

1D-FEM in C: Steady State Heat Conduction

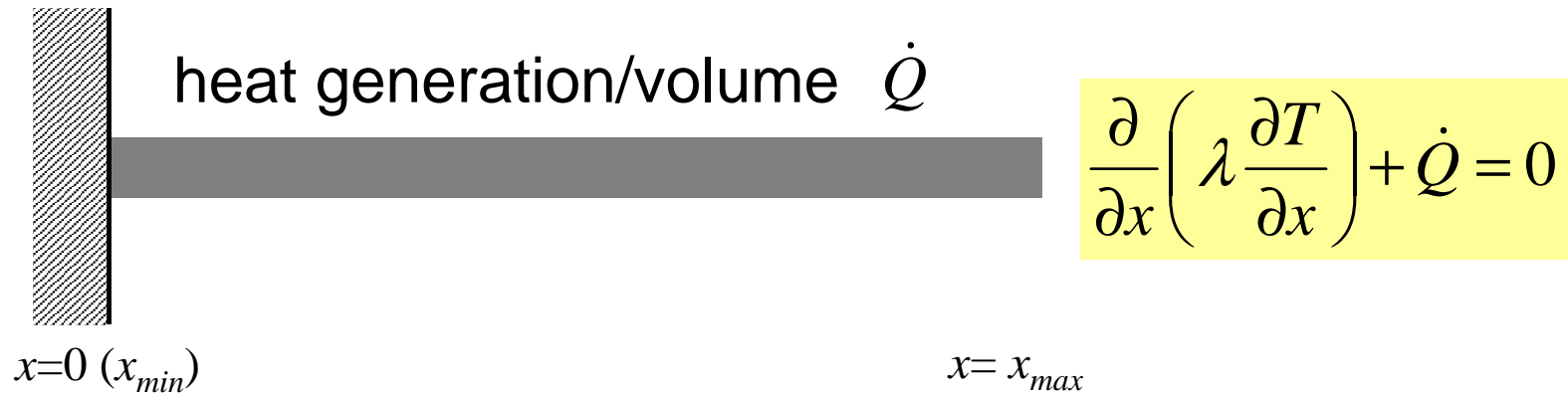
Kengo Nakajima
Information Technology Center
The University of Tokyo

- 1D-code for Steady State Heat Conduction Problems by Galerkin FEM
- Sparse Linear Solver
 - Conjugate Gradient Method
 - Preconditioning
- Storage of Sparse Matrices
- Program

Keywords

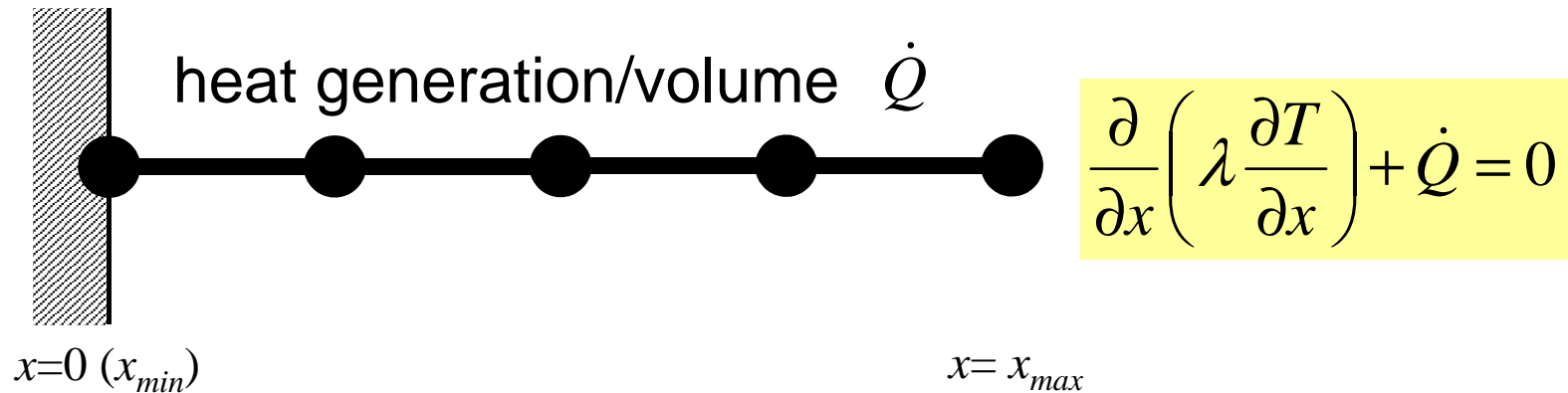
- 1D Steady State Heat Conduction Problems
- Galerkin Method
- Linear Element
- Preconditioned Conjugate Gradient Method

1D Steady State Heat Conduction



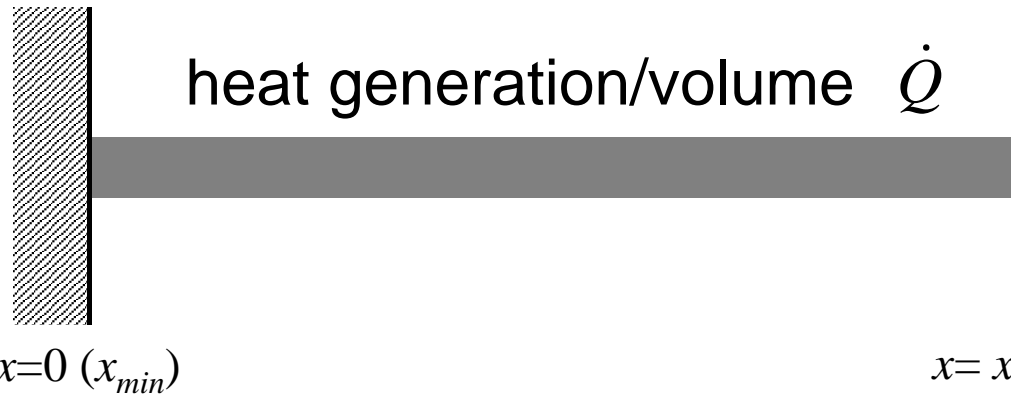
- **Uniform: Sectional Area: A , Thermal Conductivity: λ**
- Heat Generation Rate/Volume/Time [$QL^{-3}T^{-1}$] \dot{Q}
- Boundary Conditions
 - $x=0$: $T=0$ (Fixed Temperature)
 - $x=x_{max}$: $\frac{\partial T}{\partial x} = 0$ (Insulated)

1D Steady State Heat Conduction



- **Uniform: Sectional Area: A , Thermal Conductivity: λ**
- Heat Generation Rate/Volume/Time [$QL^{-3}T^{-1}$] \dot{Q}
- Boundary Conditions
 - $x=0$: $T=0$ (Fixed Temperature)
 - $x=x_{max}$: $\frac{\partial T}{\partial x} = 0$ (Insulated)

Analytical Solution



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

$$T = 0 @ x = 0$$

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

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

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

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

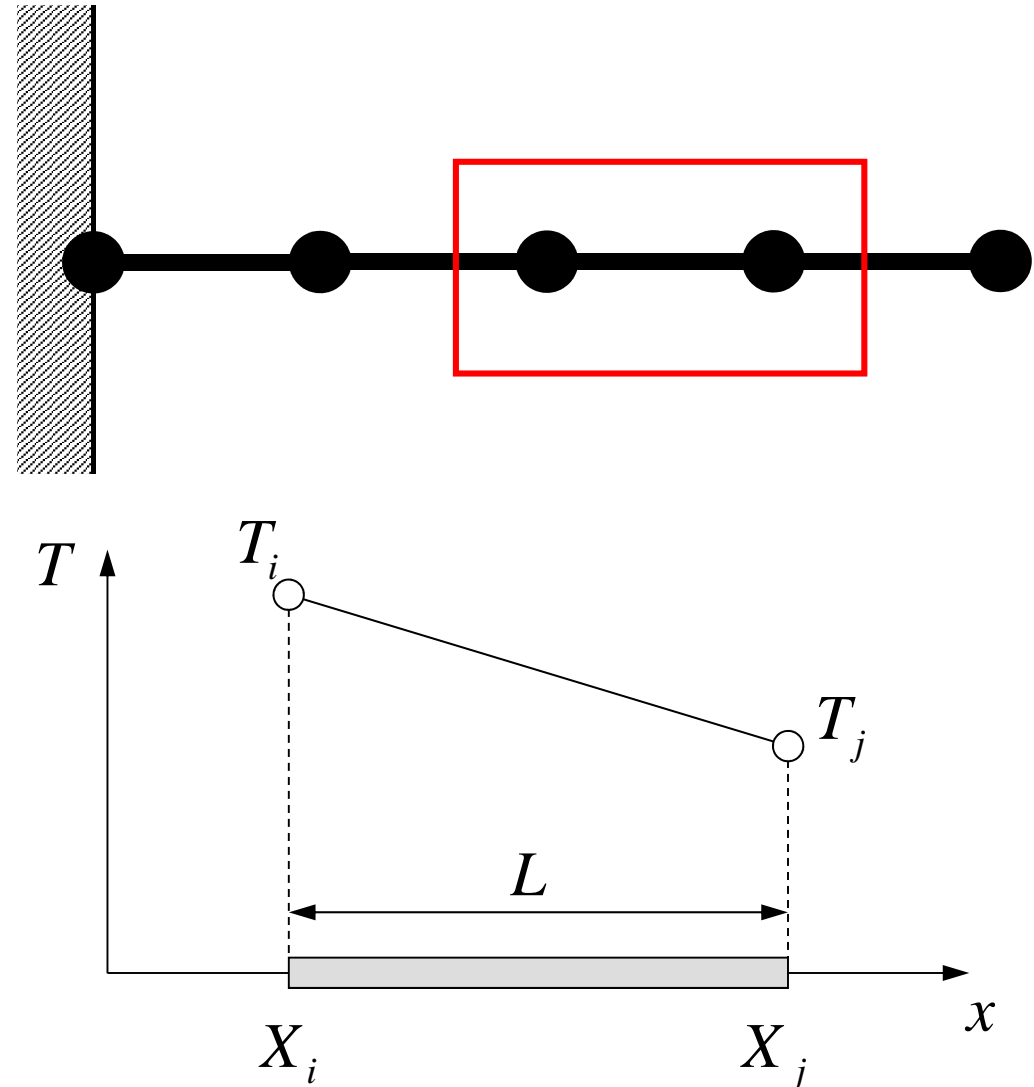
$$\therefore T = -\frac{1}{2\lambda}\dot{Q}x^2 + \frac{\dot{Q}x_{max}}{\lambda}x$$

1D Linear Element (1/4)

一次元線形要素

- 1D Linear Element
 - Length = L
 - Node (Vertex) (節点)
 - Element (要素)
 - T_i Temperature at i
 - T_j Temperature at j
 - Temperature T on each element is linear function of x (Piecewise Linear):

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



1D Linear Element (1/4)

一次元線形要素

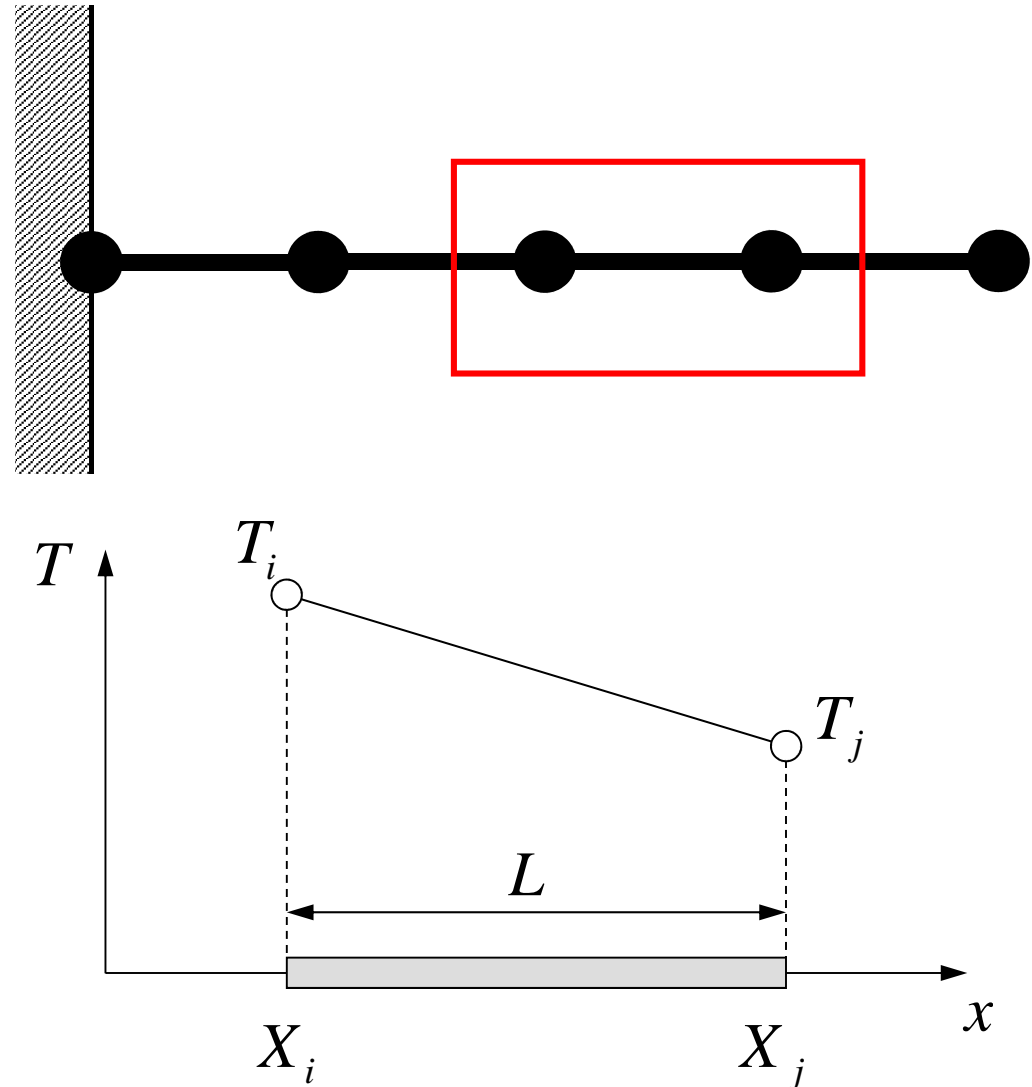
- 1D Linear Element

- Length = L

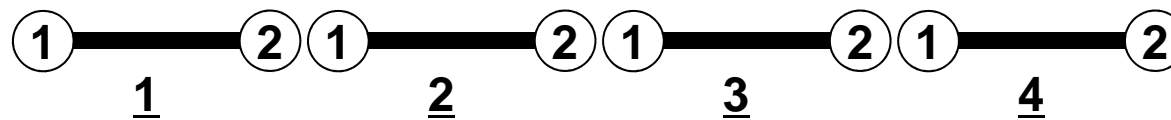
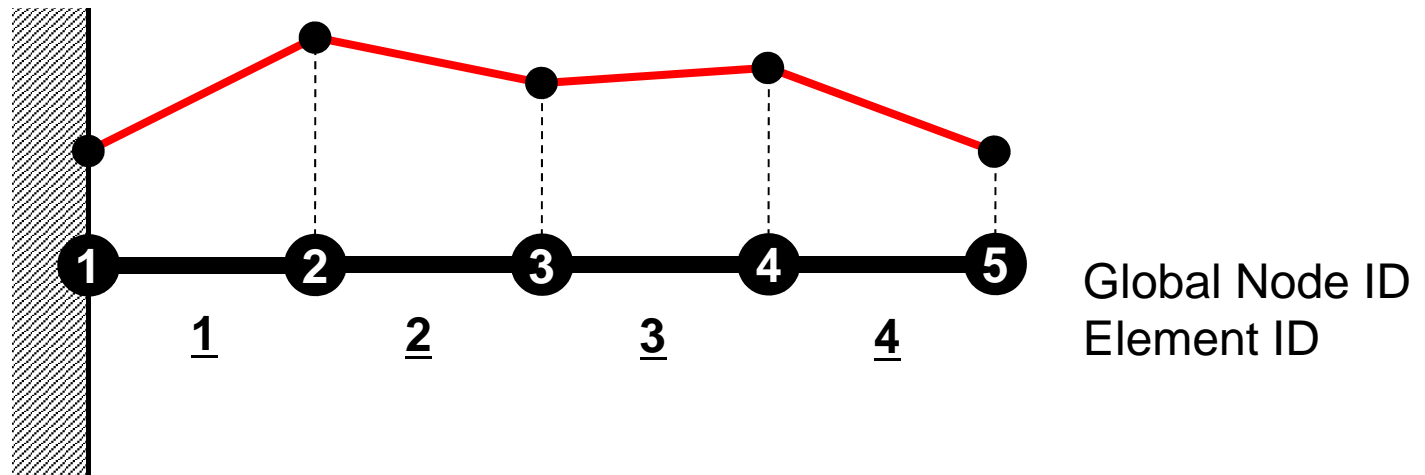
- Node (Vertex)
- Element

- T_i Temperature at i
- T_j Temperature at j
- Temperature T on each element is linear function of x (Piecewise Linear):

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



Piecewise Linear



Local Node ID
for each elem.

Gradient of temperature is constant in each element (might be discontinuous at each “node”)

1D Linear Elem.: Shape Function (2/4)

- Coef's are calculated based on info. at each node

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

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

- Coefficients:

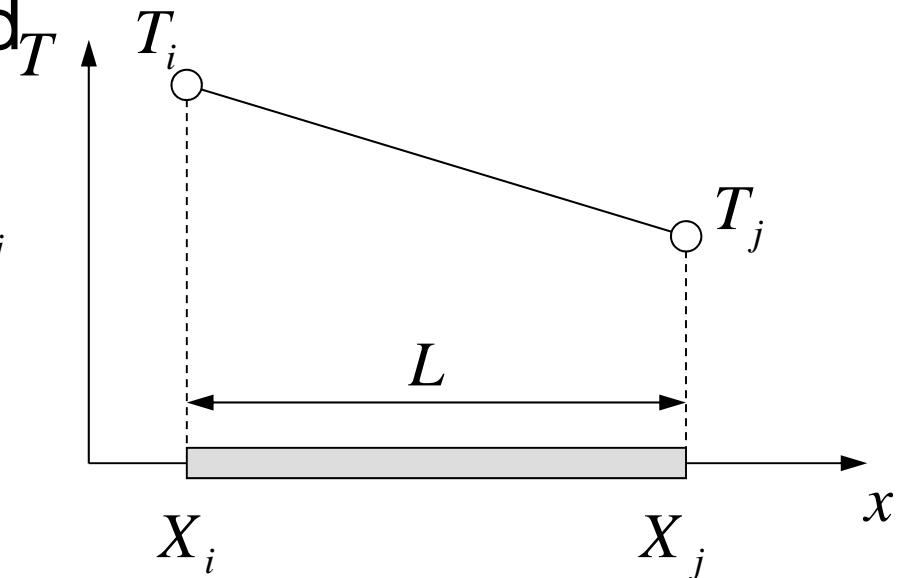
$$\alpha_1 = \frac{T_i X_j - T_j X_i}{L}, \quad \alpha_2 = \frac{T_j - T_i}{L}$$

- T can be written as follows, according to T_i and T_j :

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

N_i, N_j

Shape Function (形状関数) or
Interpolation Function (内挿関数), function of x (only)

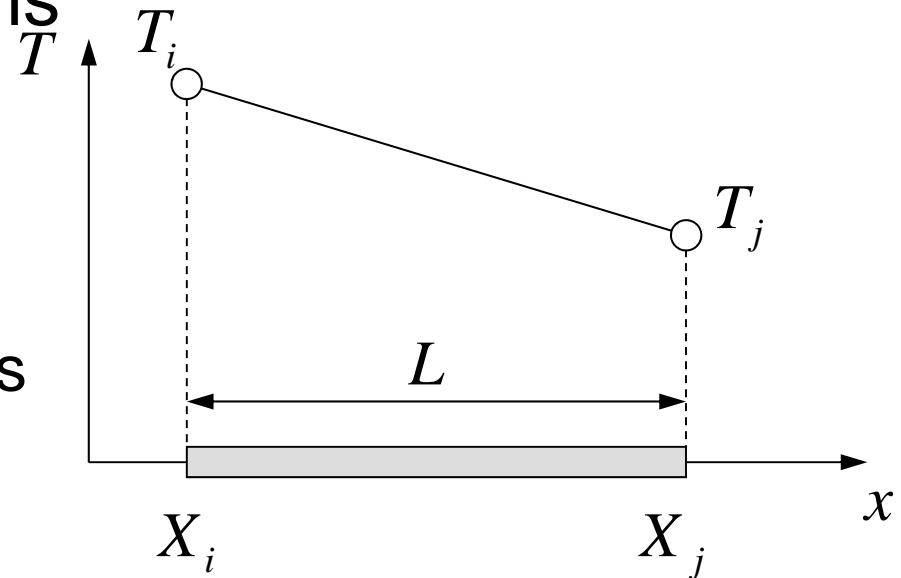


1D Linear Elem.: Shape Function (3/4)

- Number of Shape Functions = Number of Vertices of Each Element

- N_i : Function of Position
- A kind of Test/Trial Functions

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



- Linear combination of shape functions provides temperature “in” each element
 - Coef’s (unknowns): Temperature at each node

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

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

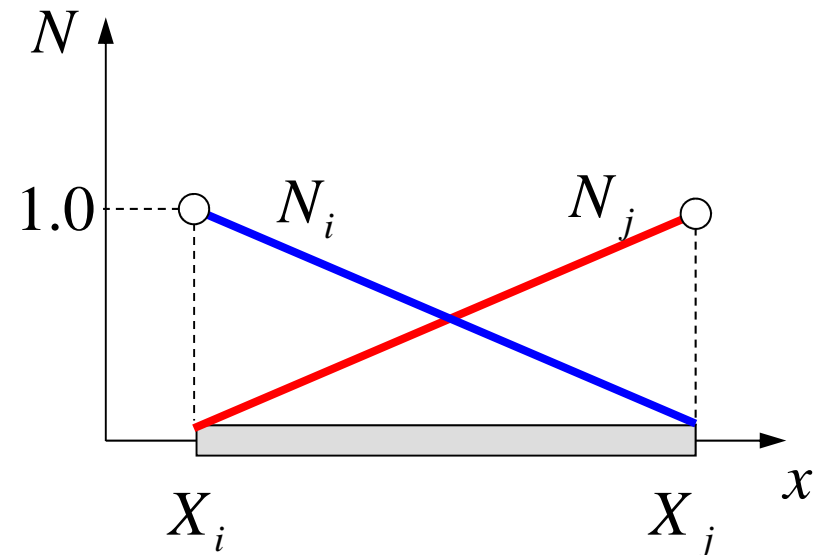
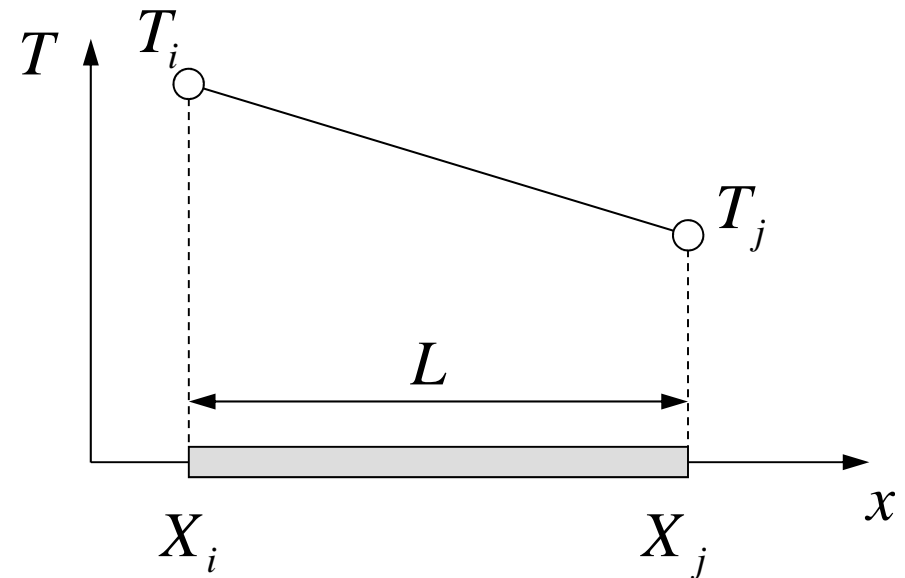
Ψ_i Trial/Test Function (known function of position, defined in domain and at boundary. “Basis” in linear algebra.

a_i Coefficients (unknown)

1D Linear Elem.: Shape Function (4/4)

- Value of N_i
 - =1 at one of the nodes in element
 - =0 on other nodes

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



Galerkin Method (1/4)

- Governing Equation for 1D Steady State Heat Conduction Problems (Uniform λ):

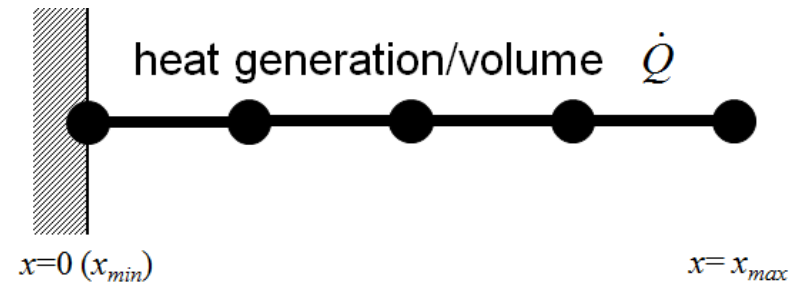
$$\lambda \left(\frac{d^2 T}{dx^2} \right) + \dot{Q} = 0$$

$$T = [N] \{ \phi \}$$

Distribution of temperature in each element (matrix form), ϕ : Temperature at each node

- Following integral equation is obtained at each element by Galerkin method, where $[N]$'s are also weighting functions:

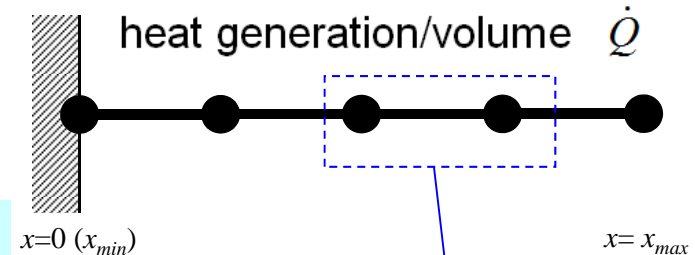
$$\int_V [N]^T \left\{ \lambda \left(\frac{d^2 T}{dx^2} \right) + \dot{Q} \right\} dV = 0$$



Galerkin Method (2/4)

- Green's Theorem (1D)

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

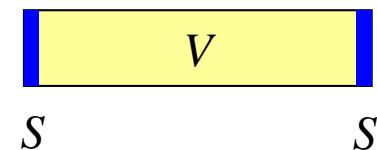


- Apply this to the 1st part of eqn with 2nd-order diff.:

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

- Consider the following terms:

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



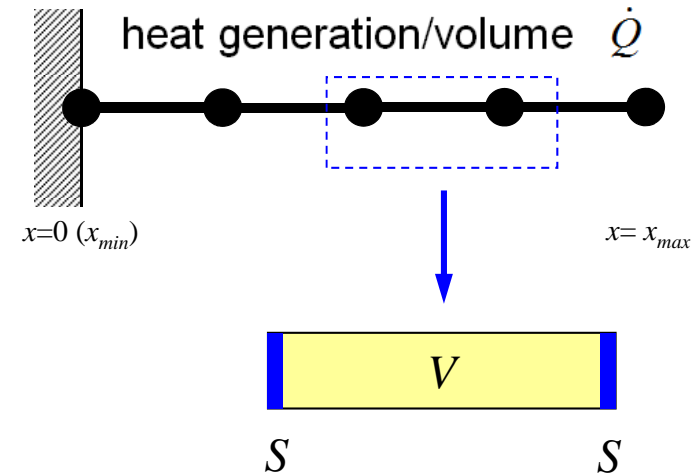
: Heat flux at element surface [QL⁻²T⁻¹]

Galerkin Method (3/4)

- Finally, following eqn is obtained by considering heat generation term \dot{Q} :

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

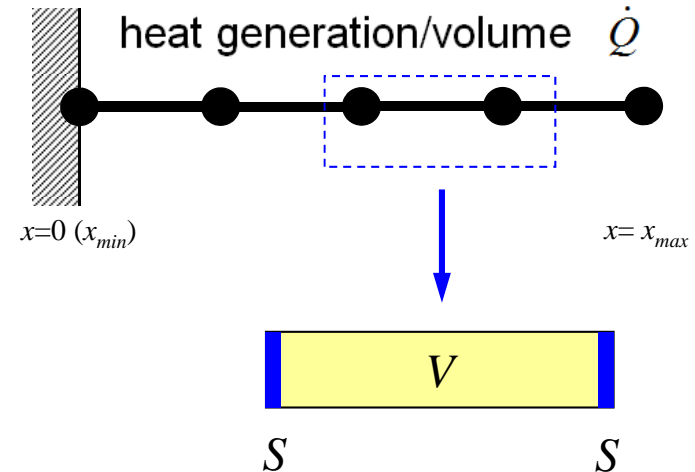
$$-\int_S \bar{q}[N]^T dS + \int_V \dot{Q}[N]^T dV = 0$$



- This is called “weak form (弱形式)”. Original PDE consists of terms with 2nd-order diff., but this “weak form” only includes 1st-order diff by Green’s theorem.
 - Requirements for shape functions are “weaker” in “weak form”. Linear functions can describe effects of 2nd-order differentiation.

Galerkin Method (4/4)

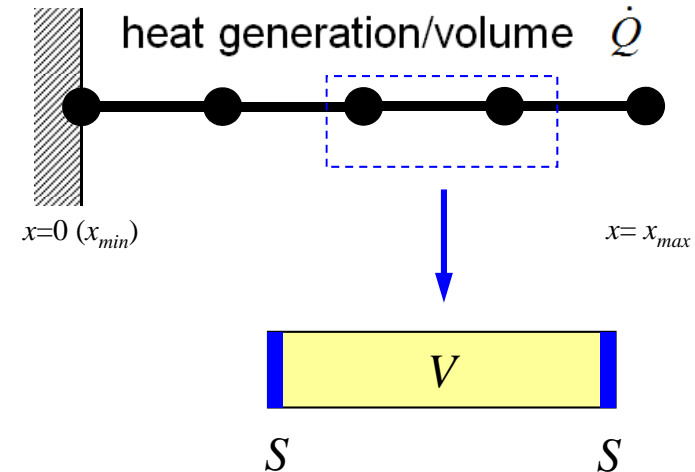
$$\begin{aligned}
 & - \int_V \lambda \left(\frac{d[N]^T}{dx} \frac{d[N]}{dx} \right) dV \cdot \{\phi\} \\
 & \boxed{- \int_S \bar{q} [N]^T dS} + \int_V \dot{Q} [N]^T dV = 0
 \end{aligned}$$



- These terms coincide at element boundaries and disappear. Finally, only terms on the domain boundaries remain.

Weak Form and Boundary Conditions

- Value of dependent variable is defined (Dirichlet)
 - Weighting Function = 0
 - Principal B.C. (Boundary Condition) (第一種境界条件)
 - Essential B.C. (基本境界条件)
- Derivatives of Unknowns (Neumann)
 - Naturally satisfied in weak form
 - Secondary B.C. (第二種境界条件)
 - Natural B.C (自然境界条件)



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

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

$$\text{where } \bar{q} = -\lambda \frac{dT}{dx}$$

Weak Form with B.C.: on each elem.

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

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

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

Integration over Each Element: $[k]$

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

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

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

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

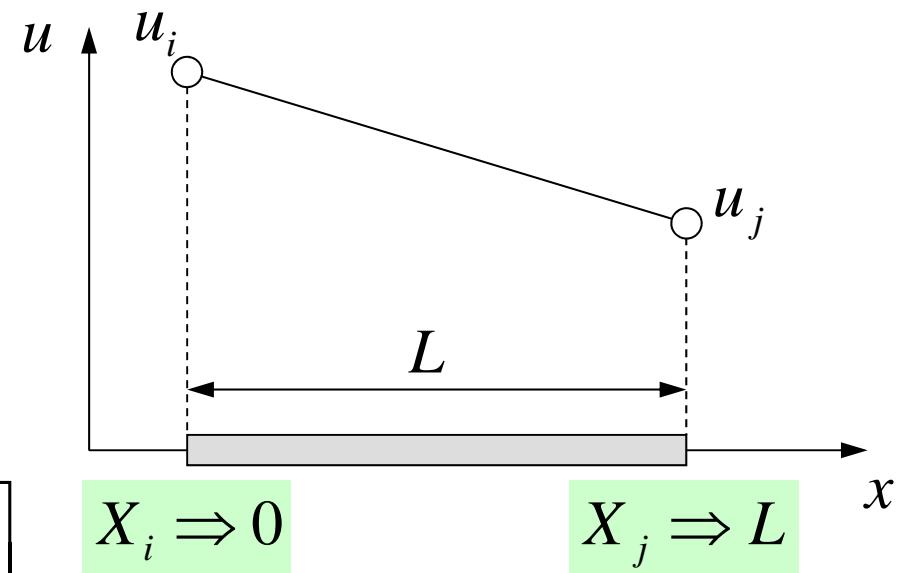
2x1 matrix

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: Sectional Area

L: Length



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

Integration over Each Element: $\{f\}$ (1/2)

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

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

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

Heat Generation
(Volume)



A: Sectional Area

L: Length

Integration over Each Element: $\{f\}$ (2/2)

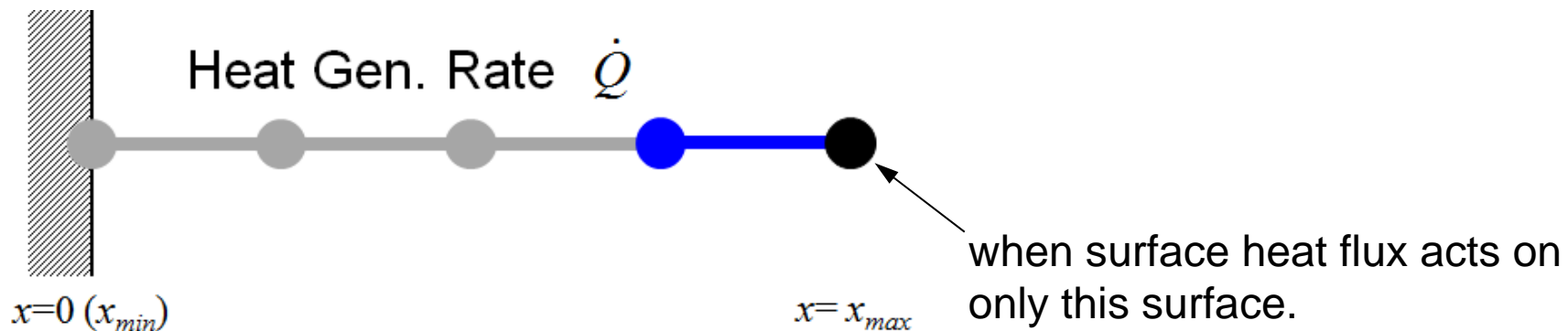
$$N_i = \left(\frac{X_j - x}{L} \right), \quad N_j = \left(\frac{x - X_i}{L} \right) \quad \frac{dN_i}{dx} = \left(\frac{-1}{L} \right), \quad \frac{dN_j}{dx} = \left(\frac{1}{L} \right)$$

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

Heat Generation
(Volume)

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

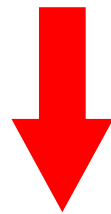
Surface Heat Flux



Global Equations

- Accumulate Element Equations:

$$[k]^{(e)} \{\phi\}^{(e)} = \{f\}^{(e)} \quad \text{Element Matrix, Element Equations}$$



$$[K] \cdot \{\Phi\} = \{F\} \quad \text{Global Matrix, Global Equations}$$

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

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

This is the final linear equations
(global equations) to be solved.

ECSS2016 System

Creating Directory

```
>$ cd Documents  
>$ mkdir 2017Winter your favorite name  
>$ cd 2017Winter
```

This is your “top” directory, and is called **<\$E-TOP>** in this class.

1D Code for Steady-State Heat Conduction Problems

```
>$ cd <$E-TOP>  
>$ (download http://nkl.cc.u-tokyo.ac.jp/files/fem-c.tar)  
>$ tar xvf fem-c.tar  
>$ cd fem-c/1d
```

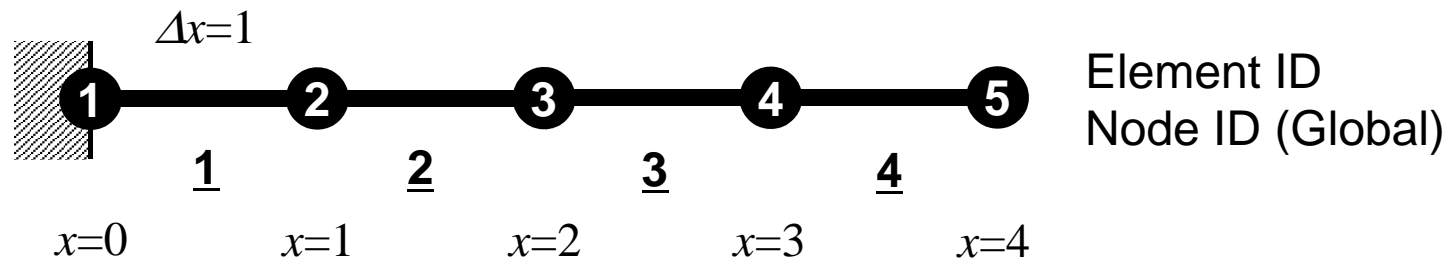
Compile & GO !

```
>$ cd <${E-TOP}>/fem1-c/1d
>$ cc -O 1d.c
>$ ./a.out
```

Control Data input.dat

```
4
1.0 1.0 1.0 1.0
100
1.e-8
```

NE (Number of Elements)
 Δx (Length of Each Elem.: \mathbf{L}) , \mathbf{Q} , \mathbf{A} , λ
 Number of MAX. Iterations for CG Solver
 Convergence Criteria for CG Solver



Results

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

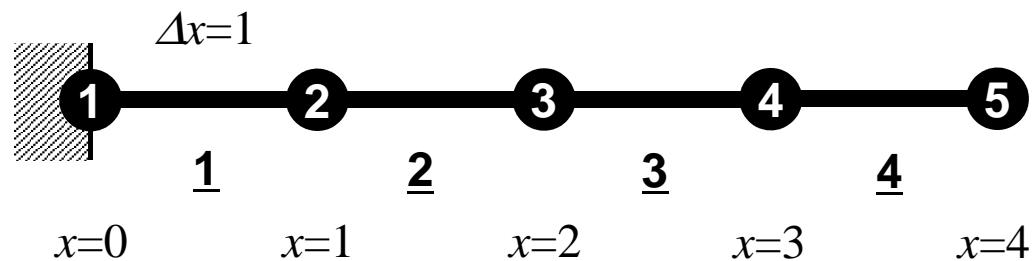
```
4 iters, RESID= 4.154074e-17
```

```
### 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

Computational

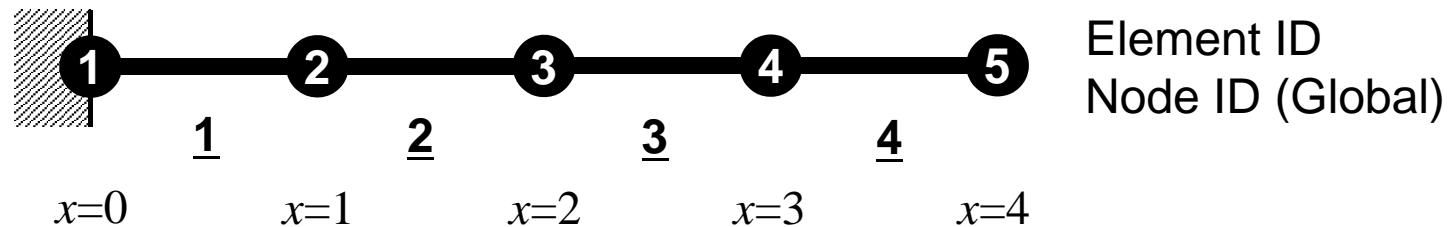
Analytical



Element ID
Node ID (Global)

Element Eqn's/Accumulation (1/3)

- 4 elements, 5 nodes



- $[k]$ and $\{f\}$ of Element-1:

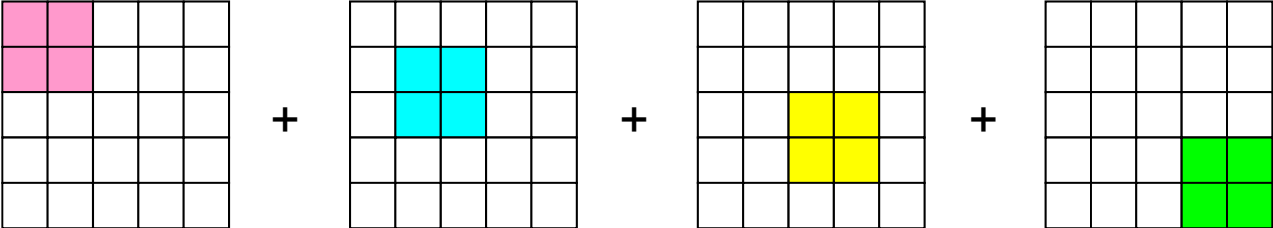
$$[k]^{(1)} = \frac{\lambda A}{L} \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \quad \{f\}^{(1)} = \frac{\dot{Q}AL}{2} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$$

- As for Element-4:

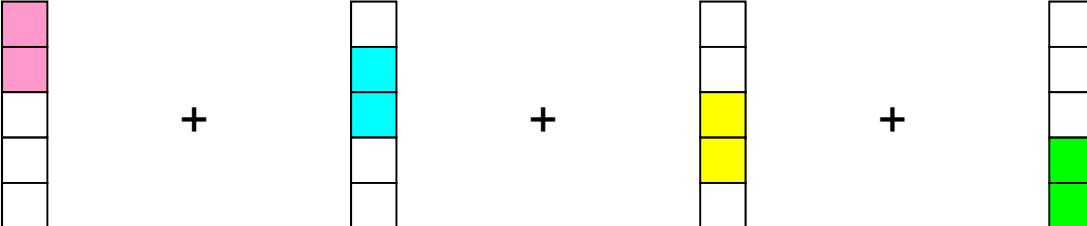
$$[k]^{(4)} = \frac{\lambda A}{L} \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \quad \{f\}^{(4)} = \frac{\dot{Q}AL}{2} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$$

Element Eqn's/Accumulation (2/3)

- Element-by-Element Accumulation:

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


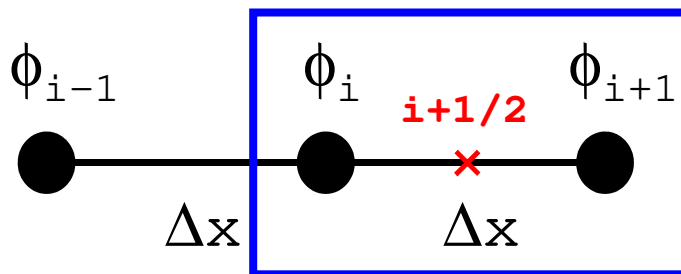
The diagram illustrates the element-by-element accumulation of the global stiffness matrix $[K]$. It shows four 4x4 grids, each representing an element matrix $[k]^{(e)}$. The first grid has a pink 2x2 block in the top-left corner. The second grid has a cyan 2x2 block in the center. The third grid has a yellow 2x2 block in the bottom-right area. The fourth grid has a green 2x2 block in the bottom-right corner. These are summed together to form the global matrix $[K]$.

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


The diagram illustrates the element-by-element accumulation of the global force vector $\{F\}$. It shows four 4x1 column vectors, each representing an element force vector $\{f\}^{(e)}$. The first vector has a pink 2x1 block in the top. The second vector has a cyan 2x1 block in the center. The third vector has a yellow 2x1 block in the bottom. The fourth vector has a green 2x1 block in the bottom-right. These are summed together to form the global force vector $\{F\}$.

2nd Order Differentiation in FDM

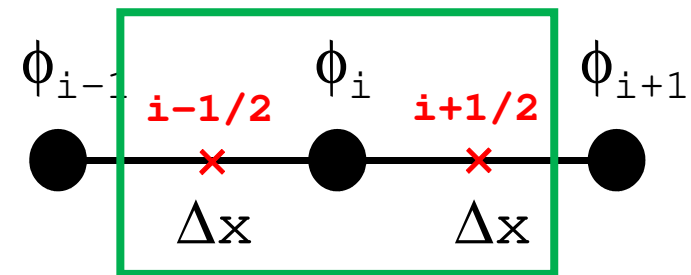
- Approximate Derivative at x (center of i and $i+1$)



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

$\Delta x \rightarrow 0$: Real Derivative

- 2nd-Order Diff. at i



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

Element-by-Element Operation

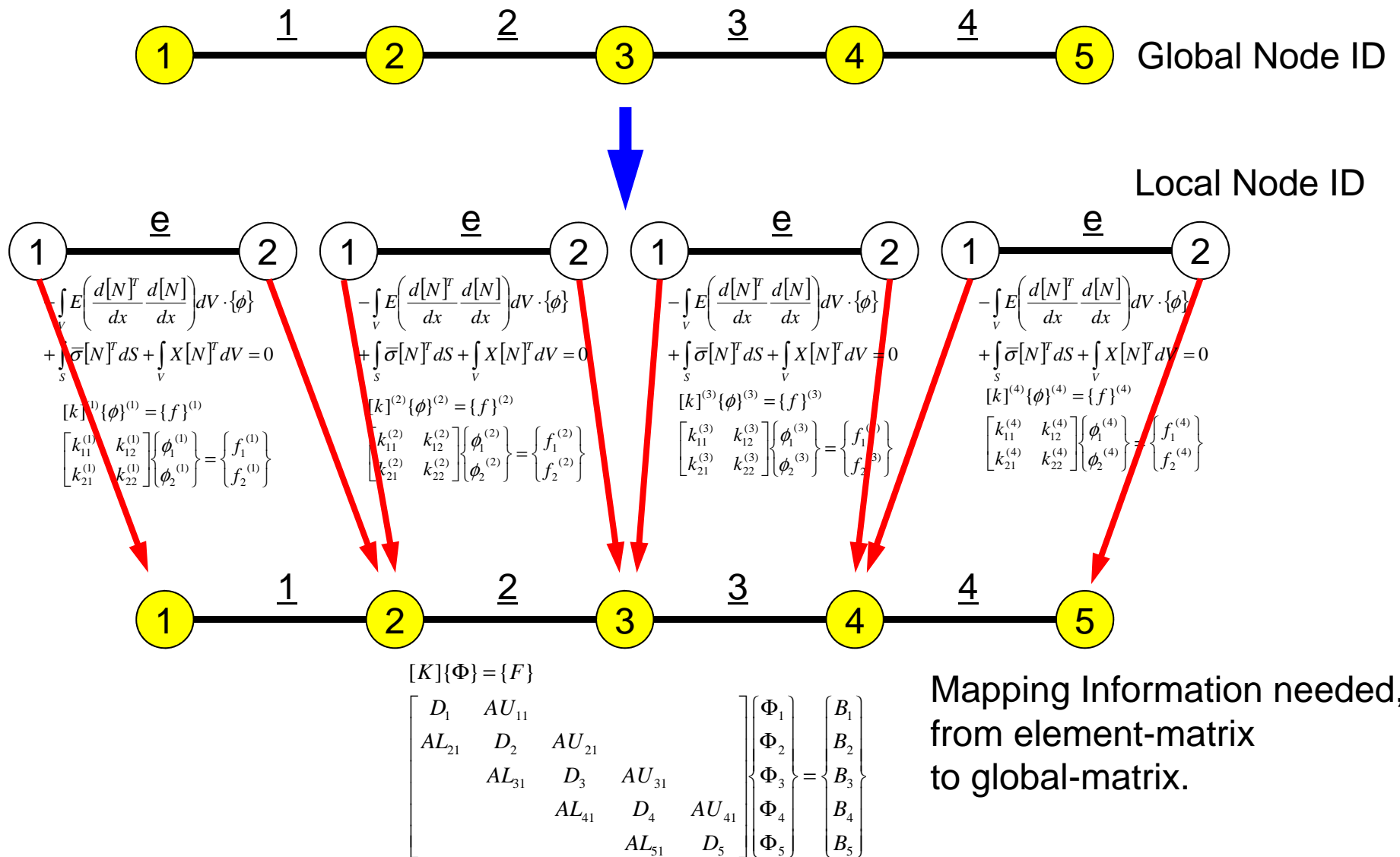
very flexible if each element has different material property, size, etc.

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

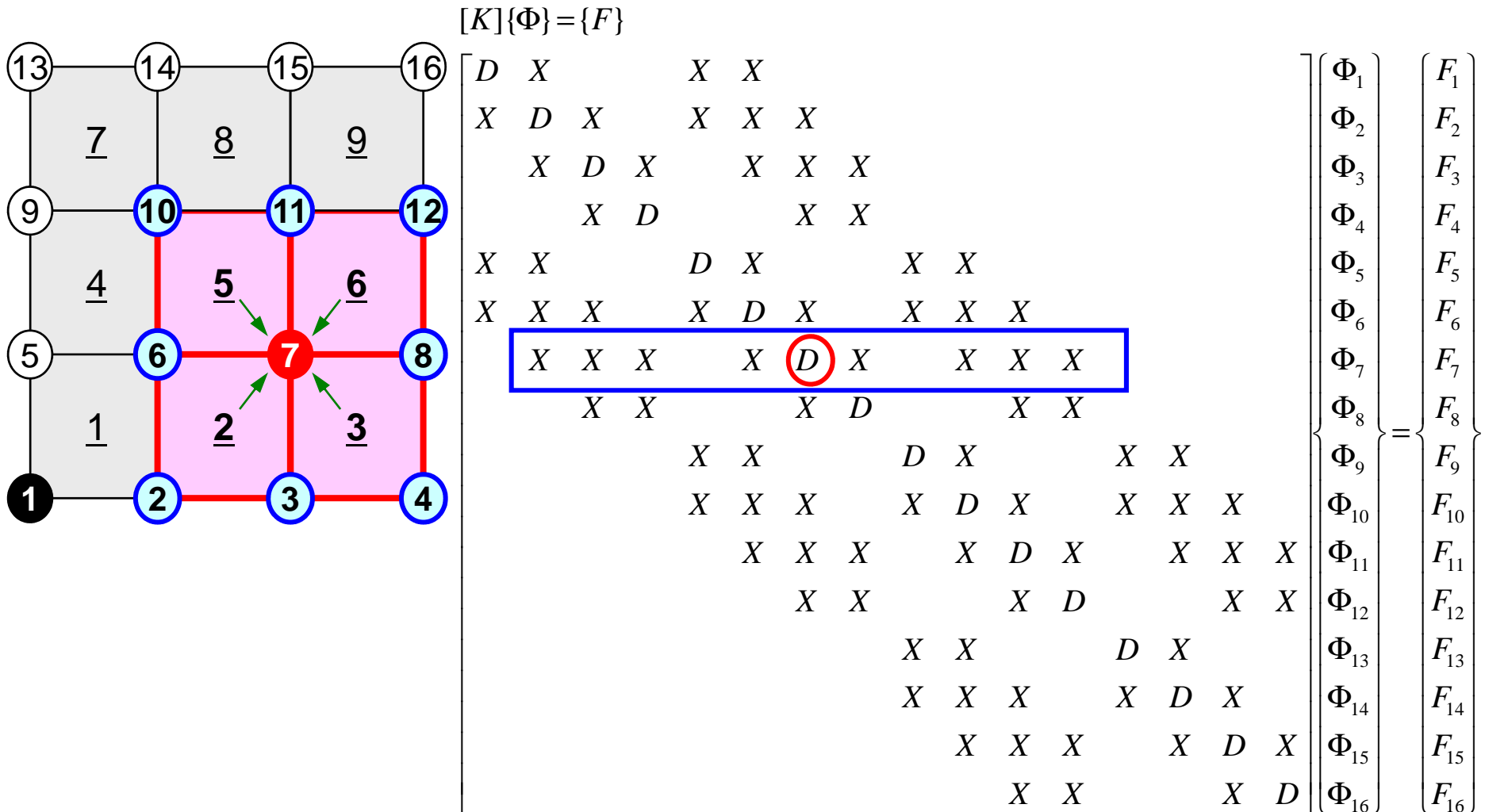
$$[K] = \sum_{e=1}^4 [k]^{(e)} =$$

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

Element/Global Operations



Accumulation to Global/overall Matrix



- 1D-code for Static Linear-Elastic Problems by Galerkin FEM
- Sparse Linear Solver
 - Conjugate Gradient Method
 - Preconditioning
- Storage of Sparse Matrices
- Program

Large-Scale Linear Equations in Scientific Applications

- Solving large-scale linear equations $\mathbf{Ax}=\mathbf{b}$ is the most important and **expensive** part of various types of scientific computing.
 - for both linear and nonlinear applications
- Various types of methods proposed & developed.
 - for dense and sparse matrices
 - classified into **direct** and **iterative** methods
- Dense Matrices: 密行列: Globally Coupled Problems
 - BEM, Spectral Methods, MO/MD (gas, liquid)
- Sparse Matrices: 疎行列: Locally Defined Problems
 - **FEM**, FDM, DEM, MD (solid), BEM w/FMM

Direct Method

直接法

- Gaussian Elimination/LU Factorization.
 - compute \mathbf{A}^{-1} directly.

Good

- Robust for wide range of applications.
- Good for both dense and sparse matrices

Bad

- More expensive than iterative methods (memory, CPU)
 - not scalable

What is Iterative Method ?

反復法

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

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

Iterative Method

反復法

- Stationary Method
 - Only \mathbf{x} (solution vector) changes during iterations.
 - SOR, Gauss-Seidel, Jacobi
 - Generally slow, impractical

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

- Non-Stationary Method
 - With restriction/optimization conditions
 - Krylov-Subspace
 - CG: Conjugate Gradient
 - BiCGSTAB: Bi-Conjugate Gradient Stabilized
 - GMRES: Generalized Minimal Residual

Iterative Method (cont.)

Good

- Less expensive than direct methods, especially in memory.
- Suitable for parallel and vector computing.

Bad

- Convergence strongly depends on problems, boundary conditions (condition number etc.)
- Preconditioning is required : Key Technology for Parallel FEM

Non-Stationary/Krylov Subspace Method (1/2)

非定常法・クリロフ部分空間法

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

Compute $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k$ by the following iterative procedures:

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

where $\mathbf{r}_k = \mathbf{b} - \mathbf{Ax}_k$: residual



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

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

Non-Stationary/Krylov Subspace Method (2/2)

非定常法・クリロフ部分空間法

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

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



\mathbf{z}_k is a vector which belongs to k^{th} Krylov Subspace (クリロフ部分空間), approximate solution vector \mathbf{x}_k is derived by the Krylov Subspace:

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

Conjugate Gradient Method

共役勾配法

- Conjugate Gradient: CG
 - Most popular “non-stationary” iterative method
- for Symmetric Positive Definite (SPD) Matrices
 - 対称正定
 - $\{x\}^T[A]\{x\} > 0$ for arbitrary $\{x\}$
 - All of diagonal components, eigenvalues and leading principal minors > 0 (主小行列式・首座行列式)
 - Matrices of Galerkin-based FEM: heat conduction, Poisson, static linear elastic problems

- Algorithm

- “Steepest Descent Method”

- $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \alpha_i \mathbf{p}^{(i)}$

- $\mathbf{x}^{(i)}$: solution, $\mathbf{p}^{(i)}$: search direction, α_i : coefficient

- Solution $\{x\}$ minimizes $\{x-y\}^T[A]\{x-y\}$, where $\{y\}$ is exact solution.

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

Procedures of Conjugate Gradient

```

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

```

- Mat-Vec. Multiplication
- Dot Products
- DAXPY (Double Precision: $a\{X\} + \{Y\}$)

$x^{(i)}$: Vector

α_i : Scalar

Procedures of Conjugate Gradient

```

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

```

- Mat-Vec. Multiplication
- Dot Products
- DAXPY

$x^{(i)}$: Vector

α_i : Scalar

Procedures of Conjugate Gradient

```

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

```

- Mat-Vec. Multiplication
- Dot Products
- DAXPY

$x^{(i)}$: Vector

α_i : Scalar

Procedures of Conjugate Gradient

```

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

```

- Mat-Vec. Multiplication
- Dot Products
- DAXPY
 - Double
 - $\{y\} = a\{x\} + \{y\}$

$x^{(i)}$: Vector

α_i : Scalar

Procedures of Conjugate Gradient

```

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

```

$x^{(i)}$: Vector

α_i : Scalar

Derivation of CG Algorithm (1/5)

Solution x minimizes the following equation if y is the exact solution ($Ay=b$)

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

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

Therefore, the solution x minimizes the following $f(x)$:

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

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

Arbitrary vector h

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

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

Arbitrary vector h

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

Derivation of CG Algorithm (2/5)

CG method minimizes $f(x)$ at each iteration. Assume that approximate solution: $x^{(k)}$, and search direction vector $p^{(k)}$ is defined at k -th iteration.

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

Minimization of $f(x^{(k+1)})$ is done as follows:

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

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

$$r^{(k)} = b - Ax^{(k)} \text{ residual vector}$$

Derivation of CG Algorithm (3/5)

Residual vector at $(k+1)$ -th iteration:

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

$$r^{(k+1)} = b - Ax^{(k+1)}, r^{(k)} = b - Ax^{(k)}$$

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

Search direction vector p is defined by the following recurrence formula:

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

It's lucky if we can get exact solution y at $(k+1)$ -th iteration:

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

Derivation of CG Algorithm (4/5)

BTW, we have the following (convenient) orthogonality relation:

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

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

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

Thus, following relation is obtained:

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

Derivation of CG Algorithm (5/5)

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

$(p^{(k+1)}, Ap^{(k)}) = 0$ $p^{(k)}$ and $p^{(k+1)}$ are “conjugate (共役)” for matrix A

```

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

```

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

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

Properties of CG Algorithm

Following “conjugate (共役)” relationship is obtained for arbitrary (i, j) :

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

Following relationships are also obtained for $p^{(k)}$ and $r^{(k)}$:

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

In N-dimensional space, only N sets of orthogonal and linearly independent residual vector $r^{(k)}$. This means CG method converges after N iterations if number of unknowns is N. Actually, round-off error sometimes affects convergence.

Proof (1/3)

Mathematical Induction

数学的帰納法

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

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

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

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

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

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 $i \neq j$

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

$$\text{if } i = k \quad (r^{(k+1)}, r^{(k)}) \stackrel{(2)}{=} (r^{(k)}, r^{(k)}) - (r^{(k)}, \alpha_k Ap^{(k)})$$

$$\stackrel{(3)}{=} (r^{(k)}, r^{(k)}) - (p^{(k)} - \beta_{k-1} p^{(k-1)}, \alpha_k Ap^{(k)})$$

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

$$\stackrel{(*)}{=} (r^{(k)}, r^{(k)}) - \alpha_k (p^{(k)}, Ap^{(k)}) \stackrel{(1)}{=} (r^{(k)}, r^{(k)}) - (p^{(k)}, r^{(k)})$$

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

$$\stackrel{(3)}{=} (r^{(k)}, r^{(k)}) - (\beta_{k-1} p^{(k-1)} + r^{(k)}, r^{(k)})$$

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

$$= -\beta_{k-1} (p^{(k-1)}, r^{(k)}) \stackrel{(2)}{=} -\beta_{k-1} (p^{(k-1)}, r^{(k-1)} - \alpha_{k-1} Ap^{(k-1)})$$

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

$$= -\beta_{k-1} \left\{ (p^{(k-1)}, r^{(k-1)}) - \alpha_{k-1} (p^{(k-1)}, Ap^{(k-1)}) \right\} \stackrel{(1)}{=} 0$$

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 $i \neq j$

$$\begin{aligned} \text{if } i < k \quad (p^{(k+1)}, Ap^{(i)}) &\stackrel{(3)}{=} (r^{(k+1)} + \beta_k p^{(k)}, Ap^{(i)}) \\ &\stackrel{(*)}{=} (r^{(k+1)}, Ap^{(i)}) \\ &\stackrel{(2)}{=} \frac{1}{\alpha_i} (r^{(k+1)}, r^{(i)} - r^{(i+1)}) = 0 \end{aligned}$$

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

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

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

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

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

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

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

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

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

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

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

$$\alpha_k, \beta_k$$

Usually, we use simpler definitions of α_k, β_k as follows:

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

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

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

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

Procedures of Conjugate Gradient

```

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

```

$x^{(i)}$: Vector

α_i : Scalar

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

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

Preconditioning for Iterative Solvers

- Convergence rate of iterative solvers strongly depends on the spectral properties (eigenvalue distribution) of the coefficient matrix \mathbf{A} .
 - Eigenvalue distribution is small, eigenvalues are close to 1
 - In “ill-conditioned” problems, “condition number” (ratio of max/min eigenvalue if \mathbf{A} is symmetric) is large (条件数) .
- A preconditioner \mathbf{M} (whose properties are similar to those of \mathbf{A}) transforms the linear system into one with more favorable spectral properties (前处理)
 - \mathbf{M} transforms $\mathbf{Ax}=\mathbf{b}$ into $\mathbf{A}'\mathbf{x}=\mathbf{b}'$ where $\mathbf{A}'=\mathbf{M}^{-1}\mathbf{A}$, $\mathbf{b}'=\mathbf{M}^{-1}\mathbf{b}$
 - If $\mathbf{M}\sim\mathbf{A}$, $\mathbf{M}^{-1}\mathbf{A}$ is close to identity matrix.
 - If $\mathbf{M}^{-1}=\mathbf{A}^{-1}$, this is the best preconditioner (Gaussian Elim.)
 - Generally, $\mathbf{A}'\mathbf{x}'=\mathbf{b}'$ where $\mathbf{A}'=\mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1}$, $\mathbf{b}'=\mathbf{M}_L^{-1}\mathbf{b}$, $\mathbf{x}'=\mathbf{M}_R\mathbf{x}$
 - $\mathbf{M}_L/\mathbf{M}_R$: Left/Right Preconditioning (左/右前处理)

Preconditioned CG Solver

```

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

```

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

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

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

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

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

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

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

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

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

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

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

In CG method, preconditioner usually satisfies $[M_2] = [M_1]^T$, such as Incomplete Cholesky/Incomplete Modified Cholesky Factorizations.

In this problem, let us define $[M_1]$ and $[M_2]$ as follows:

$$[M_1] = [X]^T, [M_2] = [X], [M] = [M_1][M_2]$$

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

$$[A'] = [M_1]^{-1}[A][M_2]^{-1} = [[X]^T]^{-1}[A][X]^{-1} = [X]^{-T}[A][X]^{-1}$$

$$x' = [X]x, \quad b' = [X]^{-T}b, \quad r' = [X]^{-T}r$$

$$\begin{aligned} \alpha'_{i-1} &= \frac{\left(r^{(i-1)}, r^{(i-1)} \right)}{\left(p^{(i-1)}, A' p^{(i-1)} \right)} = \frac{\left([X]^{-T} r^{(i-1)}, [X]^{-T} r^{(i-1)} \right)}{\left([X] p^{(i-1)}, [X]^{-T} [A] [X]^{-1} [X] p^{(i-1)} \right)} \\ &= \frac{\left(\left([X]^{-T} r^{(i-1)} \right)^T, [X]^{-T} r^{(i-1)} \right)}{\left(\left([X] p^{(i-1)} \right)^T, [X]^{-T} [A] p^{(i-1)} \right)} = \frac{\left(\left(r^{(i-1)} \right)^T [X]^{-1}, [X^T]^{-1} r^{(i-1)} \right)}{\left(\left(p^{(i-1)} \right)^T [X]^T, [X]^{-T} [A] p^{(i-1)} \right)} \\ &= \frac{\left(r^{(i-1)}, \left[[X^T] [X] \right]^{-1} r^{(i-1)} \right)}{\left(p^{(i-1)}, [A] p^{(i-1)} \right)} = \frac{\left(r^{(i-1)}, [M]^{-1} r^{(i-1)} \right)}{\left(p^{(i-1)}, [A] p^{(i-1)} \right)} = \frac{\left(r^{(i-1)}, z^{(i-1)} \right)}{\left(p^{(i-1)}, [A] p^{(i-1)} \right)} \end{aligned}$$

$$\begin{aligned}
\beta'_{i-1} &= \frac{\left(r^{(i-1)}, r^{(i-1)} \right)}{\left(r^{(i-2)}, r^{(i-2)} \right)} = \frac{\left([\mathbf{X}]^{-T} r^{(i-1)}, [\mathbf{X}]^{-T} r^{(i-1)} \right)}{\left([\mathbf{X}]^{-T} r^{(i-2)}, [\mathbf{X}]^{-T} r^{(i-2)} \right)} \\
&= \frac{\left(\left([\mathbf{X}]^{-T} r^{(i-1)} \right)^T, [\mathbf{X}]^{-T} r^{(i-1)} \right)}{\left(\left([\mathbf{X}]^{-T} r^{(i-2)} \right)^T, [\mathbf{X}]^{-T} r^{(i-2)} \right)} = \frac{\left(\left(r^{(i-1)} \right)^T [\mathbf{X}]^{-1}, [\mathbf{X}^T]^{-1} r^{(i-1)} \right)}{\left(\left(r^{(i-2)} \right)^T [\mathbf{X}]^{-1}, [\mathbf{X}^T]^{-1} r^{(i-2)} \right)} \\
&= \frac{\left(r^{(i-1)}, \left[[\mathbf{X}^T] [\mathbf{X}] \right]^{-1} r^{(i-1)} \right)}{\left(r^{(i-2)}, \left[[\mathbf{X}^T] [\mathbf{X}] \right]^{-1} r^{(i-2)} \right)} = \frac{\left(r^{(i-1)}, [\mathbf{M}]^{-1} r^{(i-1)} \right)}{\left(r^{(i-2)}, [\mathbf{M}]^{-1} r^{(i-2)} \right)} = \frac{\left(r^{(i-1)}, z^{(i-1)} \right)}{\left(r^{(i-2)}, z^{(i-2)} \right)}
\end{aligned}$$

Preconditioned Conjugate Gradient Method (PCG)

```

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

```

Solving the following equation:

$$\{z\} = [M]^{-1} \{r\}$$

“Approximate Inverse Matrix”

$$[M]^{-1} \approx [A]^{-1}, \quad [M] \approx [A]$$

Ultimate Preconditioning:

Inverse Matrix

$$[M]^{-1} = [A]^{-1}, \quad [M] = [A]$$

Diagonal Scaling: Simple but weak

$$[M]^{-1} = [D]^{-1}, \quad [M] = [D]$$

ILU(0), IC(0)

- Widely used Preconditioners for Sparse Matrices
 - Incomplete LU Factorization (不完全LU分解)
 - Incomplete Cholesky Factorization (for Symmetric Matrices) (不完全コレスキー分解)
- Incomplete Direct Method
 - Even if original matrix is sparse, inverse matrix is not necessarily sparse.
 - fill-in
 - ILU(0)/IC(0) without fill-in have same non-zero pattern with the original (sparse) matrices

Diagonal Scaling, Point-Jacobi

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

- **solve** $[M] \mathbf{z}^{(i-1)} = \mathbf{r}^{(i-1)}$ is very easy.
- Provides fast convergence for simple problems.
- 1d.f, 1d.c

- 1D-code for Static Linear-Elastic Problems by Galerkin FEM
- Sparse Linear Solver
 - Conjugate Gradient Method
 - Preconditioning
- **Storage of Sparse Matrices**
- Program

Variables/Arrays in 1d.f, 1d.c related to coefficient matrix

name	type	size	description
N	I	–	# Unknowns
NPLU	I	–	# Non-Zero Off-Diagonal Components
Diag(:)	R	N	Diagonal Components
U(:)	R	N	Unknown Vector
Rhs(:)	R	N	RHS Vector
Index(:)	I	0:N N+1	Off-Diagonal Components (Number of Non-Zero Off-Diagonals at Each ROW)
Item(:)	I	NPLU	Off-Diagonal Components (Corresponding Column ID)
AMat(:)	R	NPLU	Off-Diagonal Components (Value)

Only non-zero components are stored according to “Compressed Row Storage”.

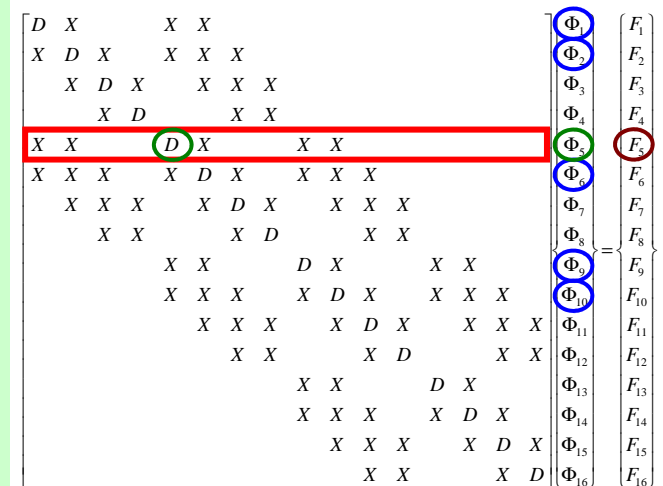
Mat-Vec. Multiplication for Sparse Matrix

Compressed Row Storage (CRS)

Diag (i) Diagonal Components (REAL, i=1~N)
Index (i) Number of Non-Zero Off-Diagonals at Each ROW (INT, i=0~N)
Item (k) Off-Diagonal Components (Corresponding Column ID)
 (INT, k=1, index(N))
AMat (k) Off-Diagonal Components (Value)
 (REAL, k=1, index(N))

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

```
do i= 1, N
  Y(i)= Diag(i)*X(i)
  do k= Index(i-1)+1, Index(i)
    Y(i)= Y(i) + Amat(k)*X(Item(k))
  enddo
enddo
```



CRS or CSR ?

for Compressed Row Storage

- In Japan and USA, “CRS” is very general for abbreviation of “Compressed Row Storage”, but they usually use “CSR” in Europe (especially in France).
- “CRS” in France
 - Compagnie Républicaine de Sécurité
 - Republic Security Company of France
- French scientists may feel uncomfortable when we use “CRS” in technical papers and/or presentations.



Mat-Vec. Multiplication for Sparse Matrix

Compressed Row Storage (CRS)

```
{Q} = [A] {P}

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


Mat-Vec. Multiplication for Dense Matrix

Very Easy, Straightforward

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

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

```

do j= 1, N
  Y(j) = 0. d0
  do i= 1, N
    Y(j) = Y(j) + A(i, j)*X(i)
  enddo
enddo

```

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

Numbering starts from 0 in program

	0	1	2	3	4	5	6	7
0	1.1 ⊙	2.4 ①			3.2 ④			
1	4.3 ⊙	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 ⊙	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

Diagonal Components

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) : C

	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) : C

						# Non-Zero Off-Diag.	Index [i] =
0	1.1 ⊙	2.4 ①	3.2 ④			2	0
1	3.6 ①	4.3 ⊙	2.5 ③	3.7 ⑤	9.1 ⑦	4	2
2	5.7 ②	1.5 ④	3.1 ⑥			2	6
3	9.8 ③	4.1 ①	2.5 ④	2.7 ⑤		3	8
4	11.5 ④	3.1 ⊙	9.5 ①	10.4 ②	4.3 ⑥	4	11
5	12.4 ⑤	6.5 ②	9.5 ⑥			2	15
6	23.1 ⑥	6.4 ①	2.5 ②	1.4 ⑤	13.1 ⑦	4	17
7	51.3 ⑦	9.5 ①	1.3 ②	9.6 ③	3.1 ⑤	4	21

NPLU= 25
(=Index[N])

(Index[i])th ~ (Index[i+1])th:

Non-Zero Off-Diag. Components corresponding to *i*-th row.

Compressed Row Storage (CRS) : C

		# Non-Zero Off-Diag.	Index [i] =										
0	<table border="1"> <tr><td>1.1</td><td>2.4</td><td>3.2</td></tr> <tr><td>⊙</td><td>①,0</td><td>④,1</td></tr> </table>	1.1	2.4	3.2	⊙	①,0	④,1	2	0				
1.1	2.4	3.2											
⊙	①,0	④,1											
1	<table border="1"> <tr><td>3.6</td><td>4.3</td><td>2.5</td><td>3.7</td><td>9.1</td></tr> <tr><td>①</td><td>⊙,2</td><td>③,3</td><td>⑤,4</td><td>⑦,5</td></tr> </table>	3.6	4.3	2.5	3.7	9.1	①	⊙,2	③,3	⑤,4	⑦,5	4	2
3.6	4.3	2.5	3.7	9.1									
①	⊙,2	③,3	⑤,4	⑦,5									
2	<table border="1"> <tr><td>5.7</td><td>1.5</td><td>3.1</td><td></td></tr> <tr><td>②</td><td>④,6</td><td>⑥,7</td><td></td></tr> </table>	5.7	1.5	3.1		②	④,6	⑥,7		2	6		
5.7	1.5	3.1											
②	④,6	⑥,7											
3	<table border="1"> <tr><td>9.8</td><td>4.1</td><td>2.5</td><td>2.7</td></tr> <tr><td>③</td><td>①,8</td><td>④,9</td><td>⑤,10</td></tr> </table>	9.8	4.1	2.5	2.7	③	①,8	④,9	⑤,10	3	8		
9.8	4.1	2.5	2.7										
③	①,8	④,9	⑤,10										
4	<table border="1"> <tr><td>11.5</td><td>3.1</td><td>9.5</td><td>10.4</td><td>4.3</td></tr> <tr><td>④</td><td>⊙,11</td><td>①,12</td><td>②,13</td><td>⑥,14</td></tr> </table>	11.5	3.1	9.5	10.4	4.3	④	⊙,11	①,12	②,13	⑥,14	4	11
11.5	3.1	9.5	10.4	4.3									
④	⊙,11	①,12	②,13	⑥,14									
5	<table border="1"> <tr><td>12.4</td><td>6.5</td><td>9.5</td><td></td></tr> <tr><td>⑤</td><td>②,15</td><td>⑥,16</td><td></td></tr> </table>	12.4	6.5	9.5		⑤	②,15	⑥,16		2	15		
12.4	6.5	9.5											
⑤	②,15	⑥,16											
6	<table border="1"> <tr><td>23.1</td><td>6.4</td><td>2.5</td><td>1.4</td><td>13.1</td></tr> <tr><td>⑥</td><td>①,17</td><td>②,18</td><td>⑤,19</td><td>⑦,20</td></tr> </table>	23.1	6.4	2.5	1.4	13.1	⑥	①,17	②,18	⑤,19	⑦,20	4	17
23.1	6.4	2.5	1.4	13.1									
⑥	①,17	②,18	⑤,19	⑦,20									
7	<table border="1"> <tr><td>51.3</td><td>9.5</td><td>1.3</td><td>9.6</td><td>3.1</td></tr> <tr><td>⑦</td><td>①,21</td><td>②,22</td><td>③,23</td><td>⑤,24</td></tr> </table>	51.3	9.5	1.3	9.6	3.1	⑦	①,21	②,22	③,23	⑤,24	4	21
51.3	9.5	1.3	9.6	3.1									
⑦	①,21	②,22	③,23	⑤,24									
			25										

NPLU= 25
(=Index [N])

(Index [i])th ~ (Index [i+1])th :
Non-Zero Off-Diag. Components corresponding to *i*-th row.

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

Item[6]= 4, AMat[6]= 1.5
Item[18]= 2, AMat[18]= 2.5

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] Diagonal Components (REAL, i=0~N-1)
Index [i] Number of Non-Zero Off-Diagonals at Each ROW (INT, i=0~N)
Item [k] Off-Diagonal Components (Corresponding Column ID) (INT, k=0, index[N])
Amat [k] Off-Diagonal Components (Value) (REAL, k=0, 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]];
    }
}
```


- 1D-code for Static Linear-Elastic Problems by Galerkin FEM
- Sparse Linear Solver
 - Conjugate Gradient Method
 - Preconditioning
- Storage of Sparse Matrices
- Program

Finite Element Procedures

- Initialization
 - Control Data
 - Node, Connectivity of Elements (N: Node#, NE: Elem#)
 - Initialization of Arrays (Global/Element Matrices)
 - Element-Global Matrix Mapping (Index, Item)
- Generation of Matrix
 - Element-by-Element Operations (do icel= 1, NE)
 - Element matrices
 - Accumulation to global matrix
 - Boundary Conditions
- Linear Solver
 - Conjugate Gradient Method

Program: 1d.c (1/6)

variables and arrays

```
/*  
// 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;
```

Variable/Arrays (1/2)

Name	Type	Size	I/O	Definition
NE	I		I	# Element
N	I		O	# Node
NPLU	I		O	# Non-Zero Off-Diag. Components
IterMax	I		I	MAX Iteration Number for CG
errno	I		O	ERROR flag
R, Z, Q, P, DD	I		O	Name of Vectors in CG
dX	R		I	Length of Each Element
Resid	R		O	Residual for CG
Eps	R		I	Convergence Criteria for CG
Area	R		I	Sectional Area of Element
QV	R		I	Heat Generation Rate/Volume/Time \dot{Q}
COND	R		I	Thermal Conductivity

Variable/Arrays (2/2)

Name	Type	Size	I/O	Definition
X	R	N	○	Location of Each Node
PHI	R	N	○	Temperature of Each Node
Rhs	R	N	○	RHS Vector
Diag	R	N	○	Diagonal Components
W	R	[4] [N]	○	Work Array for CG
Amat	R	NPLU	○	Off-Diagonal Components (Value)
Index	I	N+1	○	Number of Non-Zero Off-Diagonals at Each ROW
Item	I	NPLU	○	Off-Diagonal Components (Corresponding Column ID)
Icelnod	I	2*NE	○	Node ID for Each Element
Kmat	R	[2] [2]	○	Element Matrix [k]
Emat	R	[2] [2]	○	Element Matrix

Program: 1d.c (2/6)

Initialization, Allocation of Arrays

```

/*
// +-----+
// | 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));

```

Control Data input.dat

4	NE (Number of Elements)
1.0 1.0 1.0 1.0	Δx (Length of Each Elem.: L), Q , A , λ
100	Number of MAX. Iterations for CG Solver
1.e-8	Convergence Criteria for CG Solver



NE: # Element
N : # Node (NE+1)

Program: 1d.c (2/6)

Initialization, Allocation of Arrays

```

/*
// +-----+
// | 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));

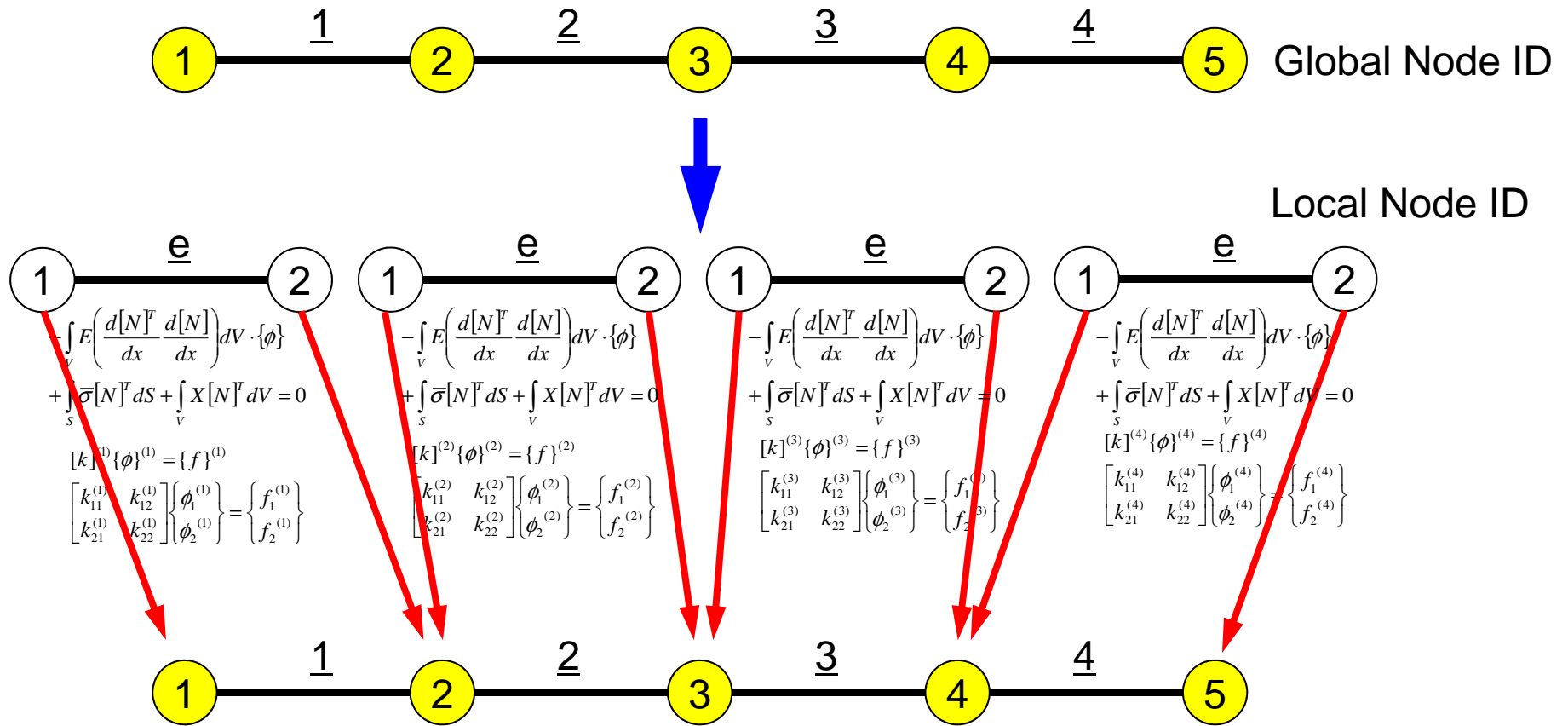
Index= calloc(N+1, sizeof(int));
Item = calloc(2*N-2, sizeof(int));

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

```

AMat: Non-Zero Off-Diag. Comp.
Item: Corresponding Column ID

Element/Global Operations

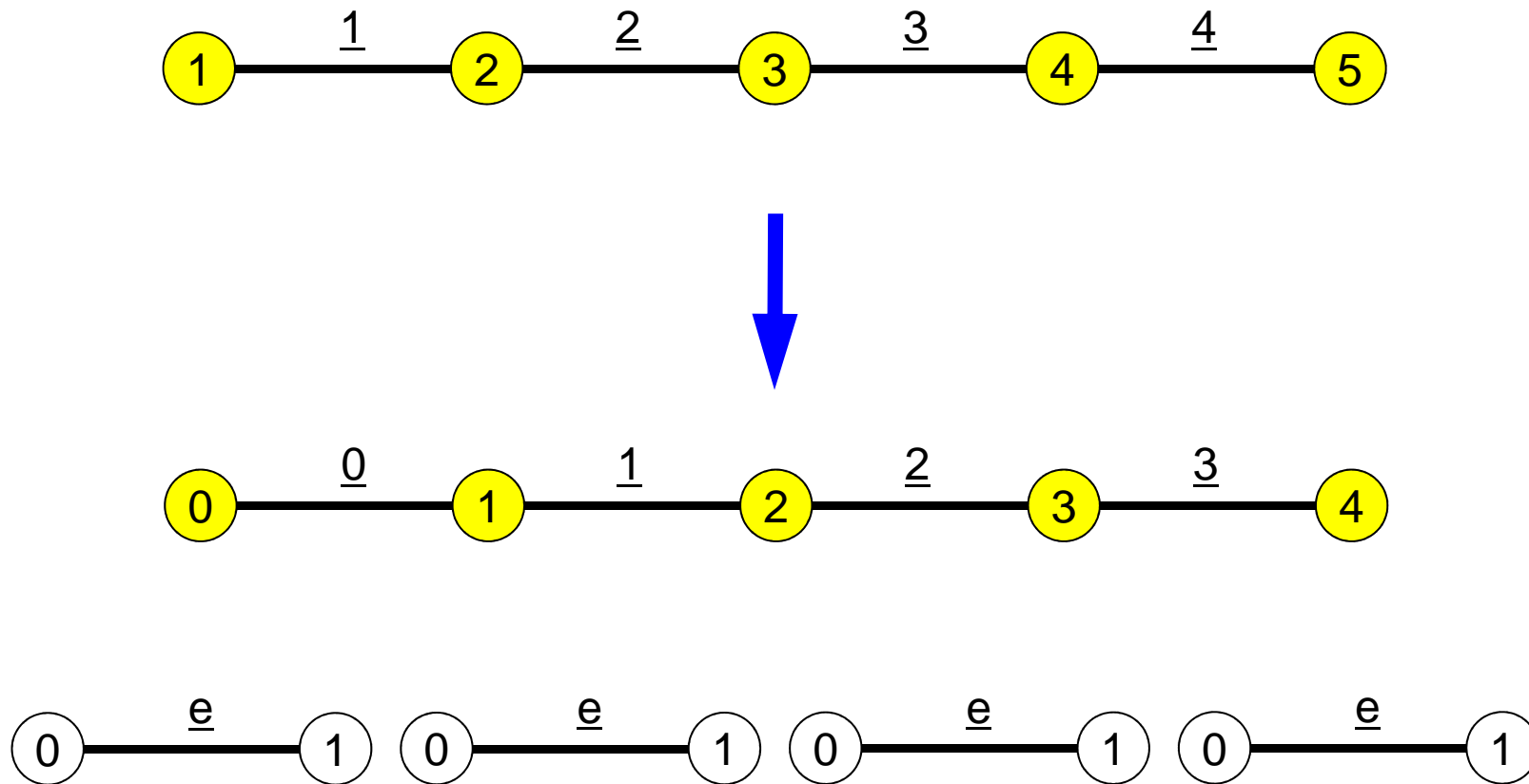


$$[K]\{\Phi\} = \{F\}$$

$$\begin{bmatrix} D_1 & AU_{11} & & & & & \\ AL_{21} & D_2 & AU_{21} & & & & \\ & & AL_{31} & D_3 & AU_{31} & & \\ & & & & AL_{41} & D_4 & AU_{41} \\ & & & & & & AL_{51} & D_5 \end{bmatrix} \begin{Bmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \\ \Phi_5 \end{Bmatrix} = \begin{Bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{Bmatrix}$$

Number of non-zero off-diag. components is 2 for each node. This number is 1 at boundary nodes).

Attention: In C program, node and element ID's start from 0.



Program: 1d.c (2/6)

Initialization, Allocation of Arrays

```

/*
// +-----+
// | 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));

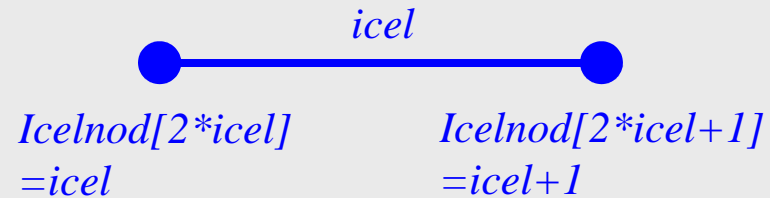
AMat = calloc(2*N-2, sizeof(double));

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: Non-Zero Off-Diag. Comp.
Item: Corresponding Column ID

Number of non-zero off-diag. components is 2 for each node. This number is 1 at boundary nodes).

Total Number of Non-Zero Off-Diag. Components:

$$2 * (N-2) + 1 + 1 = 2 * N - 2$$

Program: 1d.c (3/6)

Initialization, Allocation of Arrays (cont.)

```

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;

```

x: X-coordinate
component of each node

Program: 1d.c (3/6)

Initialization, Allocation of Arrays (cont.)

```

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;

```

```

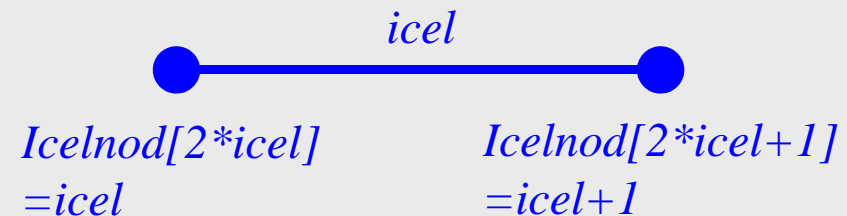
for(icel=0; icel<NE; icel++) {
  Icelnod[2*icel] = icel;
  Icelnod[2*icel+1] = icel+1;
}

```

```

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

```



Program: 1d.c (3/6)

Initialization, Allocation of Arrays (cont.)

```

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]

Program: 1d.c (4/6)

Global Matrix: Column ID for Non-Zero Off-Diag's

```

/*
// +-----+
// | 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;
    }
}

```

Number of non-zero off-diag. components is 2 for each node. This number is 1 at boundary nodes).

Total Number of Non-Zero Off-Diag. Components:
 $2*(N-2)+1+1 = 2*N-2 = NPLU = Index[N]$

		# Non-Zero Off-Diag.	Index[0] = 0										
0	<table border="1"> <tr> <td>1.1</td> <td>2.4</td> <td>3.2</td> </tr> <tr> <td>⊙</td> <td>①</td> <td>④</td> </tr> </table>	1.1	2.4	3.2	⊙	①	④	2	Index[1] = 2				
1.1	2.4	3.2											
⊙	①	④											
1	<table border="1"> <tr> <td>3.6</td> <td>4.3</td> <td>2.5</td> <td>3.7</td> <td>9.1</td> </tr> <tr> <td>①</td> <td>⊙</td> <td>③</td> <td>⑤</td> <td>⑦</td> </tr> </table>	3.6	4.3	2.5	3.7	9.1	①	⊙	③	⑤	⑦	4	Index[2] = 6
3.6	4.3	2.5	3.7	9.1									
①	⊙	③	⑤	⑦									
2	<table border="1"> <tr> <td>5.7</td> <td>1.5</td> <td>3.1</td> </tr> <tr> <td>②</td> <td>④</td> <td>⑥</td> </tr> </table>	5.7	1.5	3.1	②	④	⑥	2	Index[3] = 8				
5.7	1.5	3.1											
②	④	⑥											
3	<table border="1"> <tr> <td>9.8</td> <td>4.1</td> <td>2.5</td> <td>2.7</td> </tr> <tr> <td>③</td> <td>①</td> <td>④</td> <td>⑤</td> </tr> </table>	9.8	4.1	2.5	2.7	③	①	④	⑤	3	Index[4] = 11		
9.8	4.1	2.5	2.7										
③	①	④	⑤										
4	<table border="1"> <tr> <td>11.5</td> <td>3.1</td> <td>9.5</td> <td>10.4</td> <td>4.3</td> </tr> <tr> <td>④</td> <td>⊙</td> <td>①</td> <td>②</td> <td>⑥</td> </tr> </table>	11.5	3.1	9.5	10.4	4.3	④	⊙	①	②	⑥	4	Index[5] = 15
11.5	3.1	9.5	10.4	4.3									
④	⊙	①	②	⑥									
5	<table border="1"> <tr> <td>12.4</td> <td>6.5</td> <td>9.5</td> </tr> <tr> <td>⑤</td> <td>②</td> <td>⑥</td> </tr> </table>	12.4	6.5	9.5	⑤	②	⑥	2	Index[6] = 17				
12.4	6.5	9.5											
⑤	②	⑥											
6	<table border="1"> <tr> <td>23.1</td> <td>6.4</td> <td>2.5</td> <td>1.4</td> <td>13.1</td> </tr> <tr> <td>⑥</td> <td>①</td> <td>②</td> <td>⑤</td> <td>⑦</td> </tr> </table>	23.1	6.4	2.5	1.4	13.1	⑥	①	②	⑤	⑦	4	Index[7] = 21
23.1	6.4	2.5	1.4	13.1									
⑥	①	②	⑤	⑦									
7	<table border="1"> <tr> <td>51.3</td> <td>9.5</td> <td>1.3</td> <td>9.6</td> <td>3.1</td> </tr> <tr> <td>⑦</td> <td>①</td> <td>②</td> <td>③</td> <td>⑤</td> </tr> </table>	51.3	9.5	1.3	9.6	3.1	⑦	①	②	③	⑤	4	Index[8] = 25
51.3	9.5	1.3	9.6	3.1									
⑦	①	②	③	⑤									

$(Index[i])^{th} \sim (Index[i+1])^{th}$:

Non-Zero Off-Diag. Components corresponding to i -th row.

Program: 1d.c (4/6)

Global Matrix: Column ID for Non-Zero Off-Diag's

```

/*
// +-----+
// | 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;
    }
}
    
```



						# Non-Zero Off-Diag.		
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	

(Index[i])th ~ (Index[i+1])th:
Non-Zero Off-Diag. Components corresponding to i-th row.

Program: 1d.c (5/6)

Element Matrix ~ Global Matrix

```

/*
//  +-----+
//  | 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;
}

```



Program: 1d.c (5/6)

Element Matrix ~ Global Matrix

```

/*
// |-----|
// | 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;
}

```



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

Program: 1d.c (5/6)

Element Matrix ~ Global Matrix

```

/*
// |-----|
// | 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;
}

```



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

Program: 1d.c (5/6)

Element Matrix ~ Global Matrix

```

/*
// |-----|
// | 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];

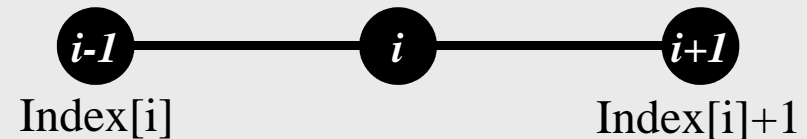
  AMat[k1]= AMat[k1] + Emat[0][1];
  AMat[k2]= AMat[k2] + Emat[1][0];

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

```



Non-zero Off-Diag. at i -th row:
Index[i], Index[i]+1



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

k2

General Elements: k1

“in2” as a off-diag. component of “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;
}

```



Non-zero Off-Diag. at i -th row:
Index[i], Index[i]+1



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

General Elements: k2

“in1” as a off-diag. component of “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;
}

```



Non-zero Off-Diag. at i -th row:
Index[i], Index[i]+1



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

k2

0-th Element: k1

“in2” as a off-diag. component of “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];

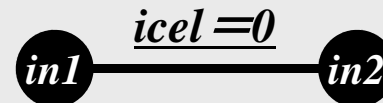
  Diag[in1]= Diag[in1] + Emat[0][0];
  Diag[in2]= Diag[in2] + Emat[1][1];

  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;
}

```



Non-zero Off-Diag. at i -th row: **Index [i]**



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

Program: 1d.c (5/6)

RHS: Heat Generation Term

```

/*
// |-----|
// | 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;
}

```



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

Program: 1d.c (6/6)

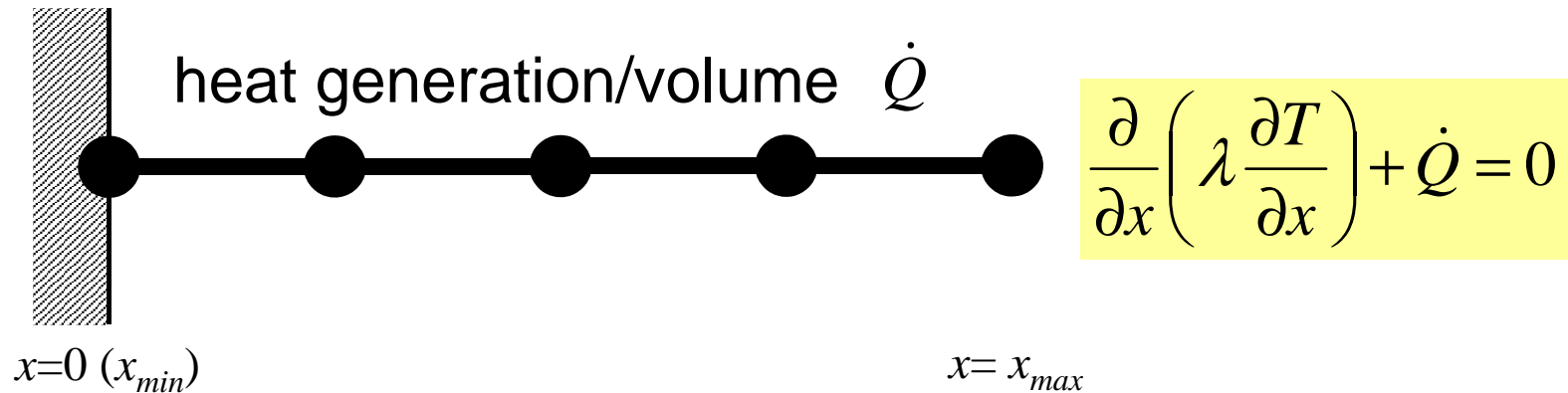
Dirichlet B.C. @ X=0

```
/*
// +-----+
// | 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;
    }}
}
```

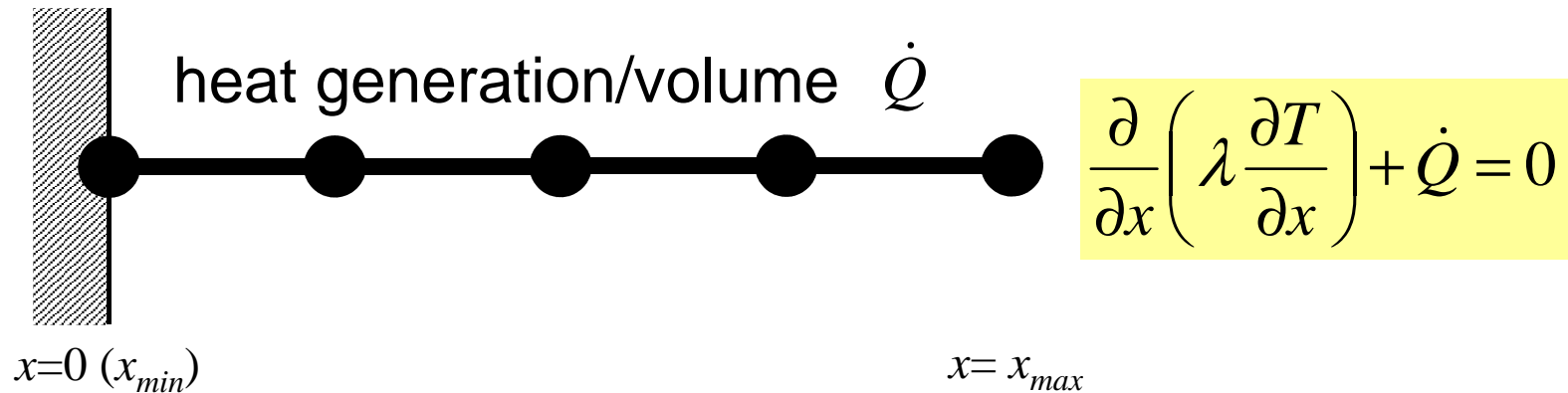

1D Steady State Heat Conduction



- **Uniform: Sectional Area: A , Thermal Conductivity: λ**
- Heat Generation Rate/Volume/Time [$\text{QL}^{-3}\text{T}^{-1}$] \dot{Q}
- Boundary Conditions
 - $x=0$: $T=0$ (Fixed Temperature)
 - $x=x_{max}$: $\frac{\partial T}{\partial x} = 0$ (Insulated)

(Linear) Equation at $x=0$

$$T_I = 0 \text{ (or } T_0 = 0)$$



- **Uniform: Sectional Area: A , Thermal Conductivity: λ**
- Heat Generation Rate/Volume/Time [$QL^{-3}T^{-1}$] \dot{Q}
- Boundary Conditions
 - $x=0$: $T=0$ (Fixed Temperature)
 - $x=x_{max}$: $\frac{\partial T}{\partial x} = 0$ (Insulated)

Program: 1d.c (6/6)

Dirichlet B.C. @ 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;
  }}

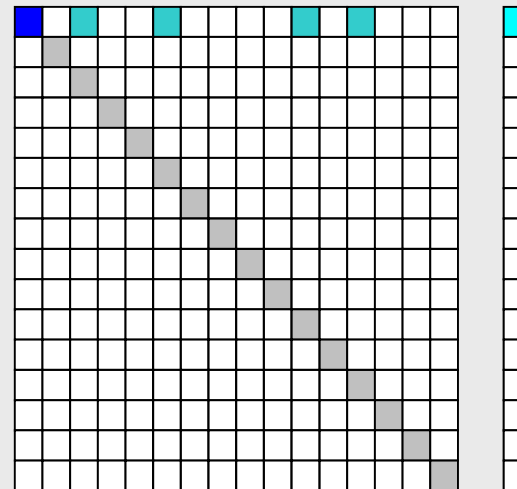
```

$$T_1=0$$

Diagonal Component=1

RHS=0

Off-Diagonal Components= 0.



Program: 1d.c (6/6)

Dirichlet B.C. @ 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;
    }}

```

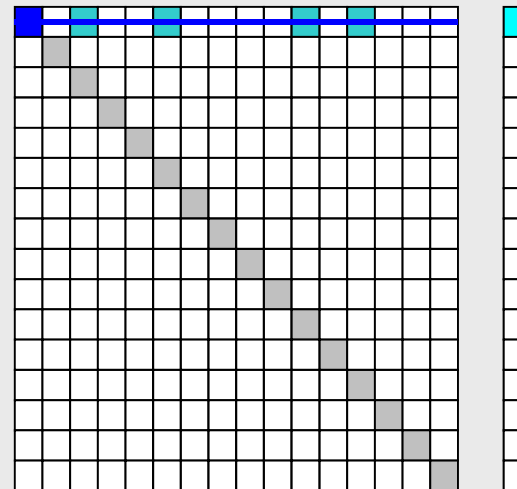
$$T_1=0$$

Diagonal Component=1

RHS=0

Off-Diagonal Components= 0.

Erase !



Program: 1d.c (6/6)

Dirichlet B.C. @ 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;
  }}

```

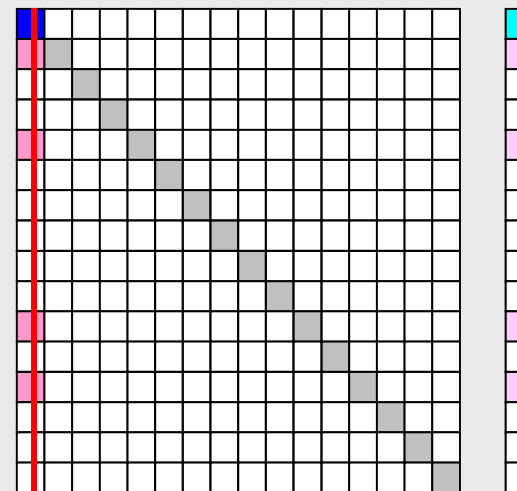
$$T_1=0$$

Diagonal Component=1

RHS=0

Off-Diagonal Components= 0.

Elimination and Erase



Column components of boundary nodes (Dirichlet B.C.) are moved to RHS and eliminated for keeping symmetrical feature of the matrix (in this case just erase off-diagonal components)

if $T_1 \neq 0$

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

Column components of boundary nodes (Dirichlet B.C.) are moved to RHS and eliminated for keeping symmetrical feature of the matrix.

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

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

```
  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;
      }
    }
  }
```

if $T_1 \neq 0$

```

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

```

```

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

```

```

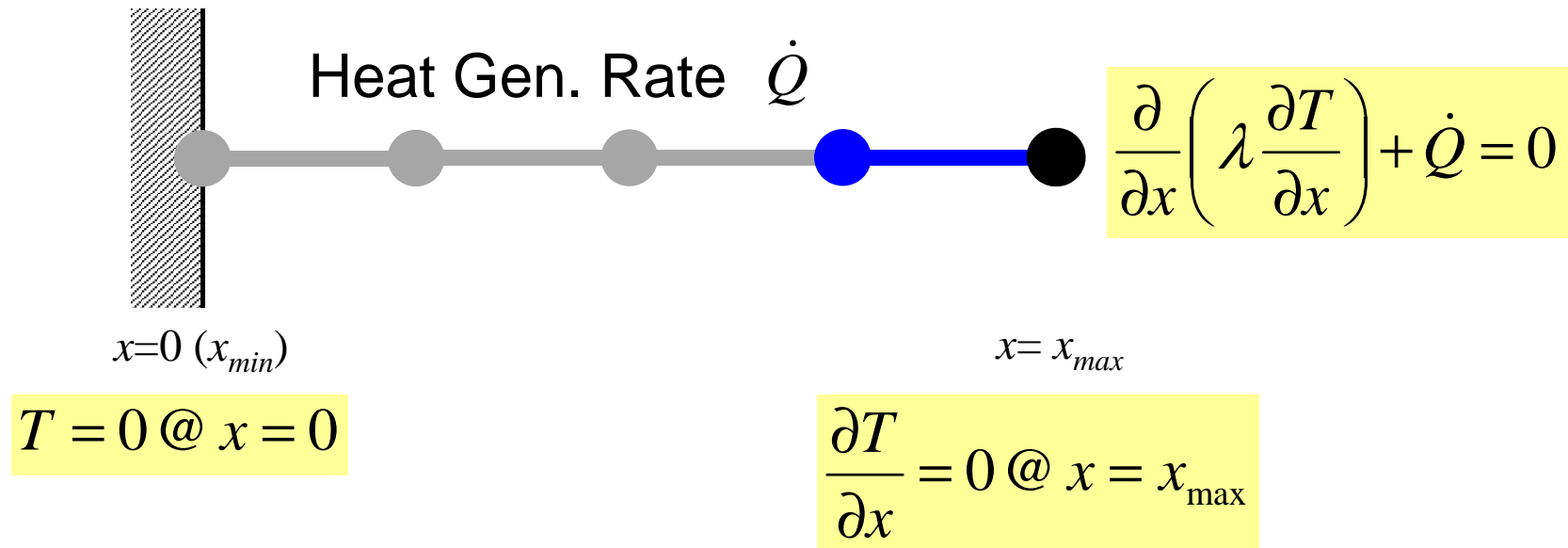
  for (j=1; i<N; i++) {
    for (k=Index[j]; k<Index[j+1]; k++) {
      if (Item[k]==0) {
        Rhs [j]= Rhs[j] - AMat[k]*PHImin;
        AMat[k]= 0.0;
      }
    }
  }

```

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

Column components of boundary nodes (Dirichlet B.C.) are moved to RHS and eliminated for keeping symmetrical feature of the matrix.

Secondary B.C. (Insulated)



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



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

According to insulated B.C., $\bar{q} = 0$ is satisfied. No contribution by this term. Insulated B.C. is automatically satisfied without explicit operations
 -> Natural B.C.

Preconditioned CG Solver

```

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

```

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

Diagonal Scaling, Point-Jacobi

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

- **solve $[M]z^{(i-1)} = r^{(i-1)}$** is very easy.
- Provides fast convergence for simple problems.
- 1d.f, 1d.c

CG Solver (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 Solver (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];
}

```

Reciprocal numbers (倒数) of diagonal components are stored in $W[DD][i]$. Computational cost for division is usually expensive.

```

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}

```

CG Solver (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|²
for convergence criteria
of CG solvers

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 Solver (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 Solver (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 Solver (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  $i = 1, 2, \dots$ 
    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 Solver (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;  $\rho_{i-2}$ 
}

```

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

Control Data input.dat

```

4          NE (Number of Elements)
1.0 1.0 1.0 1.0  $\Delta x$  (Length of Each Elem.:  $L$ ),  $\rho$ ,  $A$ ,  $\lambda$ 
100       Number of MAX. Iterations for CG Solver
1.e-8     Convergence Criteria for CG Solver

```

```

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

```

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

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

Finite Element Procedures

- Initialization
 - Control Data
 - Node, Connectivity of Elements (N: Node#, NE: Elem#)
 - Initialization of Arrays (Global/Element Matrices)
 - Element-Global Matrix Mapping (Index, Item)
- Generation of Matrix
 - Element-by-Element Operations (do icel= 1, NE)
 - Element matrices
 - Accumulation to global matrix
 - Boundary Conditions
- Linear Solver
 - Conjugate Gradient Method

Remedies for Higher Accuracy

- Finer Meshes

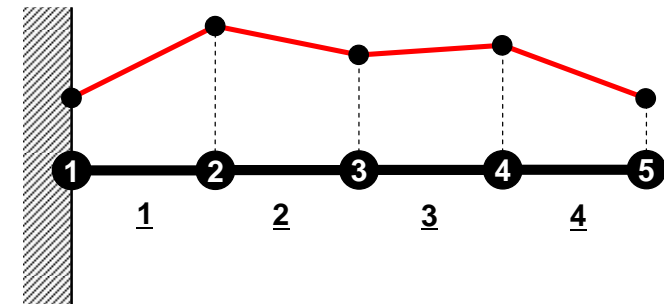
NE=8, dx=12.5

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

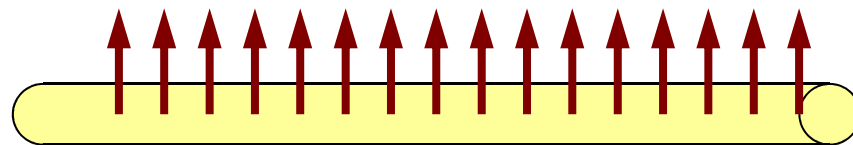
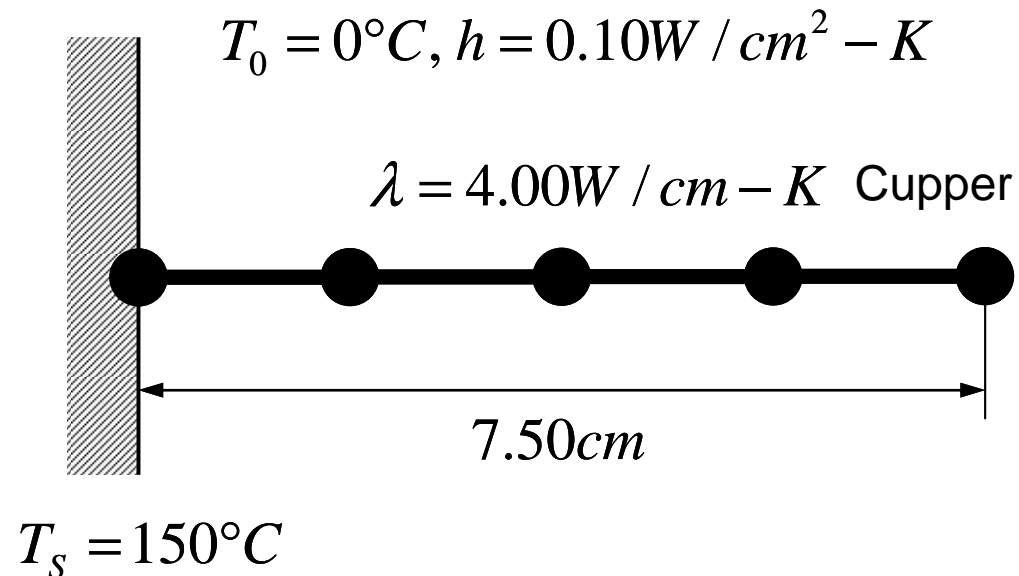
Remedies for Higher Accuracy

- Finer Meshes
- Higher Order Shape/Interpolation Function (高次補間関数・形状関数)
 - Higher-Order Element (高次要素)
 - Linear-Element, 1st-Order Element: Lower Order (低次要素)
- Formulation which assures continuity of n-th order derivatives
 - C^n Continuity (C^n 連続性)

Remedies for Higher Accuracy

- Finer Meshes
- Higher Order Shape/Interpolation Function (高次補間関数・形状関数)
 - Higher-Order Element (高次要素)
 - Linear-Element, 1st-Order Element: Lower Order (低次要素)
- Formulation which assures continuity of n-th order derivatives
 - C^n Continuity (C^n 連続性)
- Linear Elements
 - Piecewise Linear
 - C^0 Continuity
 - Only dependent variables are continuous at element boundary

Example: 1D Heat Transfer (1/2)



Convective Heat Transfer on
Cylindrical Surface

- Temp. Thermal Fins
- Circular Sectional Area, $r=1\text{cm}$
- Boundary Condition
 - $x=0$: Fixed Temperature
 - $x=7.5$: Insulated
- Convective Heat Transfer on Cylindrical Surface
 - $q = h(T - T_0)$
 - q : Heat Flux
 - Heat Flow/Unit Surface Area/sec.

Example: 1D Heat Transfer (2/2)

RESULTS (linear interpolation)

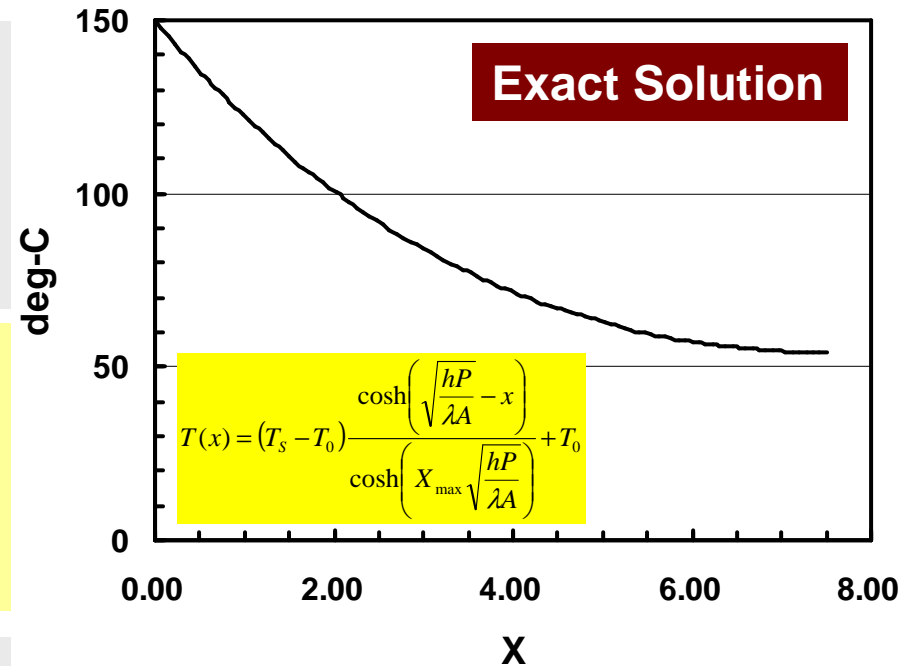
ID	X	FEM.	ANALYTICAL	ERR (%)
1	0.00000	150.00000	150.00000	0.00000
2	1.87500	102.62226	103.00165	0.25292
3	3.75000	73.82803	74.37583	0.36520
4	5.62500	58.40306	59.01653	0.40898
5	7.50000	53.55410	54.18409	0.41999

RESULTS (quadratic interpolation)

ID	X	FEM.	ANALYTICAL	ERR (%)
1	0.00000	150.00000	150.00000	0.00000
2	1.87500	102.98743	103.00165	0.00948
3	3.75000	74.40203	74.37583	0.01747
4	5.62500	59.02737	59.01653	0.00722
5	7.50000	54.21426	54.18409	0.02011

RESULTS (linear interpolation)

ID	X	FEM.	ANALYTICAL	ERR (%)
1	0.00000	150.00000	150.00000	0.00000
2	0.93750	123.71561	123.77127	0.03711
3	1.87500	102.90805	103.00165	0.06240
4	2.81250	86.65618	86.77507	0.07926
5	3.75000	74.24055	74.37583	0.09019
6	4.68750	65.11151	65.25705	0.09703
7	5.62500	58.86492	59.01653	0.10107
8	6.56250	55.22426	55.37903	0.10317
9	7.50000	54.02836	54.18409	0.10382



Quadratic interpolation provides more accurate solution, especially if X is close to 7.50cm.

$$\rho c \frac{\partial T}{\partial t} + \rho c u_j \frac{\partial T}{\partial x_j} - \frac{\partial}{\partial x_j} \left(\lambda \frac{\partial T}{\partial x_j} \right) = \sigma \varepsilon (T_\infty^4 - T^4)$$

Convection
対流

Radiation
放射・輻射

先日、海南島に行ったときに台湾の友人と話していて、某アルゴリズムを輻射熱伝導がらみのアプリで試してみるか、という話になっていたので・・・