

1D-FEM in Fortran: Steady State Heat Conduction

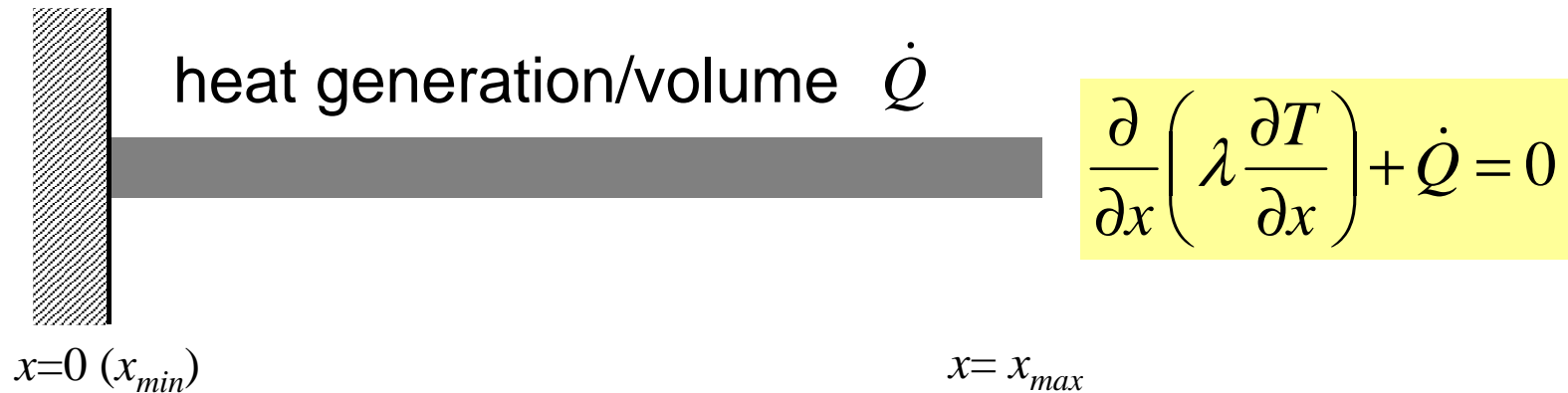
Kengo Nakajima
Information Technology Center
The University of Tokyo

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

Keywords

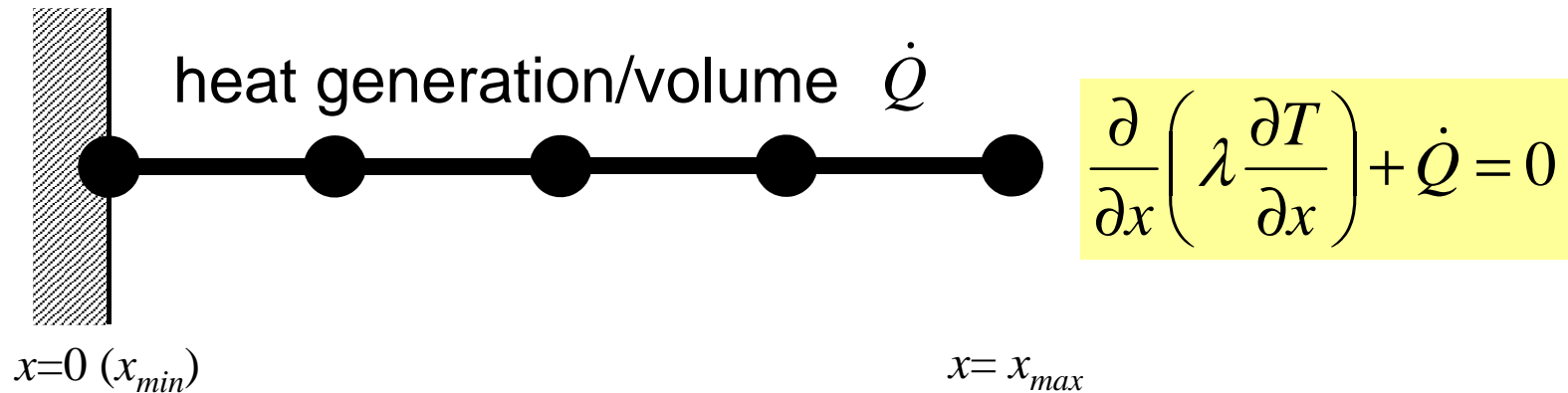
- 1D Steady State Heat Conduction Problems
- Galerkin Method
- Linear Element
- Preconditioned Conjugate Gradient Method

1D Steady State Heat Conduction



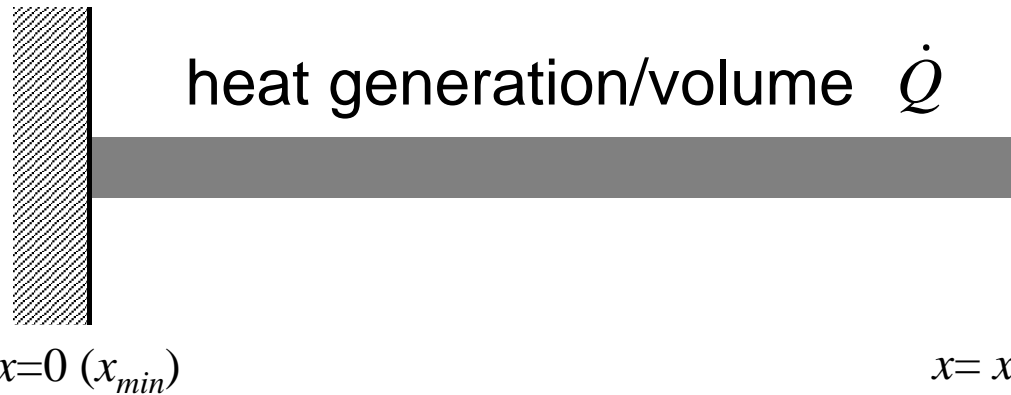
- **Uniform: Sectional Area: A , Thermal Conductivity: λ**
- Heat Generation Rate/Volume/Time [$QL^{-3}T^{-1}$] \dot{Q}
- Boundary Conditions
 - $x=0$: $T=0$ (Fixed Temperature)
 - $x=x_{max}$: $\frac{\partial T}{\partial x} = 0$ (Insulated)

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



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

$$T = 0 @ x = 0$$

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

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

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

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

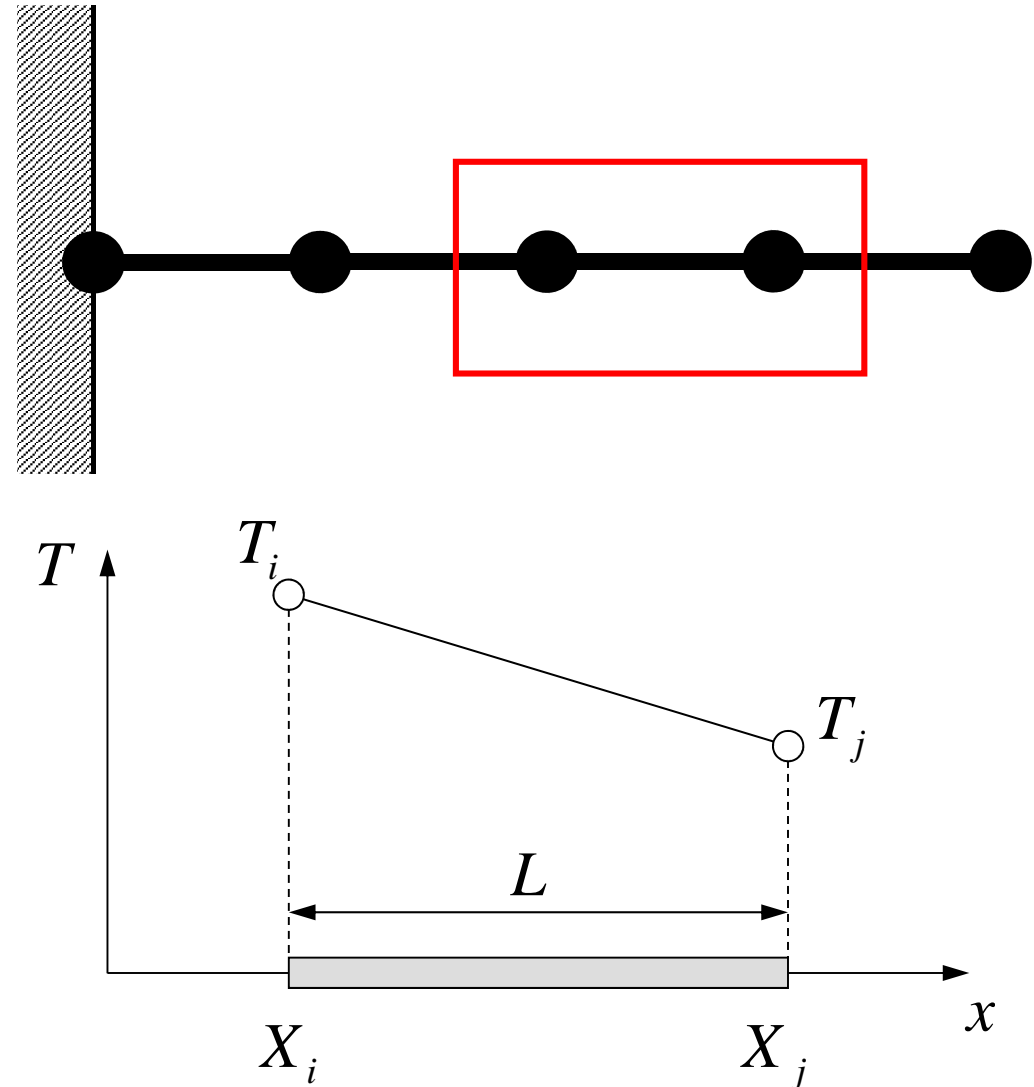
$$\therefore T = -\frac{1}{2\lambda}\dot{Q}x^2 + \frac{\dot{Q}x_{max}}{\lambda}x$$

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

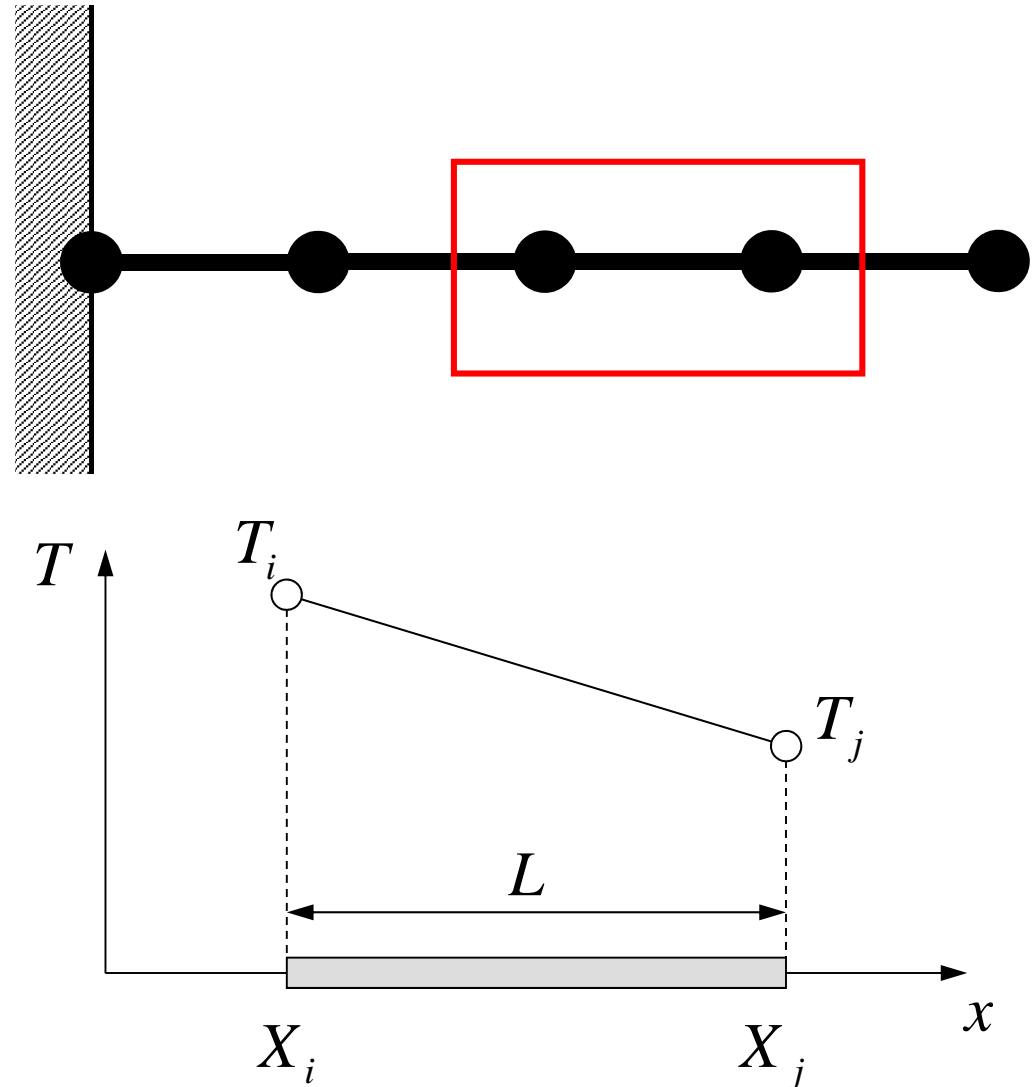
- 1D Linear Element

- Length = L

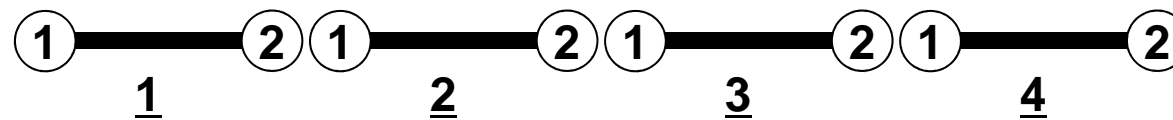
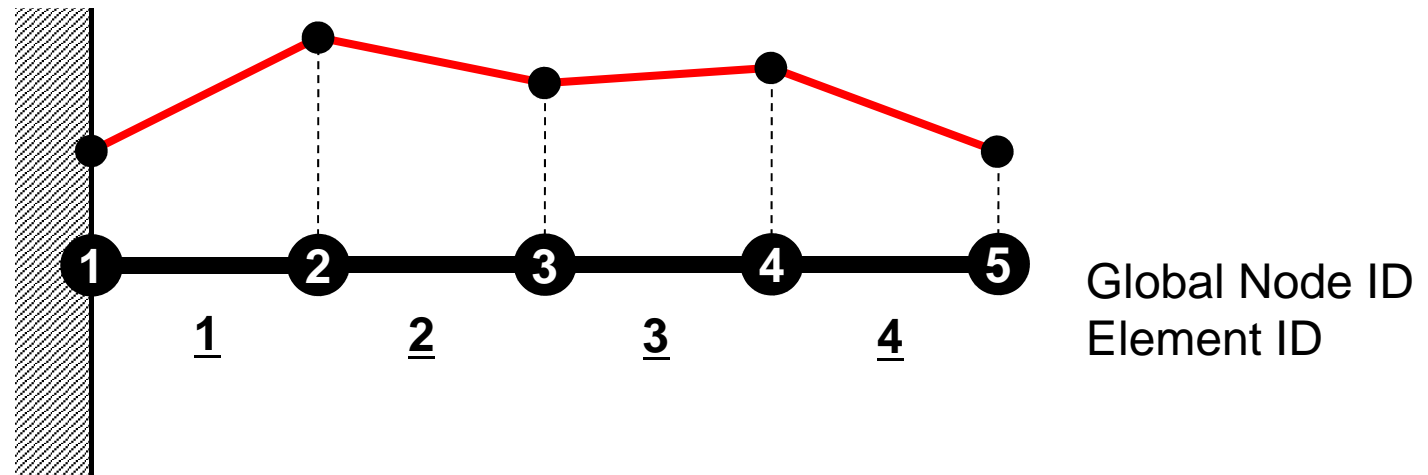
- Node (Vertex)
- Element

- 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:

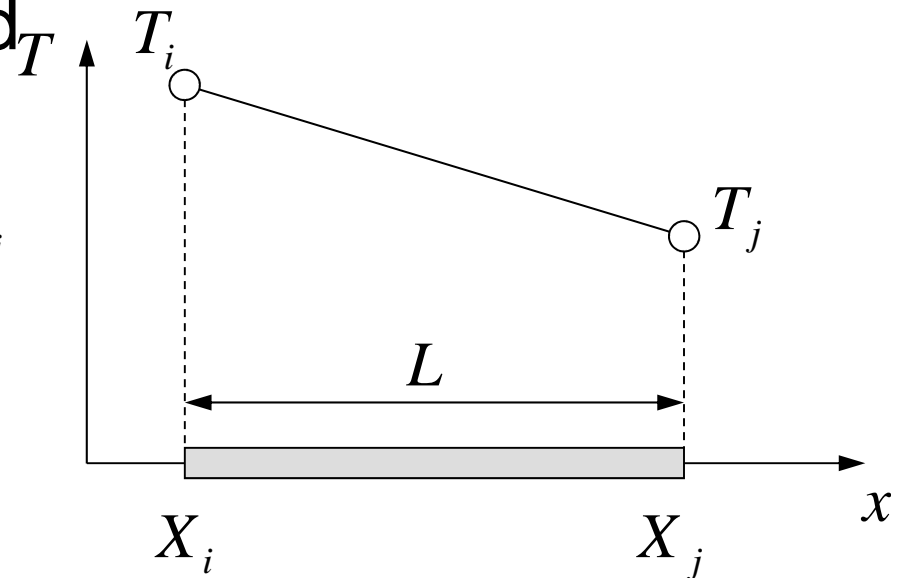
$$\alpha_1 = \frac{T_i X_j - T_j X_i}{L}, \quad \alpha_2 = \frac{T_j - T_i}{L}$$

- 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)

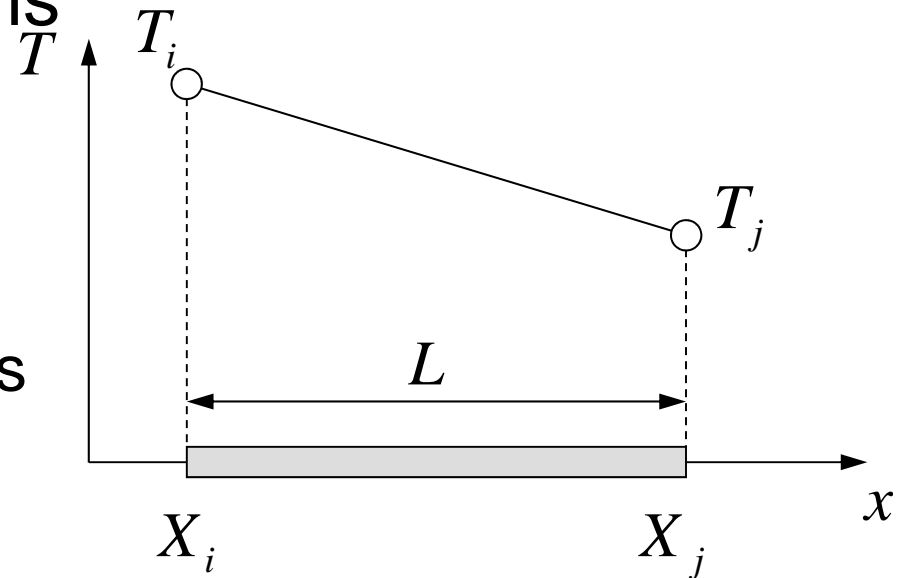


1D Linear Elem.: Shape Function (3/4)

- 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

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

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

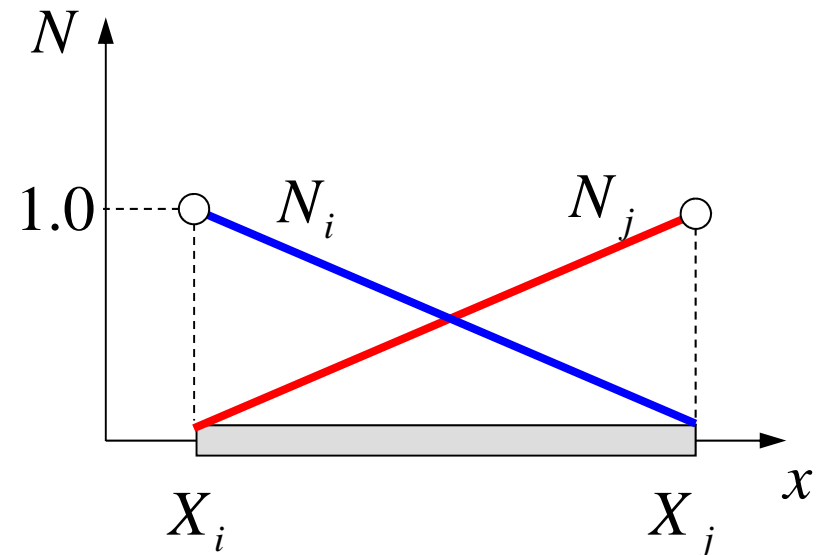
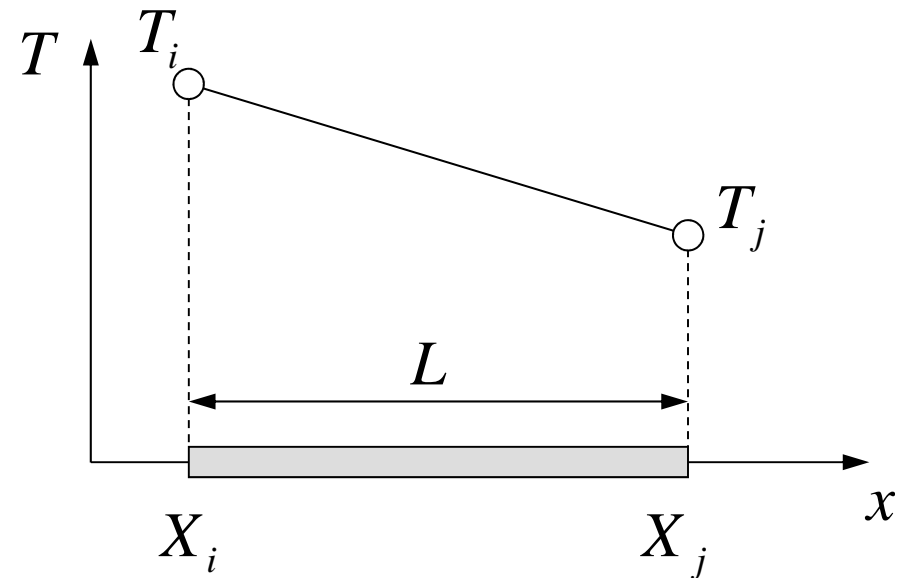
Ψ_i Trial/Test Function (known function of position, defined in domain and at boundary. “Basis” in linear algebra.)

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):

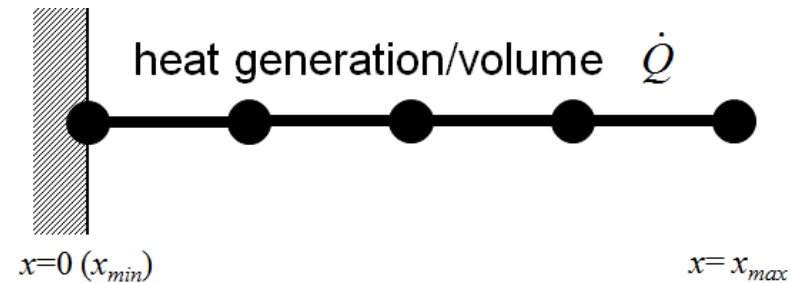
$$\lambda \left(\frac{d^2 T}{dx^2} \right) + \dot{Q} = 0$$

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

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:

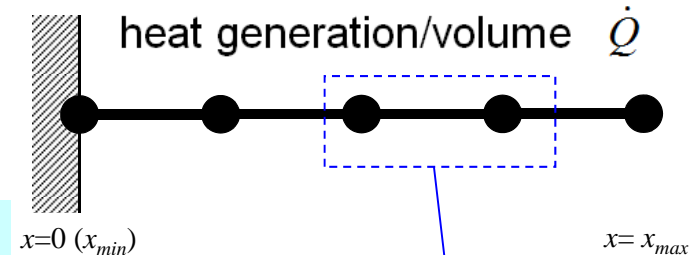
$$\int_V [N]^T \left\{ \lambda \left(\frac{d^2 T}{dx^2} \right) + \dot{Q} \right\} dV = 0$$



Galerkin Method (2/4)

- Green's Theorem (1D)

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

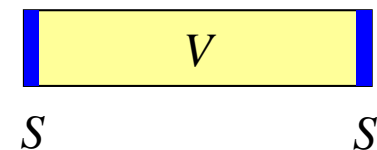


- Apply this to the 1st part of eqn with 2nd-order diff.:

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

- Consider the following terms:

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



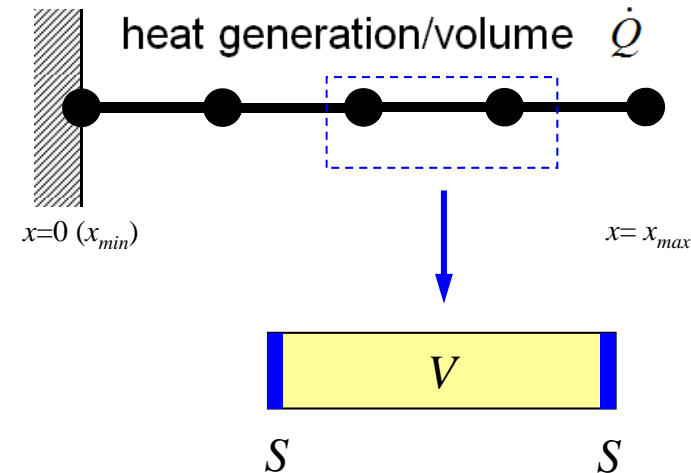
: Heat flux at element surface [QL⁻²T⁻¹]

Galerkin Method (3/4)

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

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

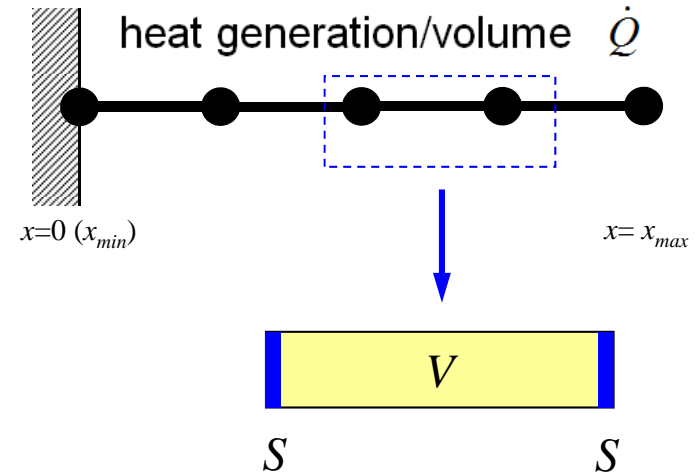
$$-\int_S \bar{q}[N]^T dS + \int_V \dot{Q}[N]^T dV = 0$$



- 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)

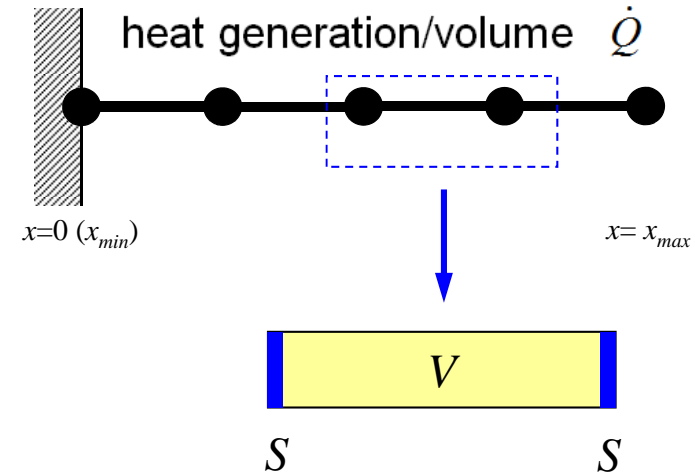
$$\begin{aligned}
 & - \int_V \lambda \left(\frac{d[N]^T}{dx} \frac{d[N]}{dx} \right) dV \cdot \{\phi\} \\
 & \boxed{- \int_S \bar{q} [N]^T dS} + \int_V \dot{Q} [N]^T dV = 0
 \end{aligned}$$



- 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: $[k]$

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

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

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

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

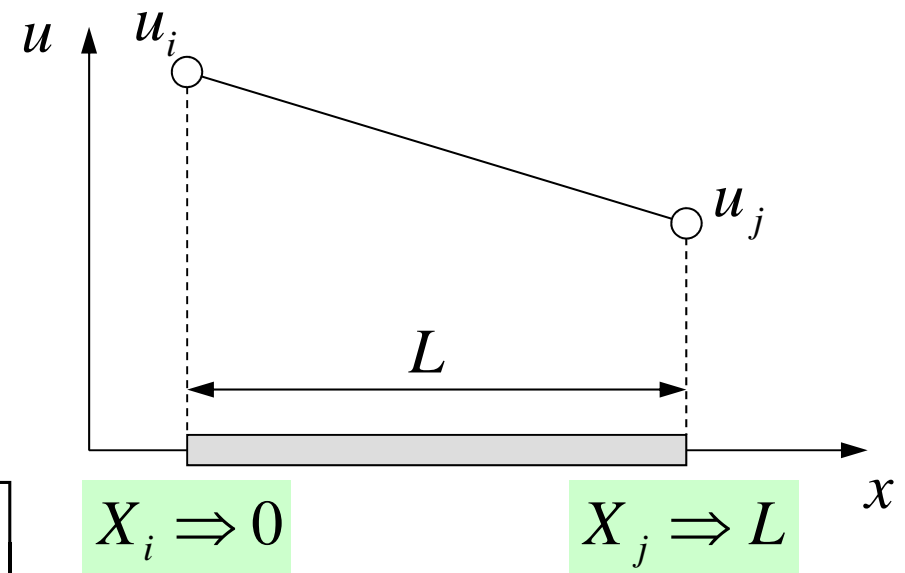
2x1 matrix

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)

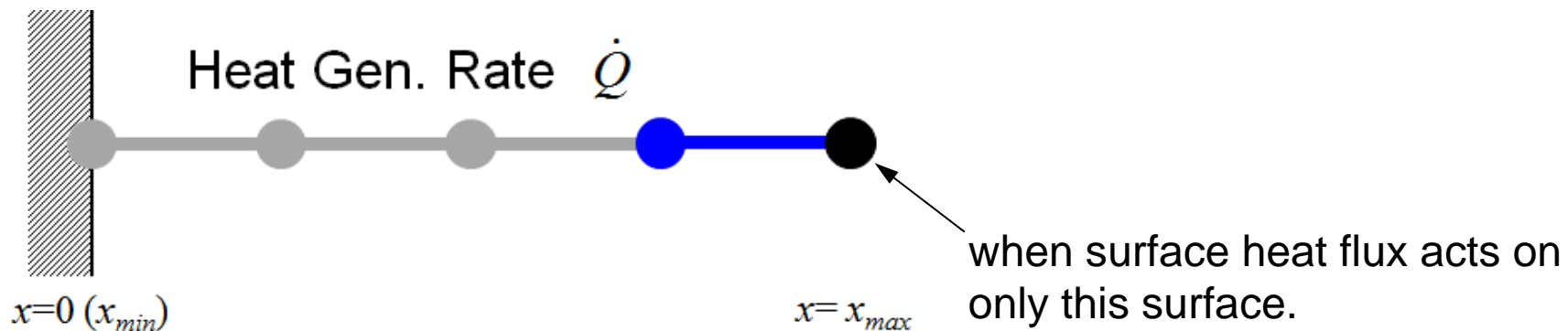
$$N_i = \left(\frac{X_j - x}{L} \right), \quad N_j = \left(\frac{x - X_i}{L} \right) \quad \frac{dN_i}{dx} = \left(\frac{-1}{L} \right), \quad \frac{dN_j}{dx} = \left(\frac{1}{L} \right)$$

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

Heat Generation
(Volume)

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

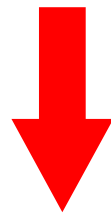
Surface Heat Flux



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.

ECCS2016 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-f.tar)  
>$ tar xvf fem-f.tar  
>$ cd fem-f/1d
```

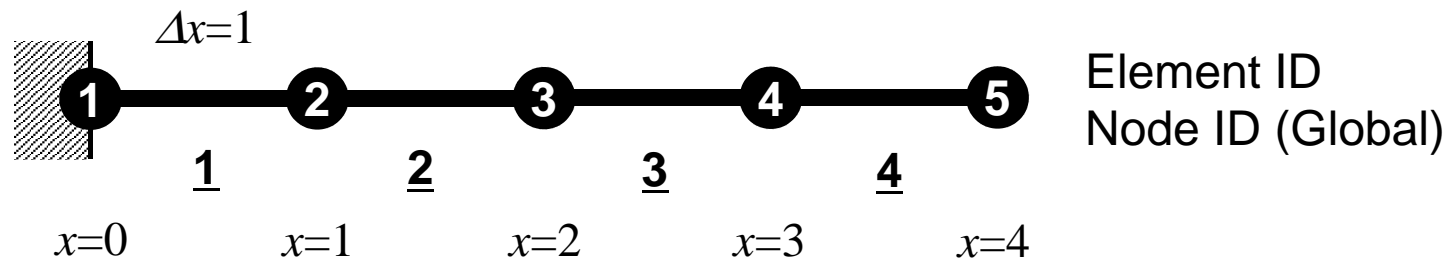
Compile & GO !

```
>$ cd <${E-TOP}>/fem-f/1d
>$ gfortran -O 1d.f
>$ ./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
```

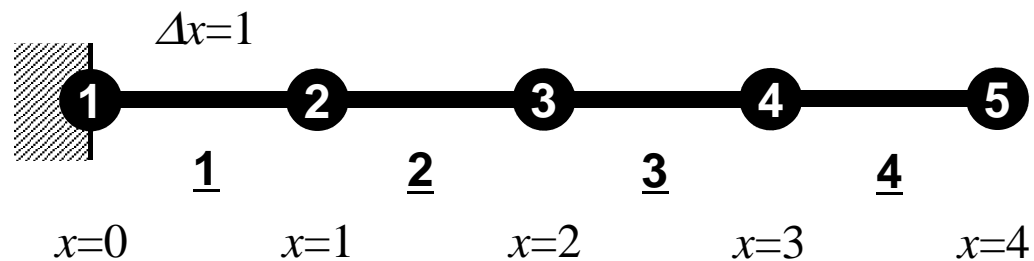
```
4 iters, RESID= 4.154074e-17
```

```
### 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

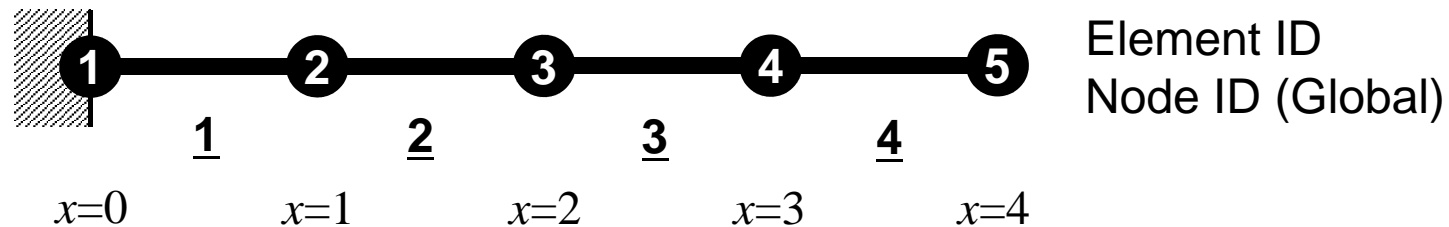
Analytical



Element ID
Node ID (Global)

Element Eqn's/Accumulation (1/3)

- 4 elements, 5 nodes



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

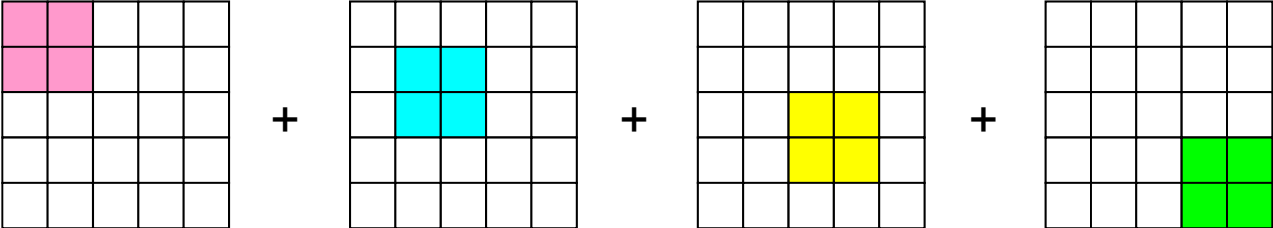
$$[k]^{(1)} = \frac{\lambda A}{L} \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \quad \{f\}^{(1)} = \frac{\dot{Q}AL}{2} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$$

- As for Element-4:

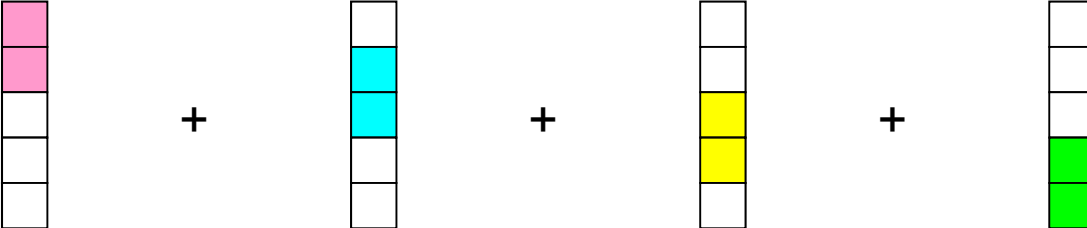
$$[k]^{(4)} = \frac{\lambda A}{L} \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \quad \{f\}^{(4)} = \frac{\dot{Q}AL}{2} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$$

Element Eqn's/Accumulation (2/3)

- Element-by-Element Accumulation:

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


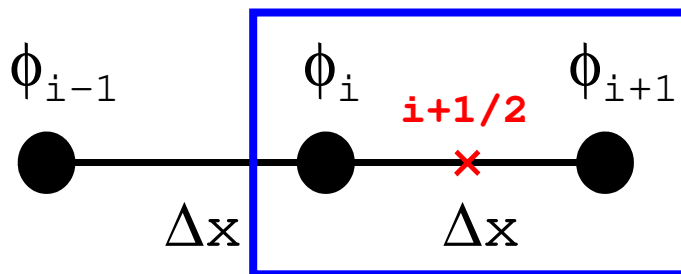
The diagram illustrates the assembly of the global stiffness matrix $[K]$ as a sum of four element matrices $[k]^{(e)}$. Each element matrix is represented as a 4x4 grid with a 2x2 colored block indicating the element's contribution. The first element (pink) is in the top-left corner. The second (cyan) is in the center. The third (yellow) is in the bottom-right. The fourth (green) is in the bottom-right corner.

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


The diagram illustrates the assembly of the global force vector $\{F\}$ as a sum of four element force vectors $\{f\}^{(e)}$. Each element force vector is represented as a 4x1 column vector with a colored block indicating the element's contribution. The first element (pink) is in the top two rows. The second (cyan) is in the second and third rows. The third (yellow) is in the third and fourth rows. The fourth (green) is in the bottom two rows.

2nd –Order Differentiation in FDM

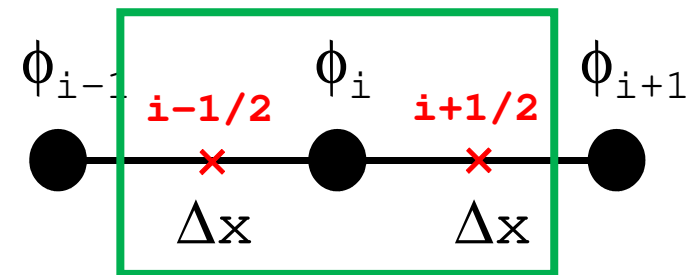
- Approximate Derivative at x (center of i and $i+1$)



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

$\Delta x \rightarrow 0$: 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.

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

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

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

Accumulation to Global/overall Matrix

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

D	X			X	X										
X	D	X		X	X	X									
	X	D	X		X	X	X								
		X	D			X	X								
X	X			D	X			X	X						
X	X	X		X	D	X		X	X	X					
X	X	X		X	D	X		X	X	X					
				X	X			D	X			X	X		
				X	X	X		X	D	X		X	X	X	
					X	X			X	D		X	X	X	
								X	X			D	X		
								X	X	X		X	D	X	
										X	X		X	D	

- 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  $i = 1, 2, \dots$ 
   $z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
  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

```

- 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  $i = 1, 2, \dots$ 
   $z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} \cdot z^{(i-1)}$ 
  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

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: $x^{(0)}$, 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 $i \neq j$

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\alpha_k, \beta_k$$

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:

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

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

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

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

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

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

Preconditioning in PCG

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
  solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
  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

```

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

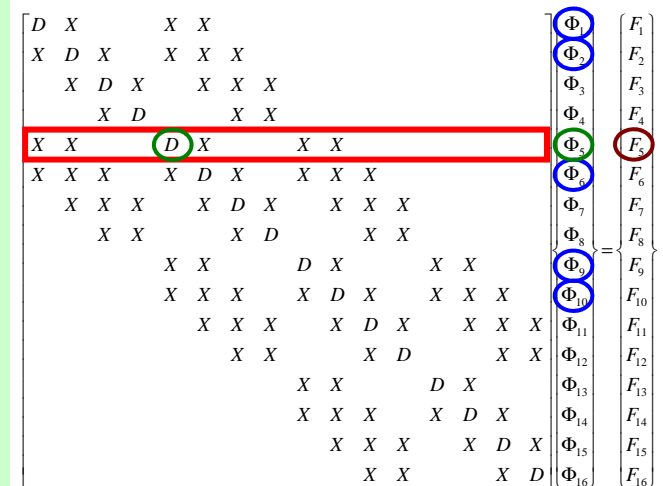
Compressed Row Storage (CRS)

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): Fortran

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

N= 8

对角成分

Diag(1) = 1.1
 Diag(2) = 3.6
 Diag(3) = 5.7
 Diag(4) = 9.8
 Diag(5) = 11.5
 Diag(6) = 12.4
 Diag(7) = 23.1
 Diag(8) = 51.3

Compressed Row Storage (CRS)

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

Compressed Row Storage (CRS)

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

NPLU= 25
(=index(N))

$index(i-1) + 1^{th} \sim index(i)^{th}$
Non-Zero Off-Diag. Components corresponding to i -th row

Compressed Row Storage (CRS)

		# Non-Zero Off-Diag.	index (i) =
1	1.1 ①	2 ②,1 ⑤,2	0
2	3.6 ②	4 ①,3 ④,4 ⑥,5 ⑧,6	2
3	5.7 ③	2 ⑤,7 ⑦,8	6
4	9.8 ④	3 ②,9 ⑤,10 ⑥,11	8
5	11.5 ⑤	4 ①,12 ②,13 ③,14 ⑦,15	11
6	12.4 ⑥	2 ③,16 ⑦,17	15
7	23.1 ⑦	4 ②,18 ③,19 ⑥,20 ⑧,21	17
8	51.3 ⑧	4 ②,22 ③,23 ④,24 ⑥,25	21
			25

NPLU= 25
(=index (N))

$\text{index}(i-1) + 1^{\text{th}} \sim \text{index}(i)^{\text{th}}$
Non-Zero Off-Diag. Components corresponding to i -th row

Compressed Row Storage (CRS)

1	1.1 ①	2.4 ②,1	3.2 ⑤,2		
2	3.6 ②	4.3 ①,3	2.5 ④,4	3.7 ⑥,5	9.1 ⑧,6
3	5.7 ③	1.5 ⑤,7	3.1 ⑦,8		
4	9.8 ④	4.1 ②,9	2.5 ⑤,10	2.7 ⑥,11	
5	11.5 ⑤	3.1 ①,12	9.5 ②,13	10.4 ③,14	4.3 ⑦,15
6	12.4 ⑥	6.5 ③,16	9.5 ⑦,17		
7	23.1 ⑦	6.4 ②,18	2.5 ③,19	1.4 ⑥,20	13.1 ⑧,21
8	51.3 ⑧	9.5 ②,22	1.3 ③,23	9.6 ④,24	3.1 ⑥,25

Example:

`item(7) = 5, AMAT(7) = 1.5`

`item(19) = 3, AMAT(19) = 2.5`

Compressed Row Storage (CRS)

1	1.1 ①	2.4 ②,1	3.2 ⑤,2		
2	3.6 ②	4.3 ①,3	2.5 ④,4	3.7 ⑥,5	9.1 ⑧,6
3	5.7 ③	1.5 ⑤,7	3.1 ⑦,8		
4	9.8 ④	4.1 ②,9	2.5 ⑤,10	2.7 ⑥,11	
5	11.5 ⑤	3.1 ①,12	9.5 ②,13	10.4 ③,14	4.3 ⑦,15
6	12.4 ⑥	6.5 ③,16	9.5 ⑦,17		
7	23.1 ⑦	6.4 ②,18	2.5 ③,19	1.4 ⑥,20	13.1 ⑧,21
8	51.3 ⑧	9.5 ②,22	1.3 ③,23	9.6 ④,24	3.1 ⑥,25

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)= D(i)*X(i)
  do k= index(i-1)+1, index(i)
    Y(i)= Y(i) + AMAT(k)*X(item(k))
  enddo
enddo
```


- 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.f (1/6)

variables and arrays

```
!C
!C 1D Steady-State Heat Transfer
!C FEM with Piece-wise Linear Elements
!C CG (Conjugate Gradient) Method
!C
!C  $d/dx(CdT/dx) + Q = 0$ 
!C  $T=0@x=0$ 
!C
program heat1D
implicit REAL*8 (A-H, O-Z)

integer :: N, NPLU, ITERmax
integer :: R, Z, P, Q, DD

real(kind=8) :: dX, RESID, EPS
real(kind=8) :: AREA, QV, COND
real(kind=8), dimension(:), allocatable :: PHI, RHS, X
real(kind=8), dimension(:), allocatable :: DIAG, AMAT
real(kind=8), dimension(:, :), allocatable :: W

real(kind=8), dimension(2, 2) :: KMAT, EMAT

integer, dimension(:), allocatable :: ICELNOD
integer, dimension(:), allocatable :: INDEX, ITEM
```

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	(N, 4)	○	Work Array for CG
Amat	R	NPLU	○	Off-Diagonal Components (Value)
Index	I	0:N	○	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.f (2/6)

Initialization, Allocation of Arrays

```
!C
!C +-----+
!C | INIT. |
!C +-----+
!C===
```

```
open (11, file='input.dat', status='unknown')
read (11,*) NE
read (11,*) dX, QV, AREA, COND
read (11,*) ITERmax
read (11,*) EPS
close (11)
```

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

```
N= NE + 1
allocate (PHI(N), DIAG(N), AMAT(2*N-2), RHS(N))
allocate (ICELNOD(2*NE), X(N))
allocate (INDEX(0:N), ITEM(2*N-2), W(N,4))
```

```
PHI = 0. d0
AMAT= 0. d0
DIAG= 0. d0
RHS= 0. d0
X= 0. d0
```



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

Program: 1d.f (2/6)

Initialization, Allocation of Arrays

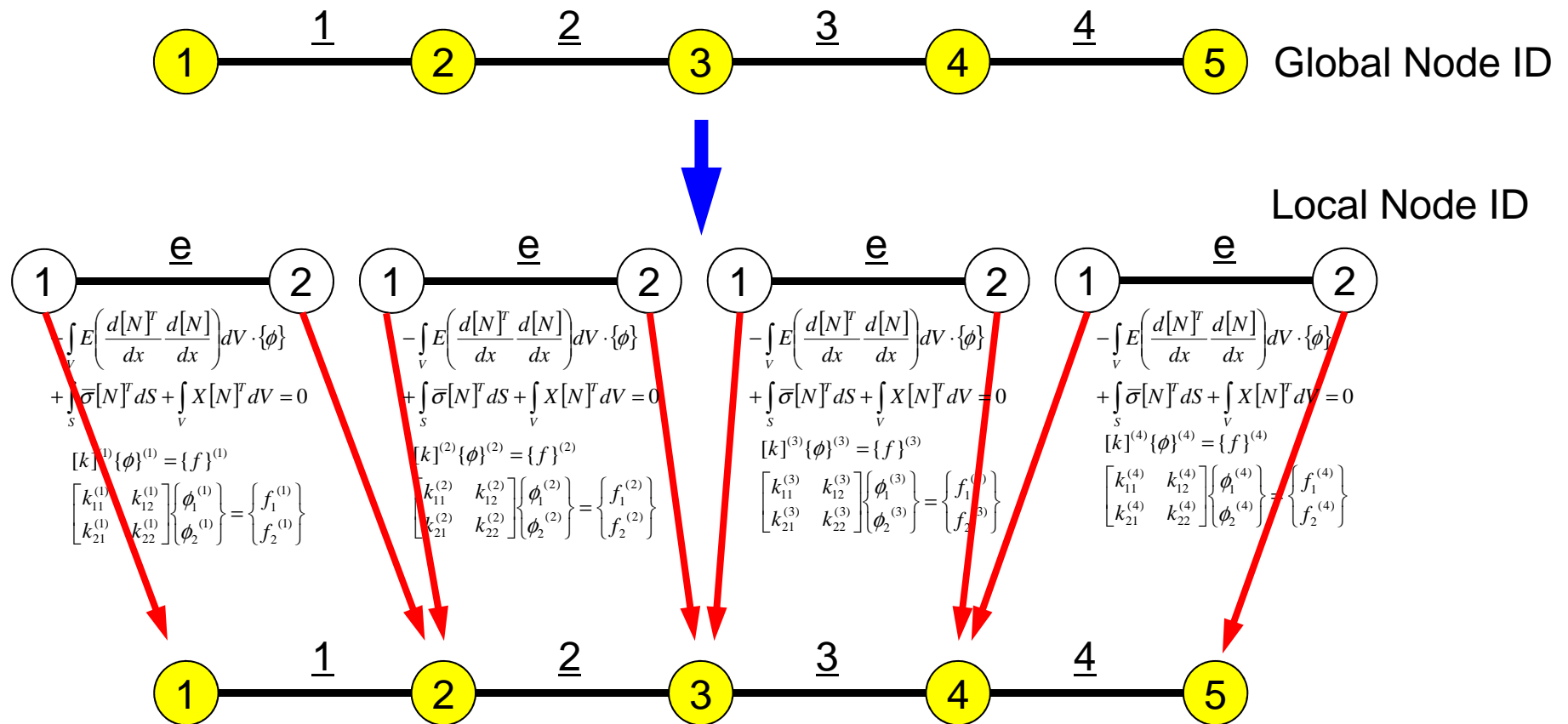
```
!C
!C +-----+
!C | INIT. |
!C +-----+
!C===
      open (11, file='input.dat', status='unknown')
      read (11,*) NE
      read (11,*) dX, QV, AREA, COND
      read (11,*) ITERmax
      read (11,*) EPS
      close (11)
```

```
      N= NE + 1
      allocate (PHI(N), DIAG(N), AMAT(2*N-2), RHS(N))
      allocate (ICELNOD(2*NE), X(N))
      allocate (INDEX(0:N), ITEM(2*N-2), W(N,4))
```

```
      PHI = 0. d0
      AMAT= 0. d0
      DIAG= 0. d0
      RHS= 0. d0
      X= 0. d0
```

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

Element/Global Operations

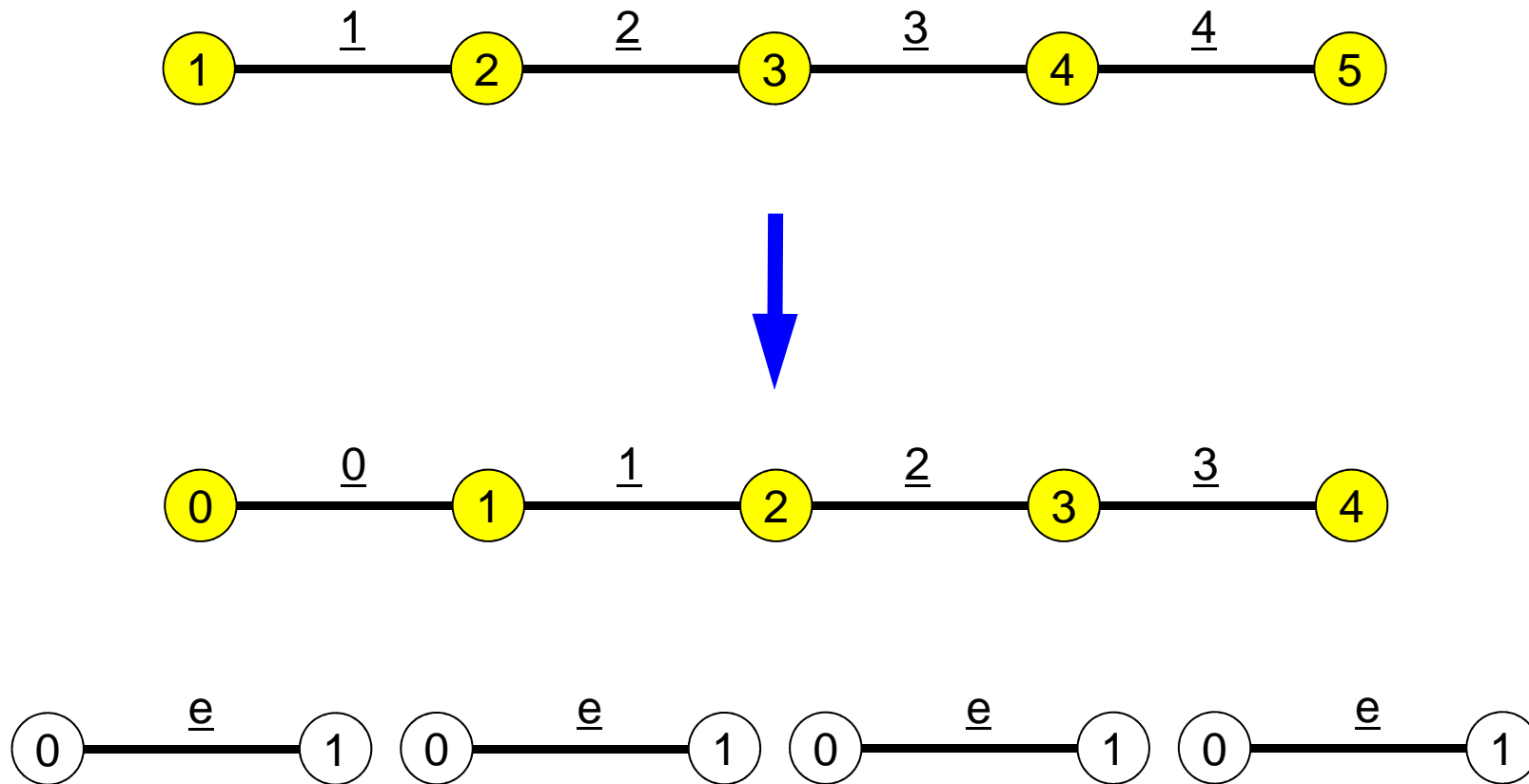


$$[K]\{\Phi\} = \{F\}$$

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

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.f (2/6)

Initialization, Allocation of Array

```

!C
!C +-----+
!C | INIT. |
!C +-----+
!C===
open (11, file='input.dat', status='unknown')
read (11,*) NE
read (11,*) dX, QV, AREA, COND
read (11,*) ITERmax
read (11,*) EPS
close (11)

```

```

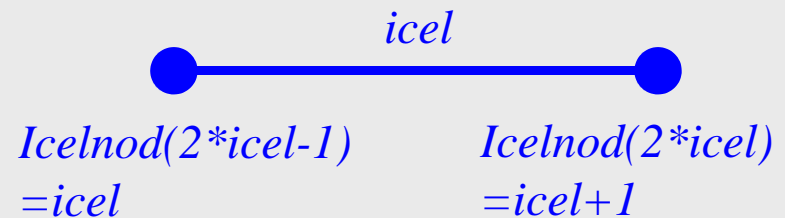
N= NE + 1
allocate (PHI(N), DIAG(N), AMAT(2*N-2), RHS(N))
allocate (ICELNOD(2*NE), X(N))
allocate (INDEX(0:N), ITEM(2*N-2), W(N,4))

```

```

PHI = 0. d0
AMAT= 0. d0
DIAG= 0. d0
RHS= 0. d0
X= 0. d0

```



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.f (3/6)

Initialization, Allocation of Arrays (cont.)

```
do i= 1, N
  X(i)= dfloat(i-1)*dX
enddo

do icel= 1, NE
  ICELNOD(2*icel-1)= icel
  ICELNOD(2*icel )= icel + 1
enddo

KMAT (1, 1)= +1. d0
KMAT (1, 2)= -1. d0
KMAT (2, 1)= -1. d0
KMAT (2, 2)= +1. d0
```

x: X-coordinate
component of each node

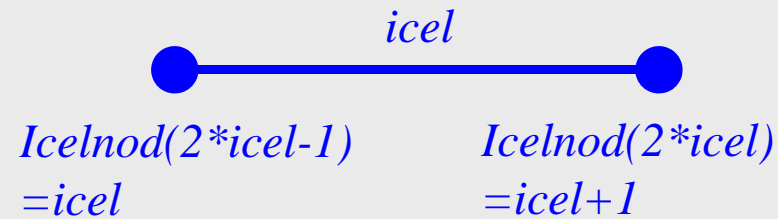
Program: 1d.f (3/6)

Initialization, Allocation of Arrays (cont.)

```
do i= 1, N
  X(i)= dfloat(i-1)*dX
enddo
```

```
do icel= 1, NE
  ICELNOD(2*icel-1)= icel
  ICELNOD(2*icel )= icel + 1
enddo
```

```
KMAT (1, 1)= +1. d0
KMAT (1, 2)= -1. d0
KMAT (2, 1)= -1. d0
KMAT (2, 2)= +1. d0
```



Program: 1d.f (3/6)

Initialization, Allocation of Arrays (cont.)

```
do i= 1, N
  X(i)= dfloat(i-1)*dX
enddo
```

```
do icel= 1, NE
  ICELNOD(2*icel-1)= icel
  ICELNOD(2*icel )= icel + 1
enddo
```

```
KMAT (1, 1)= +1. d0
KMAT (1, 2)= -1. d0
KMAT (2, 1)= -1. d0
KMAT (2, 2)= +1. d0
```

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

[Kmat]

Program: 1d.f (4/6)

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

```
!C
!C +-----+
!C | CONNECTIVITY |
!C +-----+
!C===
```

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

INDEX = 2

INDEX(0) = 0
INDEX(1) = 1
INDEX(N) = 1

```
do i= 1, N
  INDEX(i)= INDEX(i) + INDEX(i-1)
enddo
```

NPLU= INDEX(N)

```
do i= 1, N
  jS= INDEX(i-1)
  if (i.eq.1) then
    ITEM(jS+1)= i+1
  else if
& (i.eq.N) then
    ITEM(jS+1)= i-1
  else
    ITEM(jS+1)= i-1
    ITEM(jS+2)= i+1
  endif
enddo
```

```
!C===
```

						# Non-Zero Off-Diag.	index(0) = 0
1	1.1 ①	2.4 ②	3.2 ⑤			2	index(1) = 2
2	3.6 ②	4.3 ①	2.5 ④	3.7 ⑥	9.1 ⑧	4	index(2) = 6
3	5.7 ③	1.5 ⑤	3.1 ⑦			2	index(3) = 8
4	9.8 ④	4.1 ②	2.5 ⑤	2.7 ⑥		3	index(4) = 11
5	11.5 ⑤	3.1 ①	9.5 ②	10.4 ③	4.3 ⑦	4	index(5) = 15
6	12.4 ⑥	6.5 ③	9.5 ⑦			2	index(6) = 17
7	23.1 ⑦	6.4 ②	2.5 ③	1.4 ⑥	13.1 ⑧	4	index(7) = 21
8	51.3 ⑧	9.5 ②	1.3 ③	9.6 ④	3.1 ⑥	4	index(8) = 25

$index(i-1)+1^{th} \sim index(i)^{th}$
 Non-Zero Off-Diag. Components corresponding to *i*-th row

Program: 1d.f (4/6)

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

```
!C
!C +-----+
!C | CONNECTIVITY |
!C +-----+
!C===
```

INDEX = 2

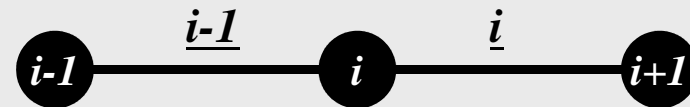
INDEX(0) = 0
 INDEX(1) = 1
 INDEX(N) = 1

```
do i= 1, N
    INDEX(i) = INDEX(i) + INDEX(i-1)
enddo
```

NPLU = INDEX(N)

```
do i= 1, N
    jS = INDEX(i-1)
    if (i.eq.1) then
        ITEM(jS+1) = i+1
    else if
    & (i.eq.N) then
        ITEM(jS+1) = i-1
    else
        ITEM(jS+1) = i-1
        ITEM(jS+2) = i+1
    endif
enddo
```

```
!C===
```



						# Non-Zero Off-Diag.	index(0) = 0
1	1.1 ①	2.4 ②	3.2 ⑤			2	index(1) = 2
2	3.6 ②	4.3 ①	2.5 ④	3.7 ⑥	9.1 ⑧	4	index(2) = 6
3	5.7 ③	1.5 ⑤	3.1 ⑦			2	index(3) = 8
4	9.8 ④	4.1 ②	2.5 ⑤	2.7 ⑥		3	index(4) = 11
5	11.5 ⑤	3.1 ①	9.5 ②	10.4 ③	4.3 ⑦	4	index(5) = 15
6	12.4 ⑥	6.5 ③	9.5 ⑦			2	index(6) = 17
7	23.1 ⑦	6.4 ②	2.5 ③	1.4 ⑥	13.1 ⑧	4	index(7) = 21
8	51.3 ⑧	9.5 ②	1.3 ③	9.6 ④	3.1 ⑥	4	index(8) = 25

$index(i-1) + 1^{th} \sim index(i)^{th}$
 Non-Zero Off-Diag. Components corresponding to i -th row

Program: 1d.f (5/6)

Element Matrix ~ Global Matrix

```

!C +-----+
!C | MATRIX ASSEMBLE |
!C +-----+
!C===
do icel= 1, NE
  in1= ICELNOD (2*icel-1)
  in2= ICELNOD (2*icel )
  X1 = X(in1)
  X2 = X(in2)
  DL = dabs(X2-X1)

  cK= AREA*COND/DL
  EMAT (1, 1)= Ck*KMAT (1, 1)
  EMAT (1, 2)= Ck*KMAT (1, 2)
  EMAT (2, 1)= Ck*KMAT (2, 1)
  EMAT (2, 2)= Ck*KMAT (2, 2)

  DIAG(in1)= DIAG(in1) + EMAT (1, 1)
  DIAG(in2)= DIAG(in2) + EMAT (2, 2)

  if (icel.eq.1) then
    k1= INDEX(in1-1) + 1
  else
    k1= INDEX(in1-1) + 2
  endif
  k2= INDEX(in2-1) + 1

  AMAT (k1)= AMAT (k1) + EMAT (1, 2)
  AMAT (k2)= AMAT (k2) + EMAT (2, 1)

  QN= 0.50d0*QV*AREA*DL
  RHS (in1)= RHS (in1) + QN
  RHS (in2)= RHS (in2) + QN
enddo
!C===

```



Program: 1d.f (5/6)

Element Matrix ~ Global Matrix

```

!C +-----+
!C | MATRIX ASSEMBLE |
!C +-----+
!C===
do icel= 1, NE
  in1= ICELNOD(2*icel-1)
  in2= ICELNOD(2*icel )
  X1 = X(in1)
  X2 = X(in2)
  DL = dabs(X2-X1)

  cK= AREA*COND/DL
  EMAT(1,1)= Ck*KMAT(1,1)
  EMAT(1,2)= Ck*KMAT(1,2)
  EMAT(2,1)= Ck*KMAT(2,1)
  EMAT(2,2)= Ck*KMAT(2,2)

  DIAG(in1)= DIAG(in1) + EMAT(1,1)
  DIAG(in2)= DIAG(in2) + EMAT(2,2)

  if (icel.eq.1) then
    k1= INDEX(in1-1) + 1
  else
    k1= INDEX(in1-1) + 2
  endif
  k2= INDEX(in2-1) + 1

  AMAT(k1)= AMAT(k1) + EMAT(1,2)
  AMAT(k2)= AMAT(k2) + EMAT(2,1)

  QN= 0.50d0*QV*AREA*DL
  RHS(in1)= RHS(in1) + QN
  RHS(in2)= RHS(in2) + QN
enddo
!C===

```



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

Program: 1d.f (5/6)

Element Matrix ~ Global Matrix

```

!C +-----+
!C | MATRIX ASSEMBLE |
!C +-----+
!C===
do icel= 1, NE
  in1= ICELNOD(2*icel-1)
  in2= ICELNOD(2*icel )
  X1 = X(in1)
  X2 = X(in2)
  DL = dabs(X2-X1)

  cK= AREA*COND/DL
  EMAT(1,1)= Ck*KMAT(1,1)
  EMAT(1,2)= Ck*KMAT(1,2)
  EMAT(2,1)= Ck*KMAT(2,1)
  EMAT(2,2)= Ck*KMAT(2,2)

  DIAG(in1)= DIAG(in1) + EMAT(1,1)
  DIAG(in2)= DIAG(in2) + EMAT(2,2)

  if (icel.eq.1) then
    k1= INDEX(in1-1) + 1
  else
    k1= INDEX(in1-1) + 2
  endif
  k2= INDEX(in2-1) + 1

  AMAT(k1)= AMAT(k1) + EMAT(1,2)
  AMAT(k2)= AMAT(k2) + EMAT(2,1)

  QN= 0.50d0*QV*AREA*DL
  RHS(in1)= RHS(in1) + QN
  RHS(in2)= RHS(in2) + QN
enddo
!C===

```



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

Program: 1d.f (5/6)

Element Matrix ~ Global Matrix

```

!C +-----+
!C | MATRIX ASSEMBLE |
!C +-----+
!C===
do icel= 1, NE
  in1= ICELNOD(2*icel-1)
  in2= ICELNOD(2*icel)
  X1 = X(in1)
  X2 = X(in2)
  DL = dabs(X2-X1)

  cK= AREA*COND/DL
  EMAT(1,1)= Ck*KMAT(1,1)
  EMAT(1,2)= Ck*KMAT(1,2)
  EMAT(2,1)= Ck*KMAT(2,1)
  EMAT(2,2)= Ck*KMAT(2,2)

  DIAG(in1)= DIAG(in1) + EMAT(1,1)
  DIAG(in2)= DIAG(in2) + EMAT(2,2)

  if (icel.eq.1) then
    k1= INDEX(in1-1) + 1
  else
    k1= INDEX(in1-1) + 2
  endif
  k2= INDEX(in2-1) + 1

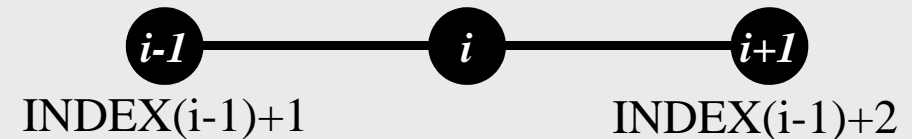
  AMAT(k1)= AMAT(k1) + EMAT(1,2)
  AMAT(k2)= AMAT(k2) + EMAT(2,1)

  QN= 0.5d0*QV*AREA*DL
  RHS(in1)= RHS(in1) + QN
  RHS(in2)= RHS(in2) + QN
enddo
!C===

```



Non-zero Off-Diag. at i -th row:
Index(i-1)+1, Index(i-1)+2



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

General Elements: k1

“in2” as a off-diag. component of “in1”

```

!C +-----+
!C | MATRIX ASSEMBLE |
!C +-----+
!C===
do icel= 1, NE
  in1= ICELNOD(2*icel-1)
  in2= ICELNOD(2*icel )
  X1 = X(in1)
  X2 = X(in2)
  DL = dabs(X2-X1)

  cK= AREA*COND/DL
  EMAT(1,1)= Ck*KMAT(1,1)
  EMAT(1,2)= Ck*KMAT(1,2)
  EMAT(2,1)= Ck*KMAT(2,1)
  EMAT(2,2)= Ck*KMAT(2,2)

  DIAG(in1)= DIAG(in1) + EMAT(1,1)
  DIAG(in2)= DIAG(in2) + EMAT(2,2)

  if (icel.eq.1) then
    k1= INDEX(in1-1) + 1
  else
    k1= INDEX(in1-1) + 2
  endif
  k2= INDEX(in2-1) + 1

  AMAT(k1)= AMAT(k1) + EMAT(1,2)
  AMAT(k2)= AMAT(k2) + EMAT(2,1)

  QN= 0.50d0*QV*AREA*DL
  RHS(in1)= RHS(in1) + QN
  RHS(in2)= RHS(in2) + QN
enddo
!C===

```



Non-zero Off-Diag. at i -th row:
Index(i-1)+1, Index(i-1)+2



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

General Elements: k2

“in1” as a off-diag. component of “in2”

```

!C +-----+
!C | MATRIX ASSEMBLE |
!C +-----+
!C===
do icel= 1, NE
  in1= ICELNOD(2*icel-1)
  in2= ICELNOD(2*icel )
  X1 = X(in1)
  X2 = X(in2)
  DL = dabs(X2-X1)

  cK= AREA*COND/DL
  EMAT(1, 1)= Ck*KMAT(1, 1)
  EMAT(1, 2)= Ck*KMAT(1, 2)
  EMAT(2, 1)= Ck*KMAT(2, 1)
  EMAT(2, 2)= Ck*KMAT(2, 2)

  DIAG(in1)= DIAG(in1) + EMAT(1, 1)
  DIAG(in2)= DIAG(in2) + EMAT(2, 2)

  if (icel.eq.1) then
    k1= INDEX(in1-1) + 1
  else
    k1= INDEX(in1-1) + 2
  endif
  k2= INDEX(in2-1) + 1

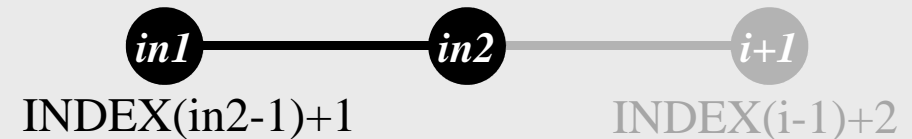
  AMAT(k1)= AMAT(k1) + EMAT(1, 2)
  AMAT(k2)= AMAT(k2) + EMAT(2, 1)

  QN= 0.50d0*QV*AREA*DL
  RHS(in1)= RHS(in1) + QN
  RHS(in2)= RHS(in2) + QN
enddo
!C===

```



Non-zero Off-Diag. at i -th row:
Index(i-1)+1, Index(i-1)+2



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

k2

0-th Element: k1

“in2” as a off-diag. component of “in1”

```

!C +-----+
!C | MATRIX ASSEMBLE |
!C +-----+
!C===
do icel= 1, NE
  in1= ICELNOD(2*icel-1)
  in2= ICELNOD(2*icel )
  X1 = X(in1)
  X2 = X(in2)
  DL = dabs(X2-X1)

  cK= AREA*COND/DL
  EMAT(1, 1)= Ck*KMAT(1, 1)
  EMAT(1, 2)= Ck*KMAT(1, 2)
  EMAT(2, 1)= Ck*KMAT(2, 1)
  EMAT(2, 2)= Ck*KMAT(2, 2)

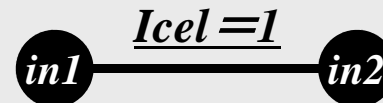
  DIAG(in1)= DIAG(in1) + EMAT(1, 1)
  DIAG(in2)= DIAG(in2) + EMAT(2, 2)

  if (icel.eq.1) then
    k1= INDEX(in1-1) + 1
  else
    k1= INDEX(in1-1) + 2
  endif
  k2= INDEX(in2-1) + 1

  AMAT(k1)= AMAT(k1) + EMAT(1, 2)
  AMAT(k2)= AMAT(k2) + EMAT(2, 1)

  QN= 0.50d0*QV*AREA*DL
  RHS(in1)= RHS(in1) + QN
  RHS(in2)= RHS(in2) + QN
enddo
!C===

```



Non-zero Off-Diag. at i -th row:
INDEX($i-1$)+1 only



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

Program: 1d.f (5/6)

RHS: Heat Generation Term

```

!C +-----+
!C | MATRIX ASSEMBLE |
!C +-----+
!C===
do icel= 1, NE
  in1= ICELNOD(2*icel-1)
  in2= ICELNOD(2*icel )
  X1 = X(in1)
  X2 = X(in2)
  DL = dabs(X2-X1)

  cK= AREA*COND/DL
  EMAT(1,1)= Ck*KMAT(1,1)
  EMAT(1,2)= Ck*KMAT(1,2)
  EMAT(2,1)= Ck*KMAT(2,1)
  EMAT(2,2)= Ck*KMAT(2,2)

  DIAG(in1)= DIAG(in1) + EMAT(1,1)
  DIAG(in2)= DIAG(in2) + EMAT(2,2)

  if (icel.eq.1) then
    k1= INDEX(in1-1) + 1
  else
    k1= INDEX(in1-1) + 2
  endif
  k2= INDEX(in2-1) + 1

  AMAT(k1)= AMAT(k1) + EMAT(1,2)
  AMAT(k2)= AMAT(k2) + EMAT(2,1)

  QN= 0.50d0*QV*AREA*DL
  RHS(in1)= RHS(in1) + QN
  RHS(in2)= RHS(in2) + QN
enddo
!C===

```



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

Program: 1d.f (6/6)

Dirichlet B.C. @ X=0

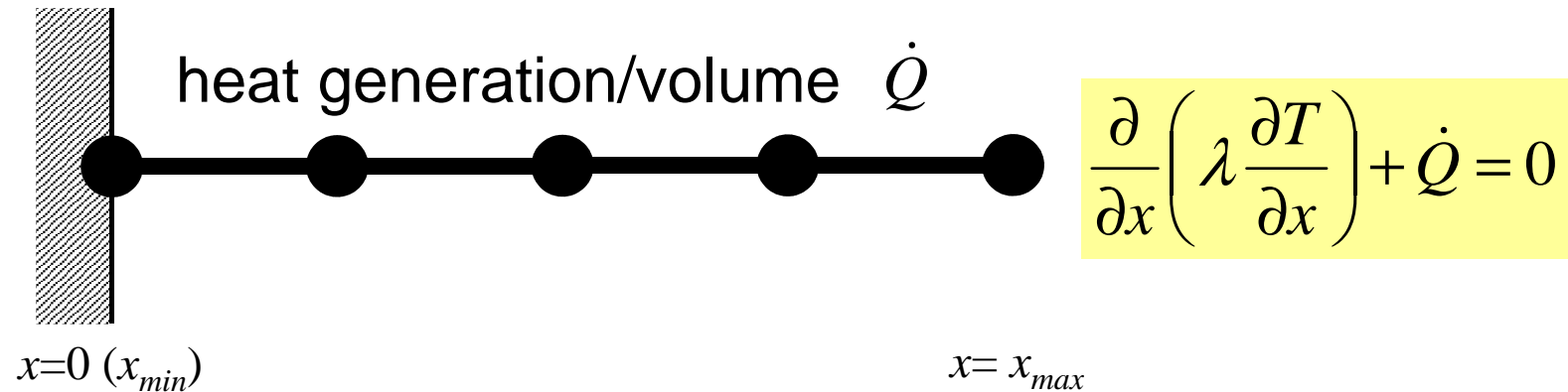
```
!C
!C +-----+
!C | BOUNDARY CONDITIONS |
!C +-----+
!C===

!C
!C-- X=Xmin
      i= 1
      jS= INDEX(i-1)

      AMAT(jS+1)= 0. d0
      DIAG(i)= 1. d0
      RHS (i)= 0. d0

      do k= 1, NPLU
        if (ITEM(k).eq. 1) AMAT(k)= 0. d0
      enddo
!C===
```

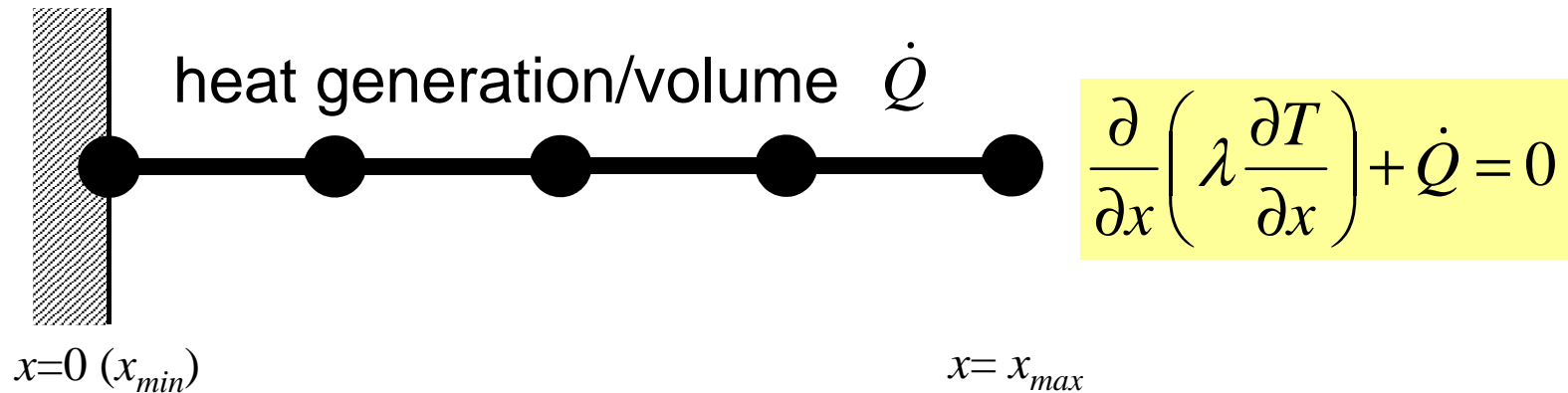

1D Steady State Heat Conduction



- **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 $x=0$

$$T_1 = 0 \text{ (or } T_0 = 0)$$



- **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)

Program: 1d.f (6/6)

Dirichlet B.C. @ X=0

```

!C
!C +-----+
!C | BOUNDARY CONDITIONS |
!C +-----+
!C===

!C
!C-- X=Xmin
  i= 1
  js= INDEX(i-1)

  AMAT (js+1)= 0. d0
  DIAG (i)= 1. d0
  RHS (i)= 0. d0

  do k= 1, NPLU
    if (ITEM(k).eq. 1) AMAT (k)= 0. d0
  enddo
!C===

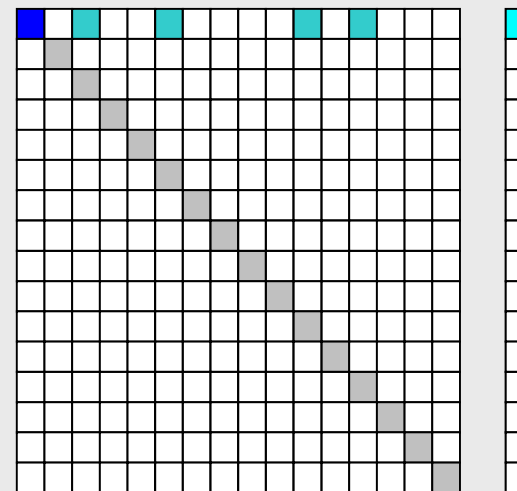
```

$$T_1=0$$

Diagonal Component=1

RHS=0

Off-Diagonal Components= 0.



Program: 1d.f (6/6)

Dirichlet B.C. @ X=0

```

!C
!C +-----+
!C | BOUNDARY CONDITIONS |
!C +-----+
!C===

!C
!C-- X=Xmin
      i= 1
      js= INDEX(i-1)

      AMAT (js+1)= 0. d0
      DIAG (i)= 1. d0
      RHS (i)= 0. d0

      do k= 1, NPLU
        if (ITEM(k).eq. 1) AMAT (k)= 0. d0
      enddo
!C===

```

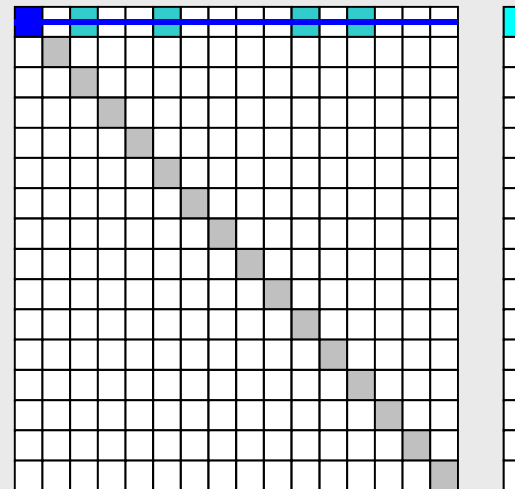
$$T_1=0$$

Diagonal Component=1

RHS=0

Off-Diagonal Components= 0.

Erase !



Program: 1d.f (6/6)

Dirichlet B.C. @ X=0

```

!C
!C +-----+
!C | BOUNDARY CONDITIONS |
!C +-----+
!C===

!C
!C-- X=Xmin
      i= 1
      js= INDEX(i-1)

      AMAT (js+1)= 0. d0
      DIAG (i)= 1. d0
      RHS (i)= 0. d0

      do k= 1, NPLU
        if (ITEM(k).eq.1) AMAT (k)= 0. d0
      enddo
!C===

```

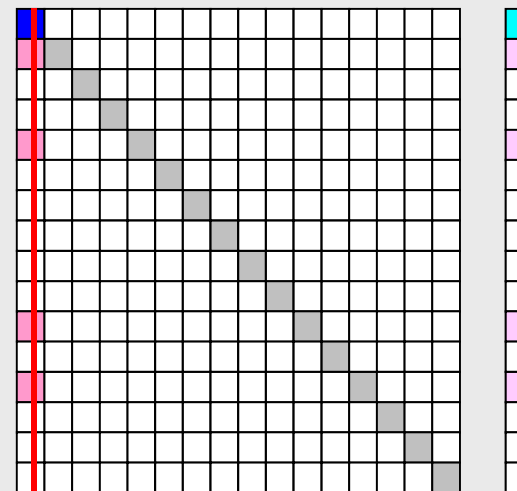
$$T_1=0$$

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 $T_1 \neq 0$

```
!C
!C +-----+
!C | BOUNDARY CONDITIONS |
!C +-----+
!C===
```

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

```
!C
!C-- X=Xmin
      i= 1
      jS= INDEX(i-1)

      AMAT(jS+1)= 0. d0
      DIAG(i)= 1. d0
      RHS (i)= PHImin
```

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

```
do i= 1, N
  do k= INDEX(i-1)+1, INDEX(i)
    if (ITEM(k).eq.1) then
      RHS (i)= RHS(i) - AMAT(k)*PHImin
      AMAT(k)= 0. d0
    endif
  enddo
enddo
!C===
```

if $T_1 \neq 0$

```
!C
!C +-----+
!C | BOUNDARY CONDITIONS |
!C +-----+
!C===
```

```
!C
!C-- X=Xmin
      i= 1
      jS= INDEX(i-1)
```

```
      AMAT(jS+1)= 0. d0
      DIAG(i)= 1. d0
      RHS (i)= PHImin
```

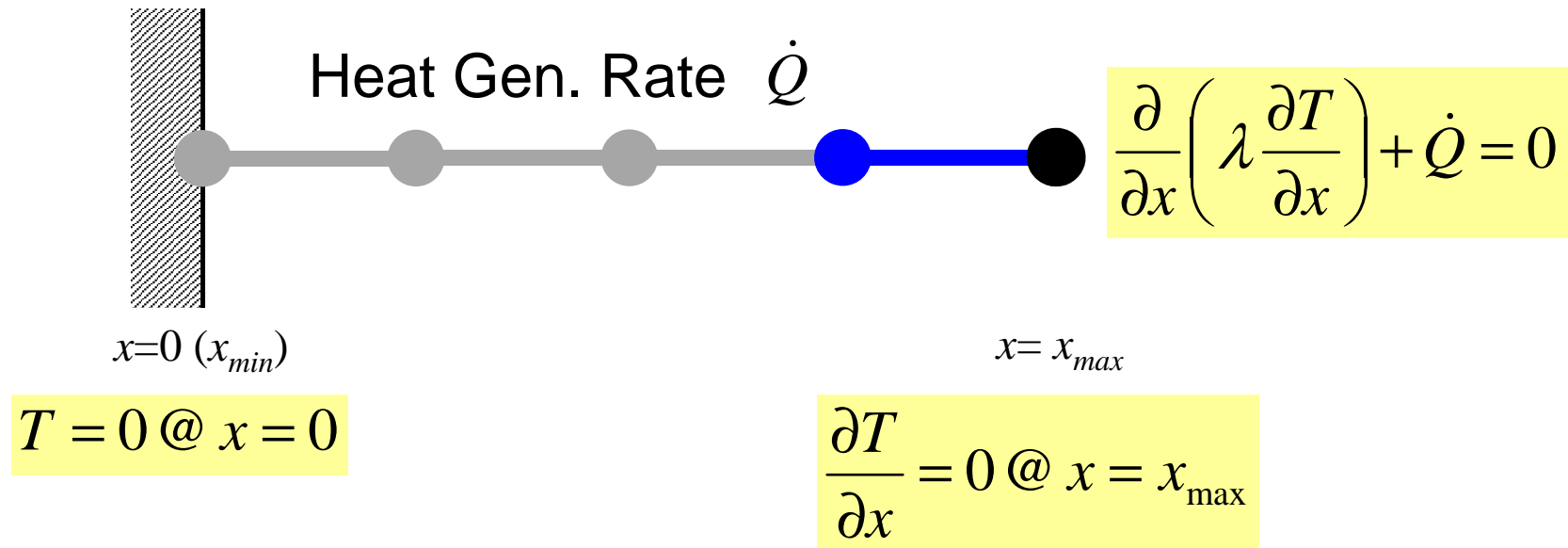
```
      do i= 1, N
        do k= INDEX(i-1)+1, INDEX(i)
          if (ITEM(k).eq.1) then
            RHS (i)= RHS(i) - AMAT(k)*PHImin
            AMAT(k)= 0. d0
          endif
        enddo
      enddo
```

```
!C===
```

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

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)

```
!C
!C +-----+
!C | CG iterations |
!C +-----+
!C===
      R = 1
      Z = 2
      Q = 2
      P = 3
      DD= 4

      do i= 1, N
        W(i,DD)= 1.0D0 / DIAG(i)
      enddo
```

```
W(i, 1) = W(i, R)   ⇒ {r}
W(i, 2) = W(i, Z)   ⇒ {z}
W(i, 2) = W(i, Q)   ⇒ {q}
W(i, 3) = W(i, P)   ⇒ {p}
W(i, 4) = W(i, DD)  ⇒ 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)

```
!C
!C +-----+
!C | CG iterations |
!C +-----+
!C===
      R = 1
      Z = 2
      Q = 2
      P = 3
      DD= 4

      do i= 1, N
        W(i,DD)= 1.0D0 / DIAG(i)
      enddo
```

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

$$\begin{aligned} W(i, 1) &= W(i, R) && \Rightarrow \{r\} \\ W(i, 2) &= W(i, Z) && \Rightarrow \{z\} \\ W(i, 2) &= W(i, Q) && \Rightarrow \{q\} \\ W(i, 3) &= W(i, P) && \Rightarrow \{p\} \\ W(i, 4) &= W(i, DD) && \Rightarrow 1/\{D\} \end{aligned}$$

CG Solver (2/6)

```
!C
!C-- {r0} = {b} - [A]{xini} |
```

!C 初期残差

```
do i= 1, N
  W(i,R) = DIAG(i)*PHI(i)
  do j= INDEX(i-1)+1, INDEX(i)
    W(i,R) = W(i,R) + AMAT(j)*PHI(ITEM(j))
  enddo
enddo
```

```
BNRM2= 0.0D0
```

```
do i= 1, N
  BNRM2 = BNRM2 + RHS(i) **2
  W(i,R) = RHS(i) - W(i,R)
enddo
```

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)

```

do iter= 1, ITERmax

!C
!C-- {z}= [Minv]{r}

do i= 1, N
  W(i,Z)= W(i,DD) * W(i,R)
enddo

!C
!C-- RHO= {r}{z}

RHO= 0.d0
do i= 1, N
  RHO= RHO + W(i,R)*W(i,Z)
enddo

```

```

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 (4/6)

```

!C
!C-- {p} = {z} if      ITER=1
!C  BETA= RHO / RH01  otherwise

      if ( iter.eq.1 ) then
        do i= 1, N
          W(i,P)= W(i,Z)
        enddo
      else
        BETA= RHO / RH01
        do i= 1, N
          W(i,P)= W(i,Z) + BETA*W(i,P)
        enddo
      endif

!C
!C-- {q}= [A] {p}

      do i= 1, N
        W(i,Q) = DIAG(i)*W(i,P)
        do j= INDEX(i-1)+1, INDEX(i)
          W(i,Q) = W(i,Q) + AMAT(j)*W(ITEM(j),P)
        enddo
      enddo

```

```

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)

```

!C
!C-- ALPHA= RHO / {p} {q}

      C1= 0.d0
      do i= 1, N
        C1= C1 + W(i, P)*W(i, Q)
      enddo
      ALPHA= RHO / C1

!C
!C-- {x}= {x} + ALPHA*{p}
!C  {r}= {r} - ALPHA*{q}

      do i= 1, N
        PHI (i)= PHI (i) + ALPHA * W(i, P)
        W(i, R)= W(i, R) - ALPHA * W(i, Q)
      enddo

```

```

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)

```

DNRM2 = 0.0
do i= 1, N
  DNRM2= DNRM2 + W(i,R)**2
enddo

RESID= dsqrt(DNRM2/BNRM2)

if ( RESID. le. EPS) goto 900
RHO1 = RHO  ρi-2

enddo
900 continue

```

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

Control Data input.dat

```

4          NE (Number of Elements)
1.0 1.0 1.0 1.0 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

```

```

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

```

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

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

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

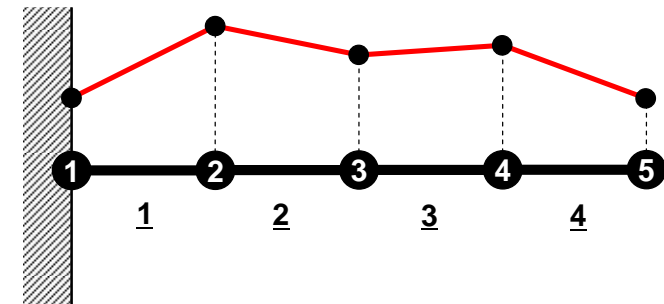
NE=8, dx=12.5

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

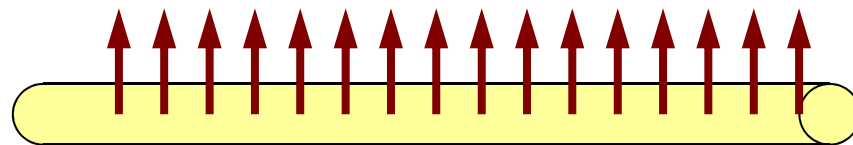
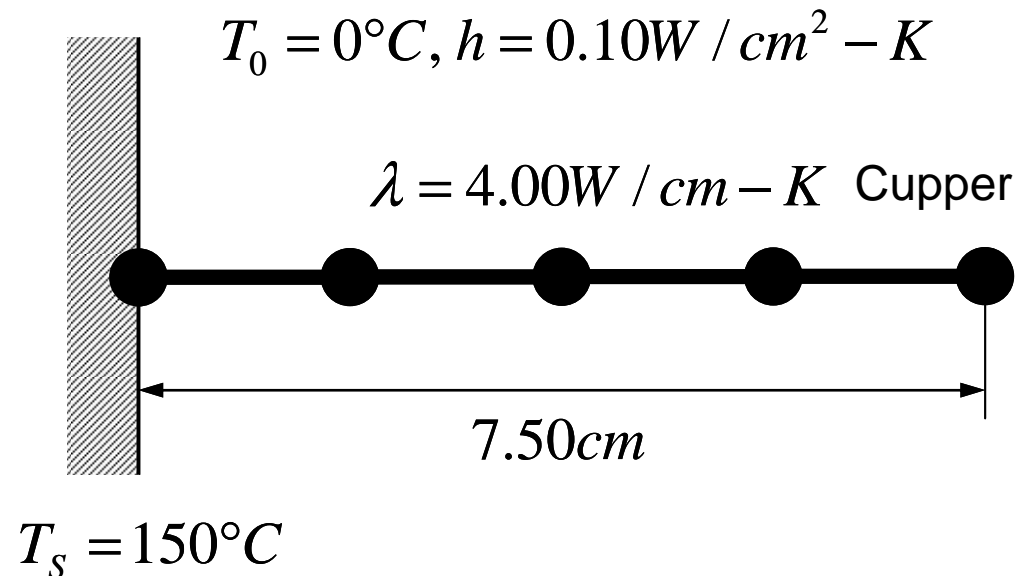
Remedies for Higher Accuracy

- 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)

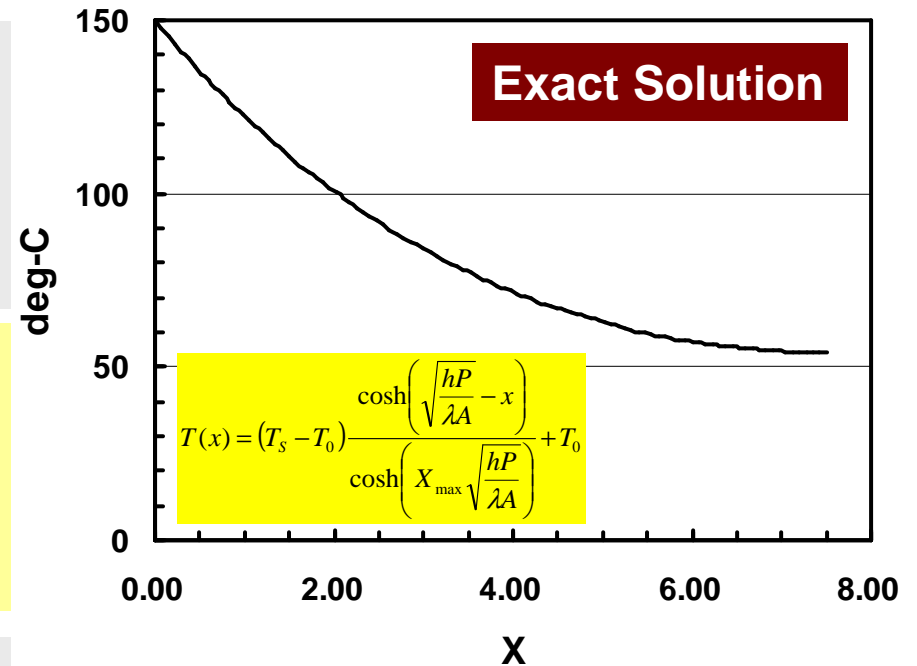
ID	X	FEM.	ANALYTICAL	ERR (%)
1	0.00000	150.00000	150.00000	0.00000
2	1.87500	102.62226	103.00165	0.25292
3	3.75000	73.82803	74.37583	0.36520
4	5.62500	58.40306	59.01653	0.40898
5	7.50000	53.55410	54.18409	0.41999

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.