」行の非対角成分: Index[i]のみ



$$[Emat] = [k]^{(e)} = \frac{\lambda A}{L} \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \quad k1$$

# プログラム: 1d.c (5/6)

## 体積発熱項, 右辺

```

/*
// +-----+
// | MATRIX assemble |
// +-----+
*/
for (icel=0; icel<NE; icel++) {
  in1= Icelnod[2*icel];
  in2= Icelnod[2*icel+1];
  X1 = X[in1];
  X2 = X[in2];
  DL = fabs(X2-X1);

  Ck= Area*COND/DL;
  Emat[0][0]= Ck*Kmat[0][0];
  Emat[0][1]= Ck*Kmat[0][1];
  Emat[1][0]= Ck*Kmat[1][0];
  Emat[1][1]= Ck*Kmat[1][1];

  Diag[in1]= Diag[in1] + Emat[0][0];
  Diag[in2]= Diag[in2] + Emat[1][1];

  if (icel==0) {k1=Index[in1];
  } else {k1=Index[in1]+1;}
  k2=Index[in2];

  AMat[k1]= AMat[k1] + Emat[0][1];
  AMat[k2]= AMat[k2] + Emat[1][0];

  QN= 0.5*QV*Area*dX;
  Rhs[in1]= Rhs[in1] + QN;
  Rhs[in2]= Rhs[in2] + QN;
}

```



$$\int_V \dot{Q}[N]^T dV = \dot{Q}A \int_0^L \begin{bmatrix} 1-x/L \\ x/L \end{bmatrix} dx = \frac{\dot{Q}AL}{2} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$$

# プログラム: 1d.c (6/6)

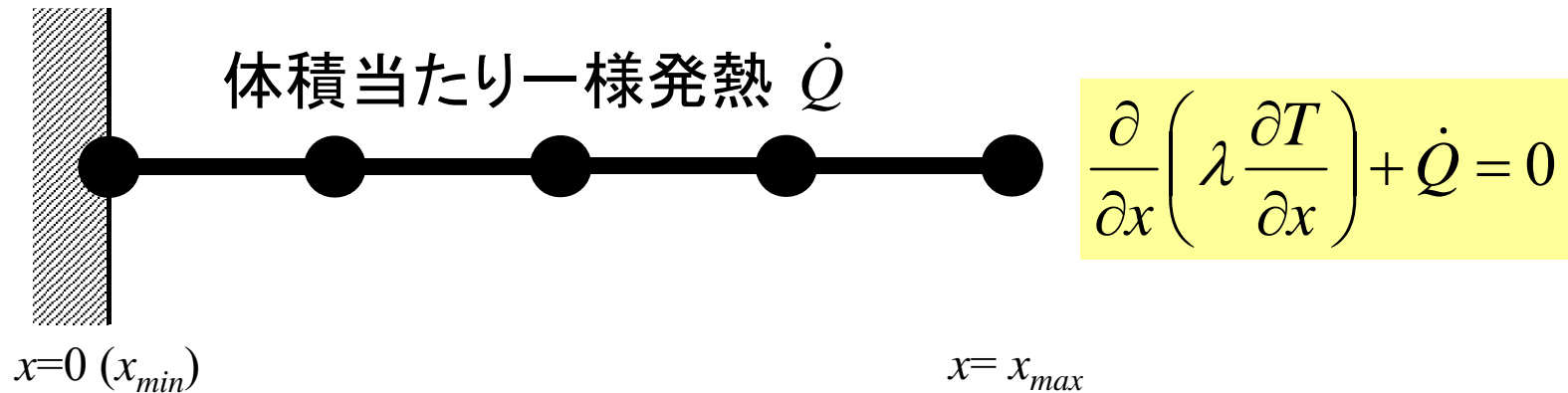
## 第一種境界条件 @x=0

```
/*
// +-----+
// | BOUNDARY conditions |
// +-----+
*/

/* X=Xmin */
  i=0;
  jS= Index[i];
  AMat[jS]= 0.0;
  Diag[i ]= 1.0;
  Rhs [i ]= 0.0;

  for (k=0;k<NPLU;k++) {
    if (Item[k]==0) {AMat[k]=0.0;
    }}
}
```

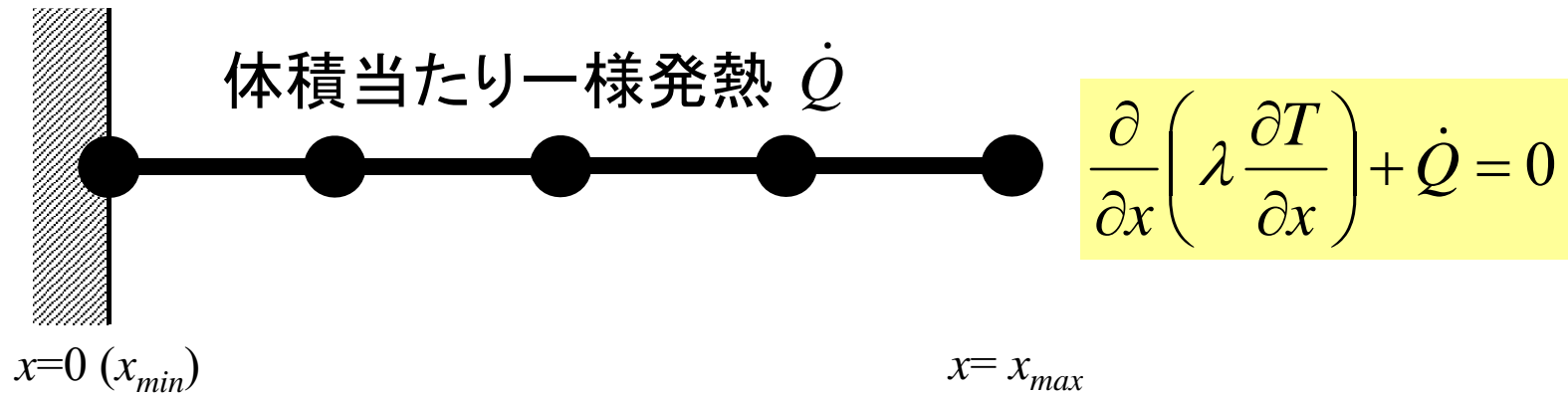
# 対象とする問題：一次元熱伝導問題



- 一様な：断面積 $A$ ，熱伝導率 $\lambda$
- 体積当たり一様発熱（時間当たり） $[QL^{-3}T^{-1}]$   $\dot{Q}$
- 境界条件
  - $x=0$  :  $T=0$ （固定）
  - $x=x_{max}$  :  $\frac{\partial T}{\partial x} = 0$ （断熱）

# $x=0$ で成立する方程式

$$T_0=0$$



- 一様な：断面積 $A$ ，熱伝導率 $\lambda$
- 体積当たり一様発熱（時間当たり） $[QL^{-3}T^{-1}]$   $\dot{Q}$
- 境界条件
  - $x=0$  :  $T=0$ （固定）
  - $x=x_{max}$  :  $\frac{\partial T}{\partial x} = 0$ （断熱）

# プログラム: 1d.c (6/6)

## 第一種境界条件 @x=0

```

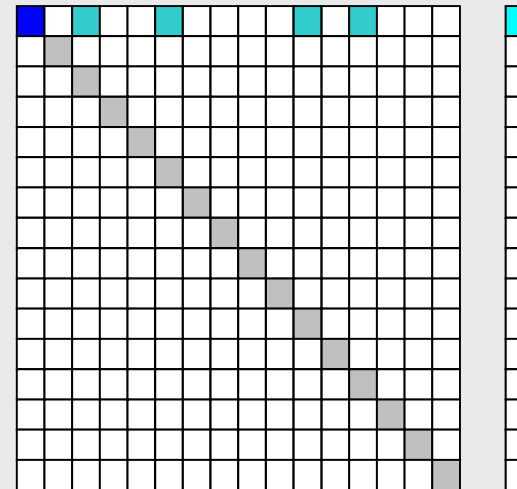
/*
// +-----+
// | BOUNDARY conditions |
// +-----+
*/

/* X=Xmin */
  i=0;
  jS= Index[i];
  AMat[jS]= 0.0;
  Diag[i ]= 1.0;
  Rhs [i ]= 0.0;

  for (k=0;k<NPLU;k++) {
    if (Item[k]==0) {AMat[k]=0.0;
    }}

```

$T_0=0$   
 対角成分=1, 右辺=0, 非対角成分=0



# プログラム: 1d.c (6/6)

## 第一種境界条件 @x=0

```

/*
// +-----+
// | BOUNDARY conditions |
// +-----+
*/

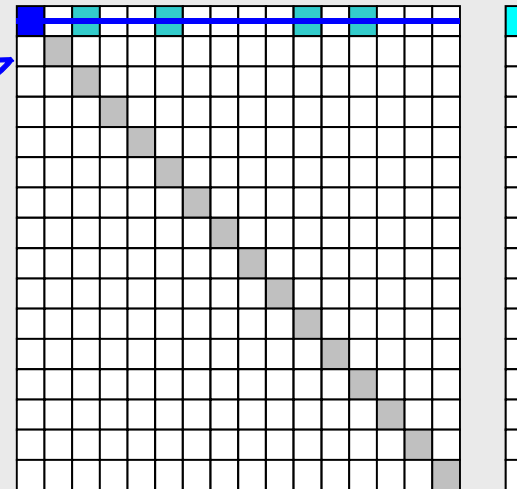
/* X=Xmin */
  i=0;
  jS= Index[i];
  AMat[jS]= 0.0;
  Diag[i ]= 1.0;
  Rhs [i ]= 0.0;

  for (k=0;k<NPLU;k++) {
    if (Item[k]==0) {AMat[k]=0.0;
    }}

```

$T_0=0$   
対角成分=1, 右辺=0, 非対角成分=0

ゼロクリア





# プログラム: 1d.c (6/6)

## 第一種境界条件 @x=0

```

/*
// +-----+
// | BOUNDARY conditions |
// +-----+
*/

/* X=Xmin */
i=0;
jS= Index[i];
AMat[jS]= 0.0;
Diag[i ]= 1.0;
Rhs [i ]= 0.0;

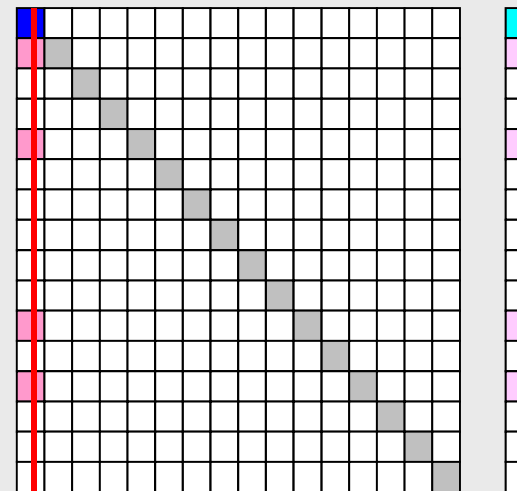
for (k=0;k<NPLU;k++) {
  if (Item[k]==0) {AMat[k]=0.0;
  }}

```

行列の対称性を保つため、第一種境界条件を適用している節点に対応する「列」を、右辺に移項して消去する(今の場合は非対角成分を0にするだけで良い)

$T_0=0$   
対角成分=1, 右辺=0, 非対角成分=0

消去, ゼロクリア



# 第一種境界条件が $T \neq 0$ の場合

```

/*
// +-----+
// | BOUNDARY conditions |
// +-----+
*/

```

行列の対称性を保つため、第一種境界条件を適用している節点に対応する「列」を、右辺に移項して消去する

```

/* X=Xmin */

```

```

    i=0;
    jS= Index[i];
    AMat[jS]= 0.0;
    Diag[i ]= 1.0;
    Rhs [i ]= PHImin;

```

```

    for (j=1; i<N; i++) {
        for (k=Index[j]; k<Index[j+1]; k++) {
            if (Item[k]==0) {
                Rhs [j]= Rhs[j] - AMat[k]*PHImin;
                AMat[k]= 0.0;
            }
        }
    }

```

$$Diag_j \phi_j + \sum_{k=Index[j]}^{Index[j+1]-1} AMat_k \phi_{Item[k]} = Rhs_j$$

# 第一種境界条件が $T \neq 0$ の場合

```

/*
// +-----+
// | BOUNDARY conditions |
// +-----+
*/

```

行列の対称性を保つため、第一種境界条件を適用している節点に対応する「列」を、右辺に移項して消去する

```

/* X=Xmin */
  i=0;
  jS= Index[i];
  AMat[jS]= 0.0;
  Diag[i ]= 1.0;
  Rhs [i ]= PHImin;

```

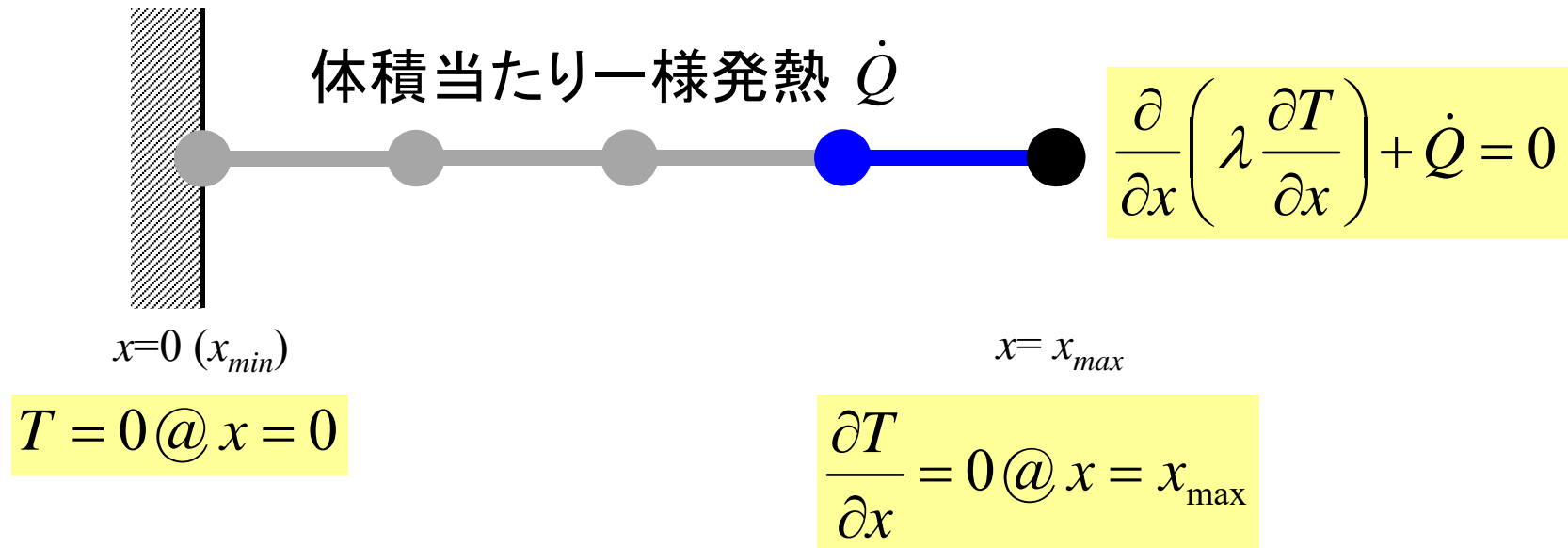
$$\begin{aligned}
 & \text{Diag}_j \phi_j + \sum_{k=\text{Index}[j], k \neq k_s}^{\text{Index}[j+1]-1} \text{AMat}_k \phi_{\text{Item}[k]} \\
 &= \text{Rhs}_j - \text{AMat}_{k_s} \phi_{\text{Item}[k_s]} \\
 &= \text{Rhs}_j - \text{AMat}_{k_s} \phi_{\min} \quad \text{where } \text{Item}[k_s] = 0
 \end{aligned}$$

```

for (j=1; j<N; j++) {
  for (k=Index[j]; k<Index[j+1]; k++) {
    if (Item[k]==0) {
      Rhs [j]= Rhs[j] - AMat[k]*PHImin;
      AMat[k]= 0.0;
    }
  }
}

```

## 第二種境界条件 (断熱)



$$\int_S \bar{q} [N]^T dS = \bar{q} A|_{x=L} = \bar{q} A \begin{Bmatrix} 0 \\ 1 \end{Bmatrix}, \quad \bar{q} = -\lambda \frac{dT}{dx} \quad \text{表面熱流束}$$



$$\frac{\partial T}{\partial x} = 0 @ x = x_{max}$$

断熱境界条件が成立するため,  $\bar{q} = 0$   
 従ってこの項の寄与は無い。  
 断熱境界条件は何もしなくても成立  
 → 自然境界条件

# 前処理付き共役勾配法

## Preconditioned Conjugate Gradient Method (CG)

```

Compute  $\mathbf{r}^{(0)} = \mathbf{b} - [\mathbf{A}]\mathbf{x}^{(0)}$ 
for i= 1, 2, ...
  solve  $[\mathbf{M}]\mathbf{z}^{(i-1)} = \mathbf{r}^{(i-1)}$ 
   $\rho_{i-1} = \mathbf{r}^{(i-1)} \mathbf{z}^{(i-1)}$ 
  if i=1
     $\mathbf{p}^{(1)} = \mathbf{z}^{(0)}$ 
  else
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $\mathbf{p}^{(i)} = \mathbf{z}^{(i-1)} + \beta_{i-1} \mathbf{p}^{(i-1)}$ 
  endif
   $\mathbf{q}^{(i)} = [\mathbf{A}]\mathbf{p}^{(i)}$ 
   $\alpha_i = \rho_{i-1} / \mathbf{p}^{(i)} \mathbf{q}^{(i)}$ 
   $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \alpha_i \mathbf{p}^{(i)}$ 
   $\mathbf{r}^{(i)} = \mathbf{r}^{(i-1)} - \alpha_i \mathbf{q}^{(i)}$ 
  check convergence  $|\mathbf{r}|$ 
end

```

前処理: 対角スケーリング

# 対角スケーリング, 点ヤコビ前処理

- 前処理行列として, もとの行列の対角成分のみを取り出した行列を前処理行列  $[M]$  とする。
  - 対角スケーリング, 点ヤコビ (point-Jacobi) 前処理

$$[M] = \begin{bmatrix} D_1 & 0 & \dots & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ \dots & & \dots & & \dots \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & \dots & 0 & D_N \end{bmatrix}$$

- **solve  $[M]z^{(i-1)} = r^{(i-1)}$**  という場合に逆行列を簡単に求めることができる。

# CGソルバー(1/6)

```

/*
// +-----+
// | CG iterations |
// +-----+
*/

    R = 0;
    Z = 1;
    Q = 1;
    P = 2;
    DD= 3;

    for (i=0; i<N; i++) {
        W[DD][i]= 1.0 / Diag[i];
    }

```

```

W[0][i]= W[R][i]   ⇒ {r}
W[1][i]= W[Z][i]   ⇒ {z}
W[1][i]= W[Q][i]   ⇒ {q}
W[2][i]= W[P][i]   ⇒ {p}
W[3][i]= W[DD][i]  ⇒ 1/{D}

```

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for i= 1, 2, ...
    solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
     $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
    if i=1
         $p^{(1)} = z^{(0)}$ 
    else
         $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
         $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
    endif
     $q^{(i)} = [A]p^{(i)}$ 
     $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$ 
     $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
     $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
    check convergence |r|
end

```

# CGソルバー (1/6)

```

/*
// +-----+
// | CG iterations |
// +-----+
*/

    R = 0;
    Z = 1;
    Q = 1;
    P = 2;
    DD= 3;

    for (i=0; i<N; i++) {
        W[DD][i]= 1.0 / Diag[i];
    }

```

対角成分の逆数(前処理用)  
その都度、除算をすると効率が  
悪いため、予め配列に格納

$W[0][i] = W[R][i] \Rightarrow \{r\}$   
 $W[1][i] = W[Z][i] \Rightarrow \{z\}$   
 $W[1][i] = W[Q][i] \Rightarrow \{q\}$   
 $W[2][i] = W[P][i] \Rightarrow \{p\}$   
 $W[3][i] = W[DD][i] \Rightarrow 1/\{D\}$



# CGソルバー(2/6)

```

/*
//-- {r0} = {b} - [A]{xini} |
*/
    for (i=0; i<N; i++) {
        W[R][i] = Diag[i]*U[i];
        for (j=Index[i]; j<Index[i+1]; j++) {
            W[R][i] += AMat[j]*U[Item[j]];
        }
    }

    BNorm2 = 0.0;
    for (i=0; i<N; i++) {
        BNorm2 += Rhs[i] * Rhs[i];
        W[R][i] = Rhs[i] - W[R][i];
    }

```

**BNRM2 = |b|<sup>2</sup>**  
**あとで収束判定に使用**

**Compute  $r^{(0)} = b - [A]x^{(0)}$**

```

for i = 1, 2, ...
    solve [M] z(i-1) = r(i-1)
    ρi-1 = r(i-1) z(i-1)
    if i = 1
        p(1) = z(0)
    else
        βi-1 = ρi-1 / ρi-2
        p(i) = z(i-1) + βi-1 p(i-1)
    endif
    q(i) = [A] p(i)
    αi = ρi-1 / p(i) q(i)
    x(i) = x(i-1) + αi p(i)
    r(i) = r(i-1) - αi q(i)
    check convergence |r|
end

```

# CGソルバー (3/6)

```

for (iter=1; iter<=IterMax; iter++) {
  /*
  //-- {z} = [Minv]{r}
  */
  for (i=0; i<N; i++) {
    W[Z][i] = W[DD][i] * W[R][i];
  }

  /*
  //-- RHO = {r}{z}
  */
  Rho = 0.0;
  for (i=0; i<N; i++) {
    Rho += W[R][i] * W[Z][i];
  }
}

```

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
  solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
  if  $i = 1$ 
     $p^{(1)} = z^{(0)}$ 
  else
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
  endif
   $q^{(i)} = [A]p^{(i)}$ 
   $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$ 
   $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
   $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
  check convergence  $|r|$ 
end

```

# CGソルバー(4/6)

```

/*
//-- {p} = {z} if      ITER=1
//  BETA= RHO / RHO1  otherwise
*/
if(iter == 1){
  for(i=0;i<N;i++){
    W[P][i] = W[Z][i];
  }
}else{
  Beta = Rho / Rho1;
  for(i=0;i<N;i++){
    W[P][i] = W[Z][i] + Beta*W[P][i];
  }
}

/*
//-- {q} = [A] {p}
*/
for(i=0;i<N;i++){
  W[Q][i] = Diag[i] * W[P][i];
  for(j=Index[i];j<Index[i+1];j++){
    W[Q][i] += AMat[j]*W[P][Item[j]];
  }
}

```

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
  solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
  if  $i=1$ 
     $p^{(1)} = z^{(0)}$ 
  else
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
  endif
   $q^{(i)} = [A]p^{(i)}$ 
   $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$ 
   $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
   $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
  check convergence  $|r|$ 
end

```

# CGソルバー (5/6)

```

/*
//-- ALPHA= RHO / {p} {q}
*/
C1 = 0.0;
for (i=0; i<N; i++) {
    C1 += W[P][i] * W[Q][i];
}

Alpha = Rho / C1;

/*
//-- {x} = {x} + ALPHA*{p}
//   {r} = {r} - ALPHA*{q}
*/
for (i=0; i<N; i++) {
    U[i] += Alpha * W[P][i];
    W[R][i] -= Alpha * W[Q][i];
}

```

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for i = 1, 2, ...
    solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
     $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
    if i=1
         $p^{(1)} = z^{(0)}$ 
    else
         $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
         $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
    endif
     $q^{(i)} = [A]p^{(i)}$ 
     $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$ 
     $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
     $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
    check convergence |r|
end

```

# CGソルバー(6/6)

```

DNorm2 = 0.0;
for (i=0; i<N; i++) {
    DNorm2 += W[R][i] * W[R][i];
}
Resid = sqrt(DNorm2/BNorm2);

if((iter)%1000 == 0) {
    printf("%8d%s%16.6e\n", iter, "
        iters, RESID=", Resid);
}
if(Resid <= Eps) {ierr = 0; break;}
Rho1 = Rho;  $\rho_{i-2}$ 
}

```

$$\text{Resid} = \sqrt{\frac{\text{DNorm2}}{\text{BNorm2}}} = \frac{|r|}{|b|} = \frac{|Ax - b|}{|b|} \leq \text{Eps}$$

制御ファイル input.dat

```

4          NE (要素数)
1.0  1.0  1.0  1.0   $\Delta x$  (要素長さL), Q, A,  $\lambda$ 
100       反復回数
1.e-8     CG法の反復打切誤差 Eps

```

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for i = 1, 2, ...
    solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
     $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
    if i=1
         $p^{(1)} = z^{(0)}$ 
    else
         $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
         $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
    endif
     $q^{(i)} = [A]p^{(i)}$ 
     $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$ 
     $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
     $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
    check convergence |r|
end

```

$$Ax = b \Rightarrow \alpha Ax = \alpha b$$

$$r = b - Ax \Rightarrow R = \alpha b - \alpha Ax = \alpha r$$

# 有限要素法の処理: プログラム

- 初期化
  - 制御変数読み込み
  - 座標読み込み⇒要素生成(N:節点数, NE:要素数)
  - 配列初期化(全体マトリクス, 要素マトリクス)
  - 要素⇒全体マトリクスマッピング(Index, Item)
- マトリクス生成
  - 要素単位の処理(do icel= 1, NE)
    - 要素マトリクス計算
    - 全体マトリクスへの重ね合わせ
  - 境界条件の処理
- 連立一次方程式
  - 共役勾配法(CG)

# より精度をあげるには？

- メッシュを細かくする

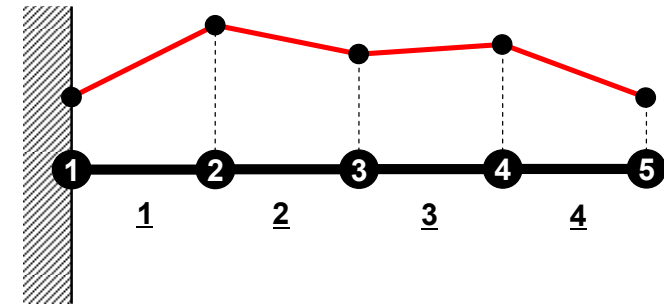
NE=8, dx=12.5

8 iters, RESID= 2.822910E-16 U(N)= 1.953586E-01

### DISPLACEMENT

1	0.000000E+00	-0.000000E+00
2	1.101928E-02	1.103160E-02
3	2.348034E-02	2.351048E-02
4	3.781726E-02	3.787457E-02
5	5.469490E-02	5.479659E-02
6	7.520772E-02	7.538926E-02
7	1.013515E-01	1.016991E-01
8	1.373875E-01	1.381746E-01
9	1.953586E-01	1.980421E-01

$$\therefore u = \frac{F}{EA_1} [\log(A_1 x + A_2) - \log(A_2)]$$



NE=20, dx=5

20 iters, RESID= 5.707508E-15 U(N)= 1.975734E-01

### DISPLACEMENT

1	0.000000E+00	-0.000000E+00
2	4.259851E-03	4.260561E-03
3	8.719160E-03	8.720685E-03
4	1.339752E-02	1.339999E-02
.....		
17	1.145876E-01	1.146641E-01
18	1.295689E-01	1.296764E-01
19	1.473466E-01	1.475060E-01
20	1.692046E-01	1.694607E-01
21	1.975734E-01	1.980421E-01

# より精度をあげるには？

- メッシュを細かくする
- 高次の補間関数(形状関数)を使用する
  - 高次要素
  - 線形要素, 一次要素は低次要素と呼ばれる
- $n$ 次微分係数の連続性を保証する定式化を適用する
  - $C^n$ 連続性



# より精度をあげるには？

- メッシュを細かくする
- 高次の補間関数(形状関数)を使用する
- $n$ 次微分係数の連続性を保証する定式化を適用する
  - $C^n$ 連続性
- これまで紹介してきたのは:
  - 一次要素(線形要素)
    - 区分的一次近似(Piecewise Linear)
  - $C^0$ 連続
    - 従属変数(のみ)が要素境界で連続
- **高次要素の例:**
  - **二次要素: 曲線の近似により適している**
    - 要素内で二次関数的な分布
  - $C^0$ 連続