1D-FEM in Fortran: Steady State Heat Conduction

Kengo Nakajima

Information Technology Center

Programming for Parallel Computing (616-2057)

Seminar on Advanced Computing (616-4009)

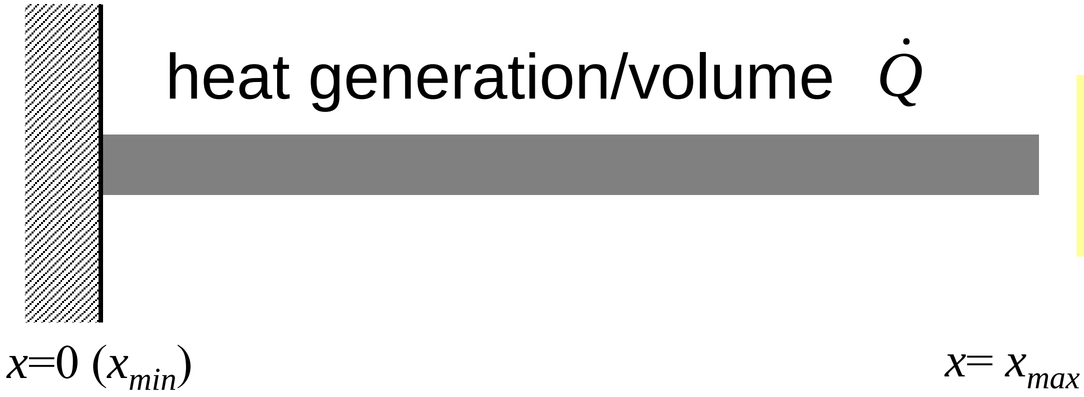
- 1D-code for Static Linear-Elastic Problems by Galerkin FEM
- Sparse Linear Solver
  - Conjugate Gradient Method
  - Preconditioning
- Storage of Sparse Matrices

- Program

Keywords

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

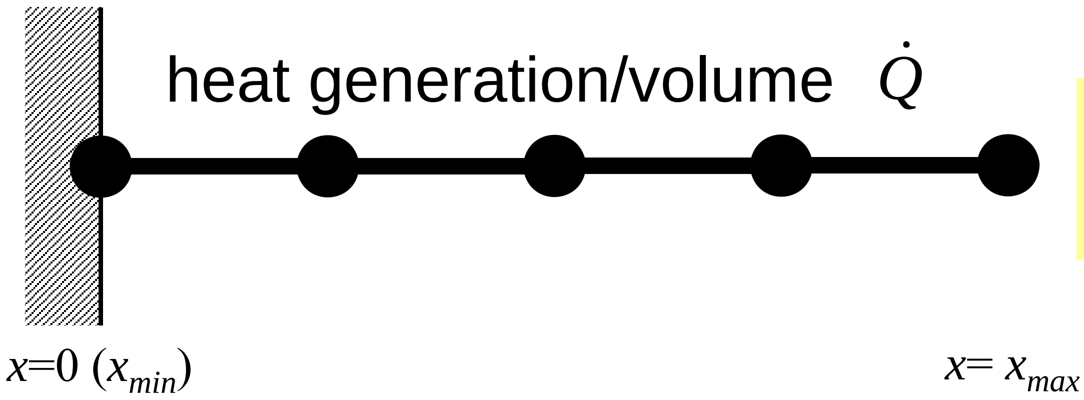
1D Steady State Heat Conduction



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

- 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} : $\dfrac{\partial T}{\partial x} = 0$ (Insulated)

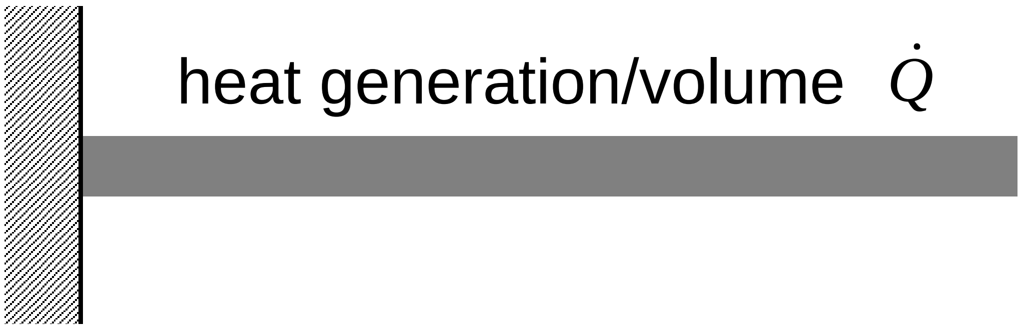
1D Steady State Heat Conduction



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

- 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)
  - $x=x_{max}$: $\dfrac{\partial T}{\partial x} = 0$ (Insulated)

Analytical Solution



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

x=0 (x_{min}) $\qquad$ x= x_{max}

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

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

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

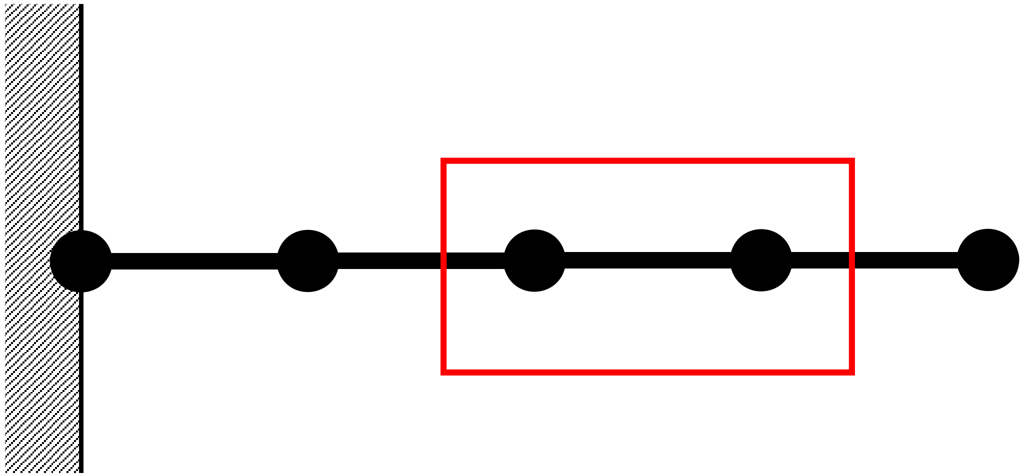
$$\lambda T=-\frac{1}{2}\dot{Q}x^2+C_1 x+C_2 \Rightarrow C_2=0, \quad T=0 @ x=0$$

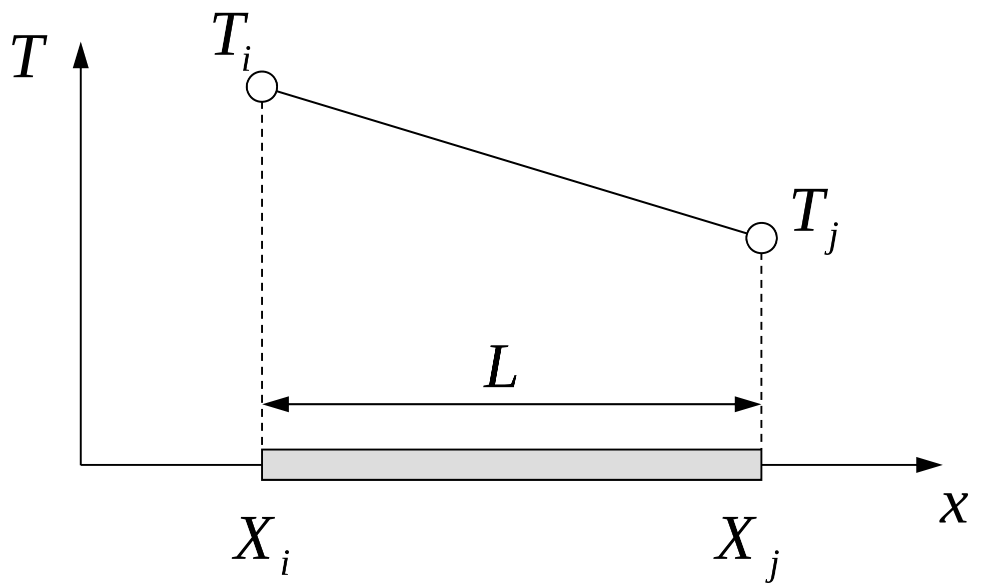
$$\therefore \quad T=-\frac{1}{2\lambda}\dot{Q}x^2+\frac{\dot{Q}x_{\max}}{\lambda}x$$

1D Linear Element (1/4)
一次元線形要素

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

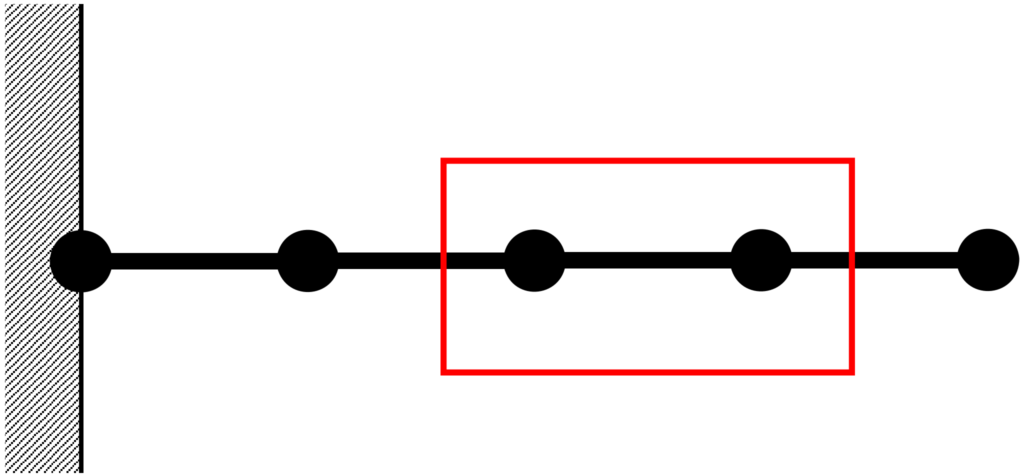
$$T = \alpha_1 + \alpha_2 x$$





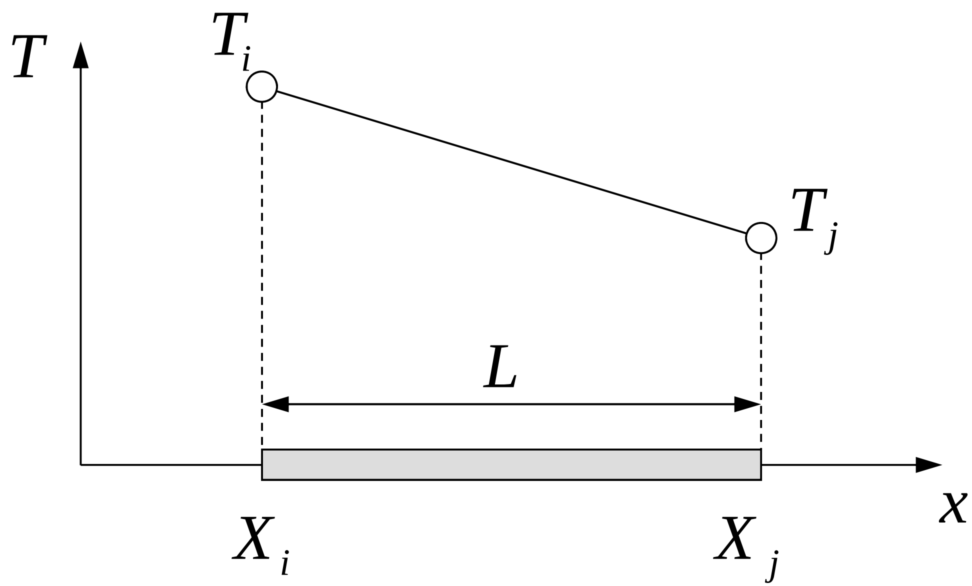
1D Linear Element (1/4)
一次元線形要素

- 1D Linear Element
  - Length= L
    - Node (Vertex)
    - Element

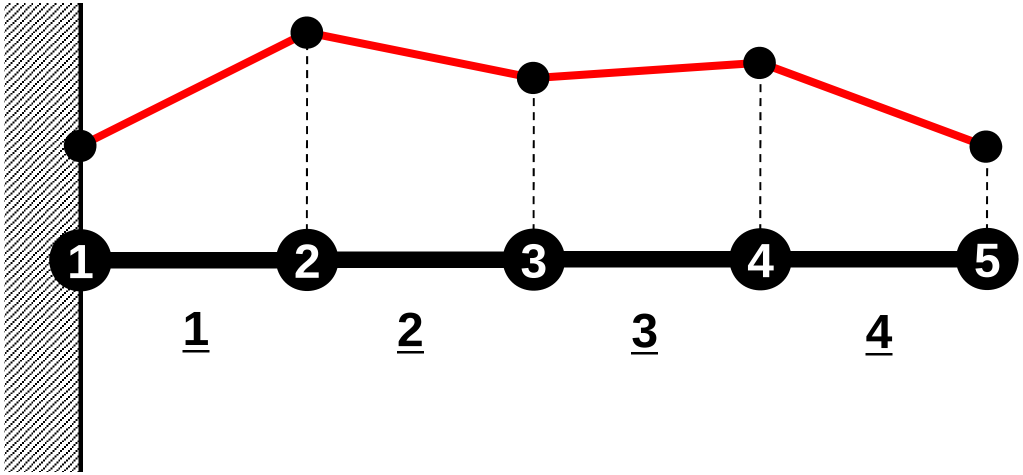


  - T_i Temperature at i
  - T_j Temperature at j
  - Temperature T on each element is linear function of x (Piecewise Linear):

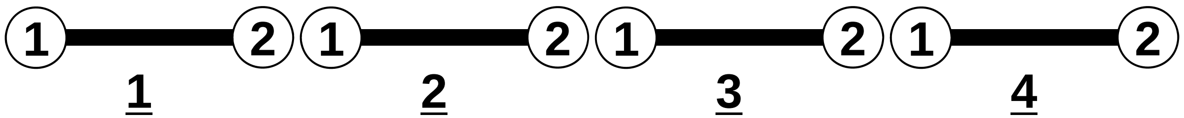
$$T = \alpha_1 + \alpha_2 x$$



Piecewise Linear



Global Node ID
Element ID

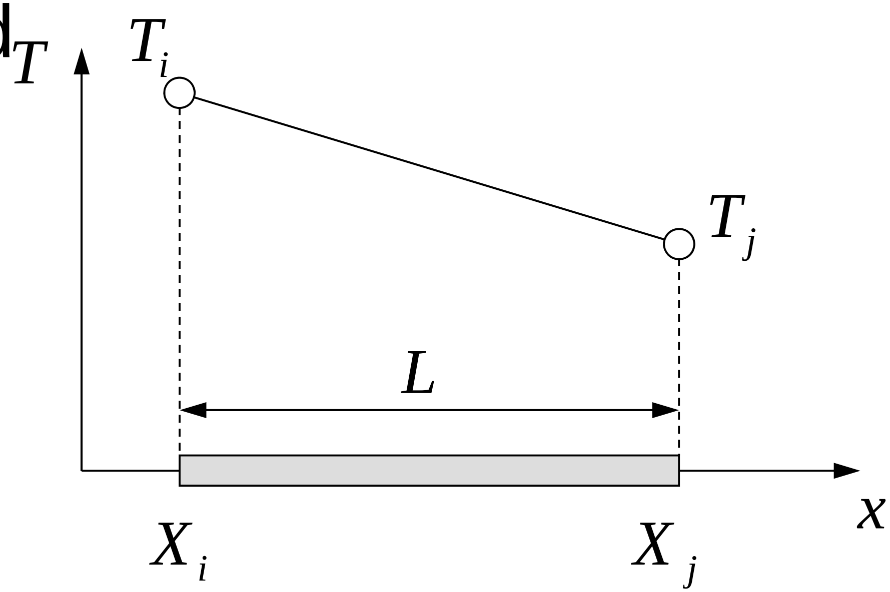


Local Node ID
for each elem.

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:

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

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

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

N_i, N_j

Shape Function or

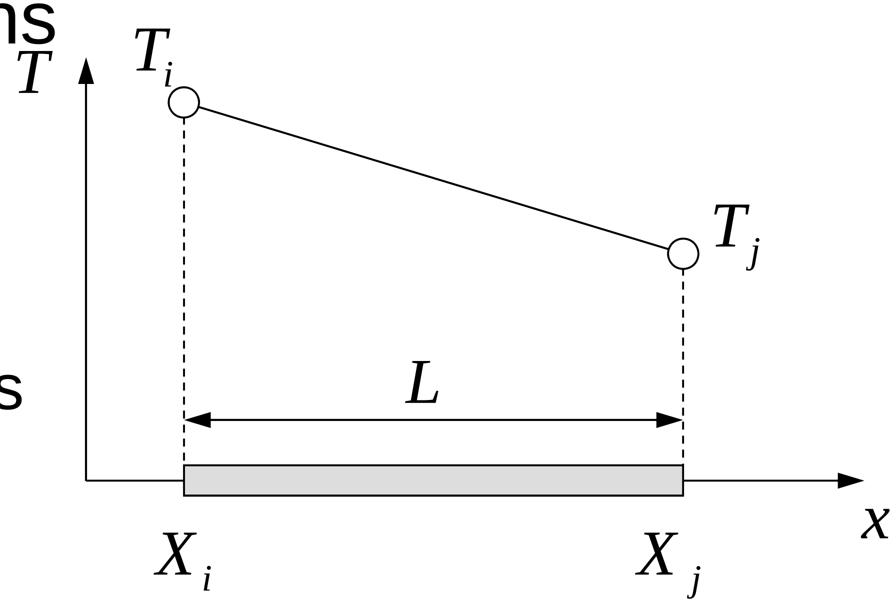
Interpolation Function

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 displacement "in" each element
  - Coef's (unknows): Temperature at each node

$$T = N_i T_i + N_j T_j$$

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

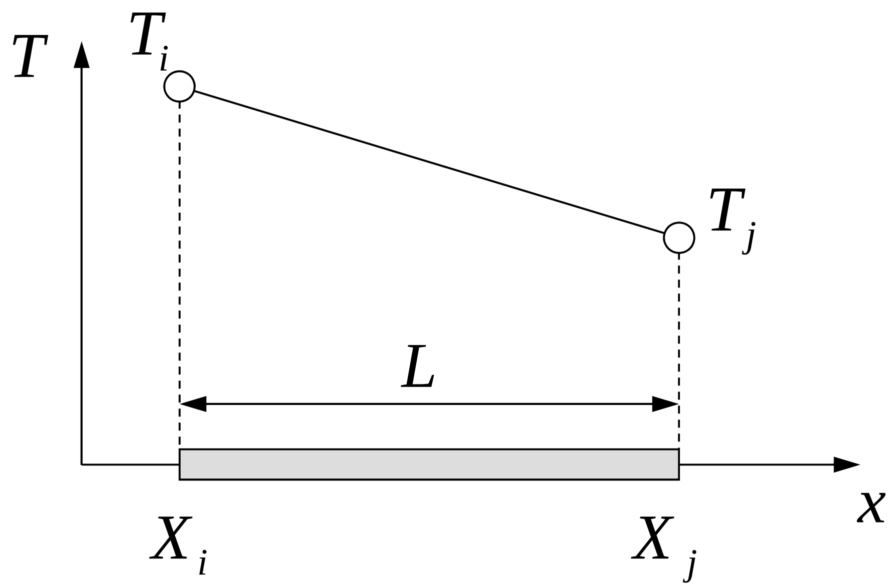
Ψ_i Trial/Test Function (known function of position, defined in domain and at boundary. "Basis" in linear algebra.

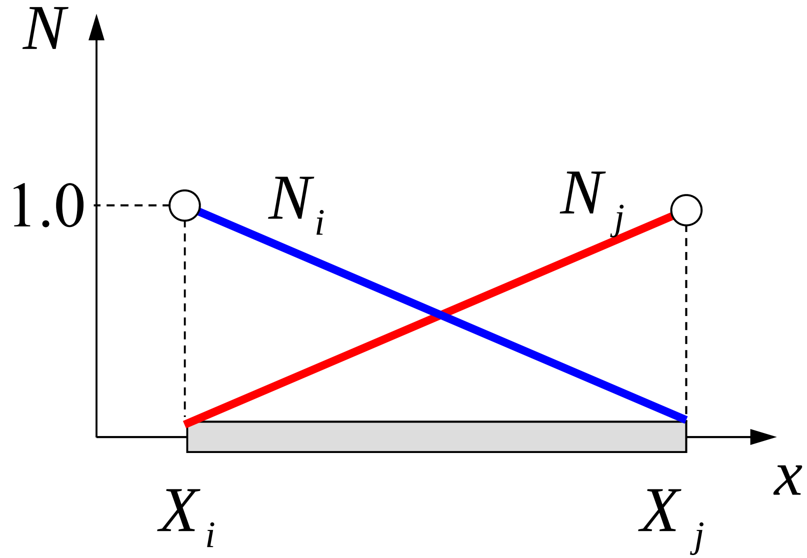
a_i Coefficients (unknown)

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

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

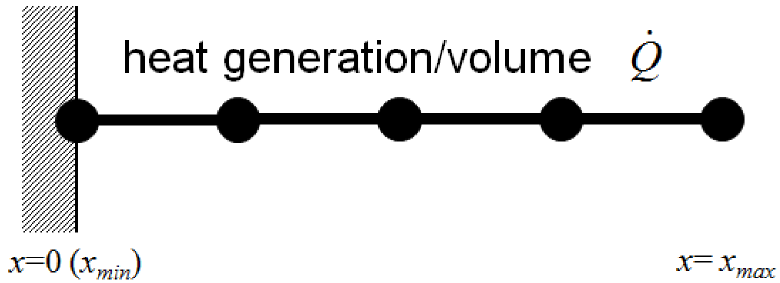
$$N_i = \left(\frac{X_j - x}{L} \right), \quad N_j = \left(\frac{x - X_i}{L} \right)$$





Galerkin Method (1/4)

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



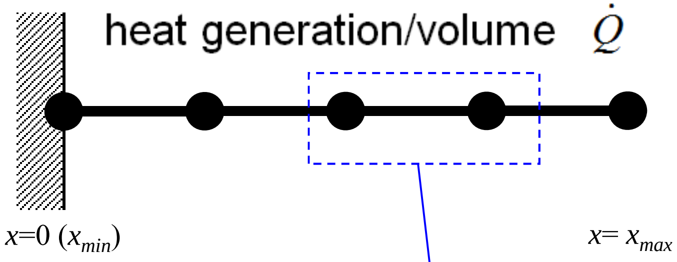
$$\lambda\left(\frac{d^2T}{dx^2}\right)+\dot{Q}=0$$

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

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

Galerkin Method (2/4)



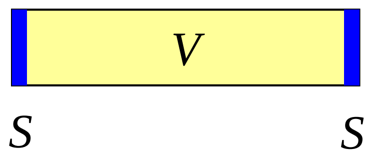
- Green's Theorem (1D)

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

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

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

- Consider the following terms:

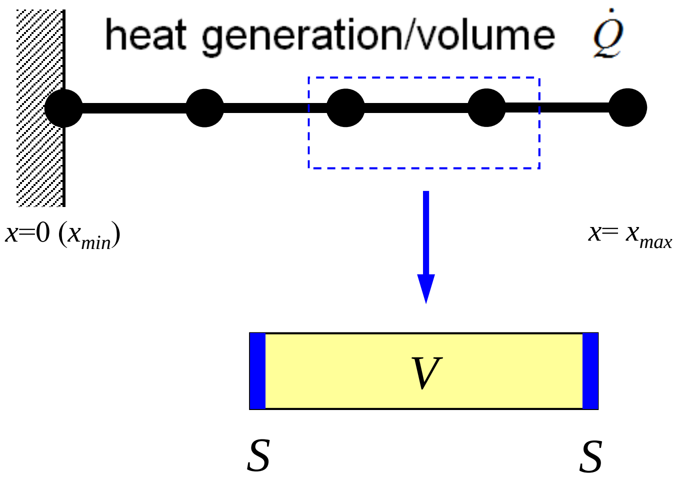


$$T = [N]\{\phi\}, \quad \frac{dT}{dx} = \frac{d[N]}{dx}\{\phi\} \qquad \bar{q} = -\lambda\frac{dT}{dx}$$

: Heat flux at element surface[$QL^{-2}T^{-1}$]

Galerkin Method (3/4)

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



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

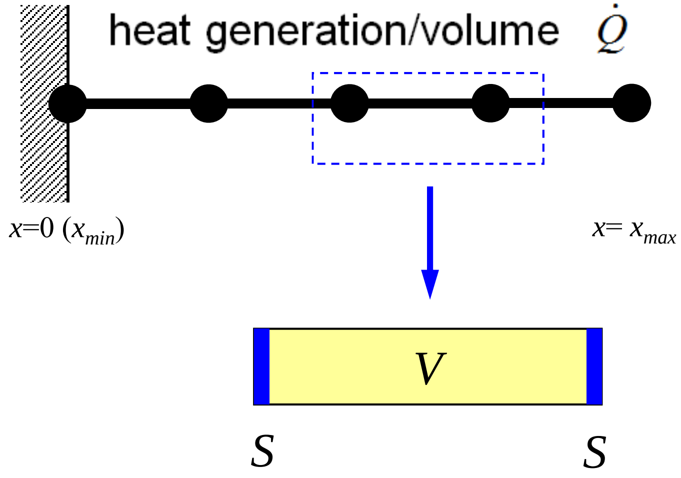
$$-\int_S \bar{q}[N]^T dS + \int_V Q[N]^T dV = 0$$

- 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)

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

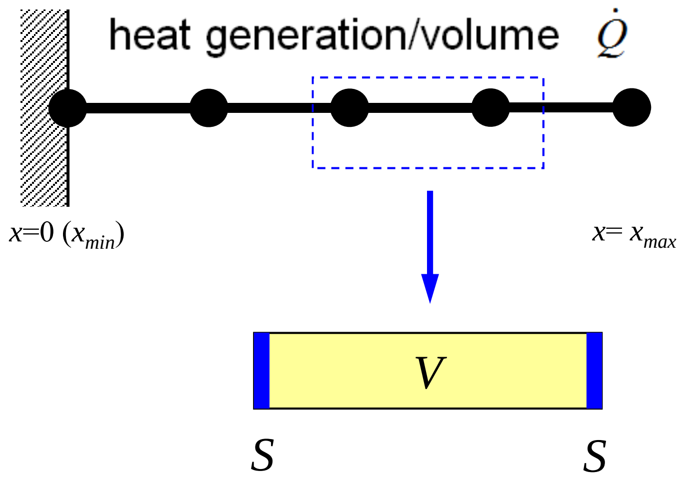
$$\boxed{-\int_S \bar{q}[N]^T dS} + \int_V \dot{Q}[N]^T dV = 0$$



- 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）

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

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

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

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

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

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

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

Integration over Each Element: $[k]$

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

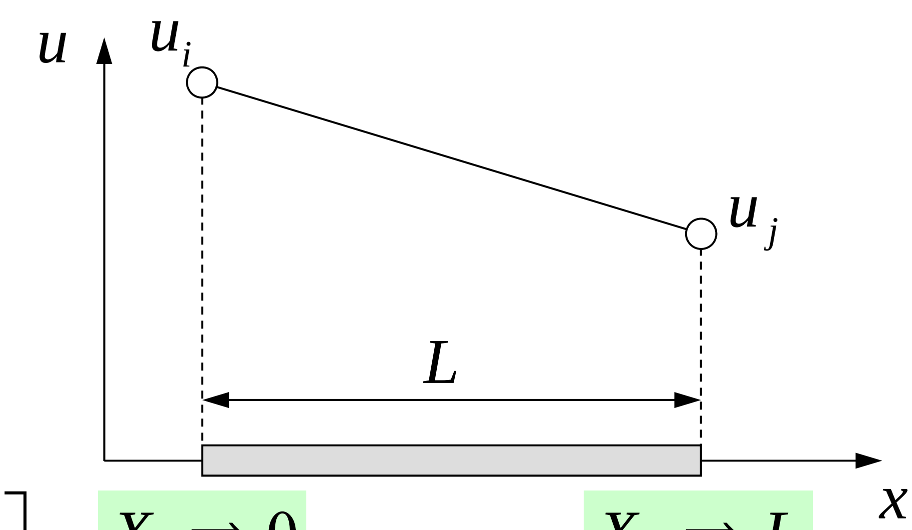
$$\frac{dN_i}{dx} = \left(\frac{-1}{L}\right), \quad \frac{dN_j}{dx} = \left(\frac{1}{L}\right)$$

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

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

2x1 matrix　　1x2 matrix

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



$X_i \Rightarrow 0$　　$X_j \Rightarrow L$

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

A: Sectional Area

L: Length

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

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

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

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

Heat Generation (Volume)

1 : 1

A: Sectional Area

L: Length

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

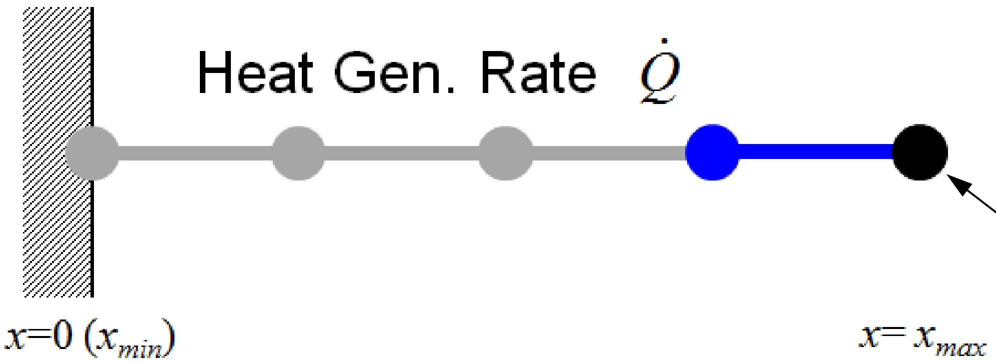
$$N_i = \left(\frac{X_j - x}{L} \right), \quad N_j = \left(\frac{x - X_i}{L} \right) \qquad \frac{dN_i}{dx} = \left(\frac{-1}{L} \right), \quad \frac{dN_j}{dx} = \left(\frac{1}{L} \right)$$

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

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



when surface heat flux acts on only this surface.

Global Equations

- Accumulate Element Equations:

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

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

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

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

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

ECCS2012 System

Creating Directory

```
>$ cd Documents
>$ mkdir 2013summer   your favorite name
>$ cd 2013summer
```

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

1D Code for Steady-State Heat Conduction Problems

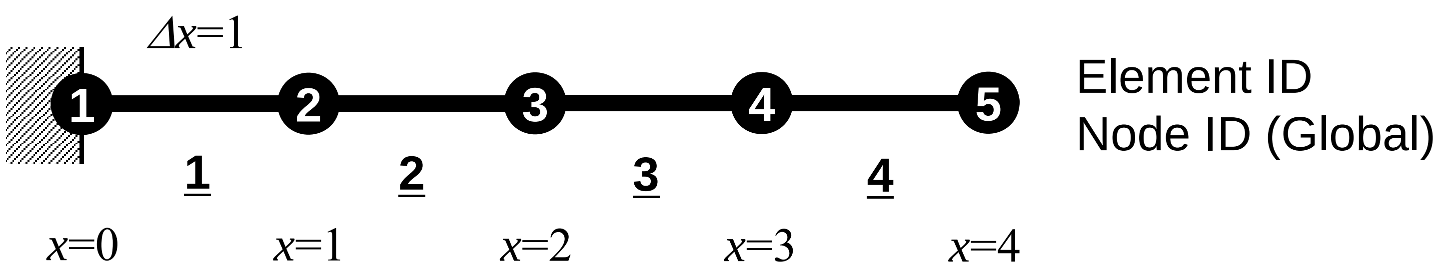
```
>$ cd <$E-TOP>
>$ cp /home03/skengon/Documents/class_eps/F/1d.tar .
>$ cp /home03/skengon/Documents/class_eps/C/1d.tar .
>$ tar xvf 1d.tar
>$ cd 1d
```

Compile & GO !

```
>$ cd <$E-TOP>/1d
>$ cc –O 1d.c       (or g95 –O 1d.f)
>$ ./a.out
```

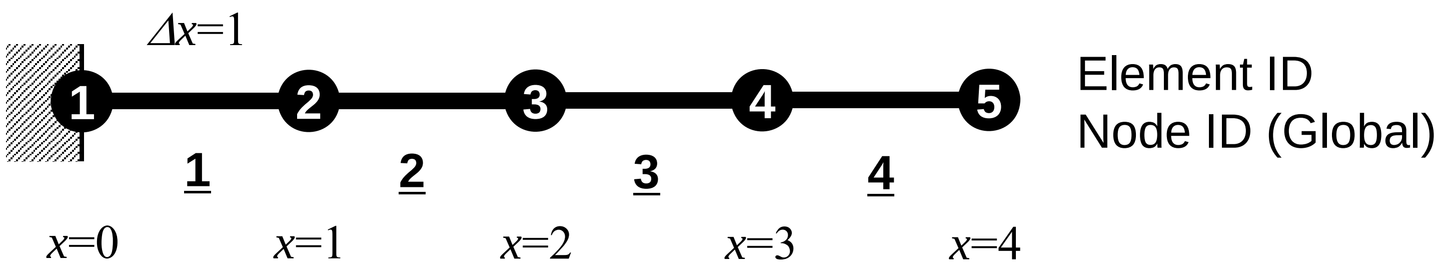
Control Data **`input.dat`**

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



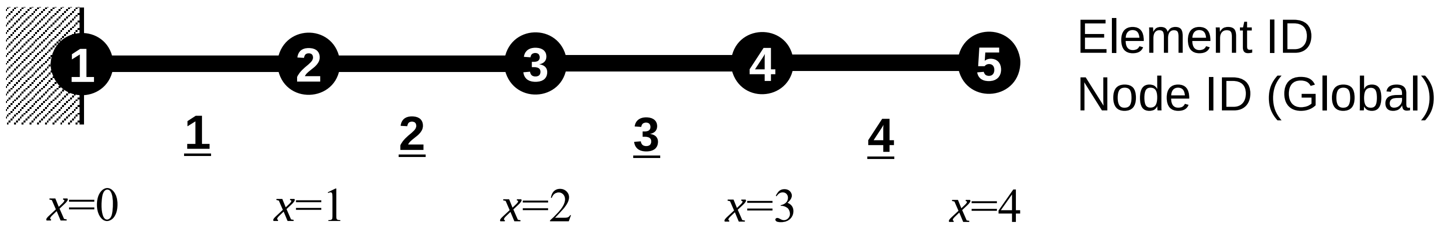
Results

```
>$ ./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 Eqn's/Accumulation (1/3)

- 4 elements, 5 nodes



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

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

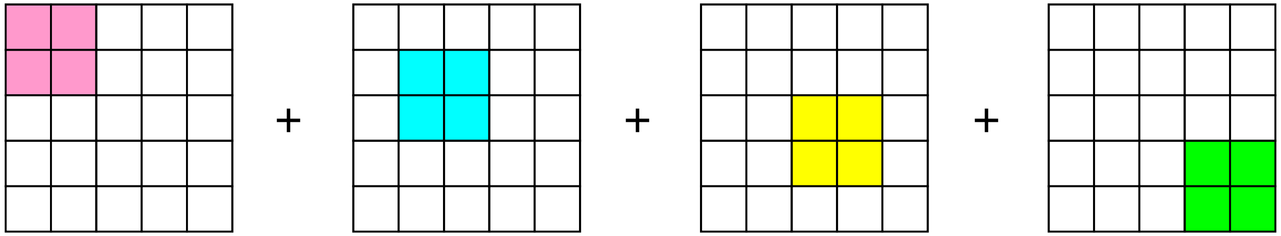
- As for Element-4:

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

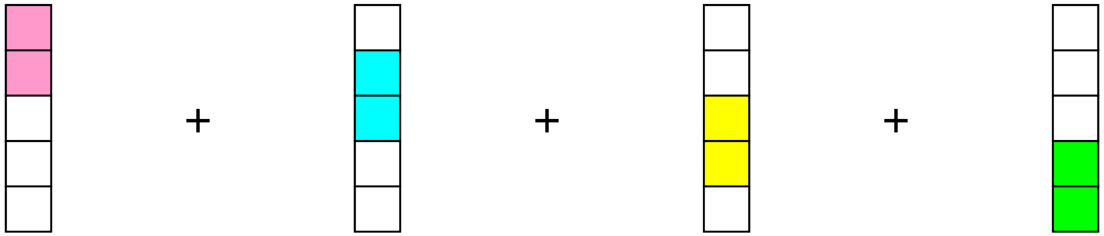
Element Eqn's/Accumulation (2/3)

- Element-by-Element Accumulation:

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



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



Element Eqn's/Accumulation (3/3)

- Relations to FDM

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

$$[K] = \sum_{e=1}^{4}[k]^{(e)} = \left[\begin{array}{|c|c|c|c|c|} \hline +1 & -1 & & & \\ \hline -1 & +1 & & & \\ \hline & & & & \\ \hline & & & & \\ \hline & & & & \\ \hline \end{array} + \begin{array}{|c|c|c|c|c|} \hline & & & & \\ \hline & +1 & -1 & & \\ \hline & -1 & +1 & & \\ \hline & & & & \\ \hline & & & & \\ \hline \end{array} + \begin{array}{|c|c|c|c|c|} \hline & & & & \\ \hline & & & & \\ \hline & & +1 & -1 & \\ \hline & & -1 & +1 & \\ \hline & & & & \\ \hline \end{array} + \begin{array}{|c|c|c|c|c|} \hline & & & & \\ \hline & & & & \\ \hline & & & & \\ \hline & & & +1 & -1 \\ \hline & & & -1 & +1 \\ \hline \end{array} \right] \times \frac{\lambda A}{L}$$

$$= \begin{array}{|c|c|c|c|c|} \hline +1 & -1 & & & \\ \hline -1 & +2 & -1 & & \\ \hline & -1 & +2 & -1 & \\ \hline & & -1 & +2 & -1 \\ \hline & & & -1 & +1 \\ \hline \end{array} \times \frac{\lambda A}{L}$$

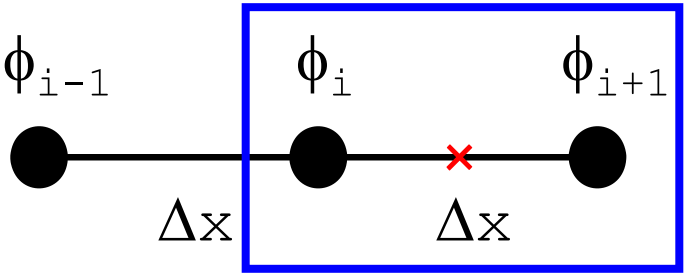
$$-\int_V \left(\frac{d^2T}{dx^2}\right) dV = -\int_V \left(\frac{T_{i+1} - 2T_i + T_{i-1}}{L^2}\right) dV$$
$$= -\left(\frac{T_{i+1} - 2T_i + T_{i-1}}{L^2}\right) \cdot AL = -\left(T_{i+1} - 2T_i + T_{i-1}\right) \cdot \frac{A}{L}$$

Something familiar …

FEM: Coefficient Matrices are generally sparse (many ZERO's)

2nd –Order Differentiation in FDM

- Approximate Derivative at × (center of *i* and *i+1*)



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

$\Delta x \to 0$: 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

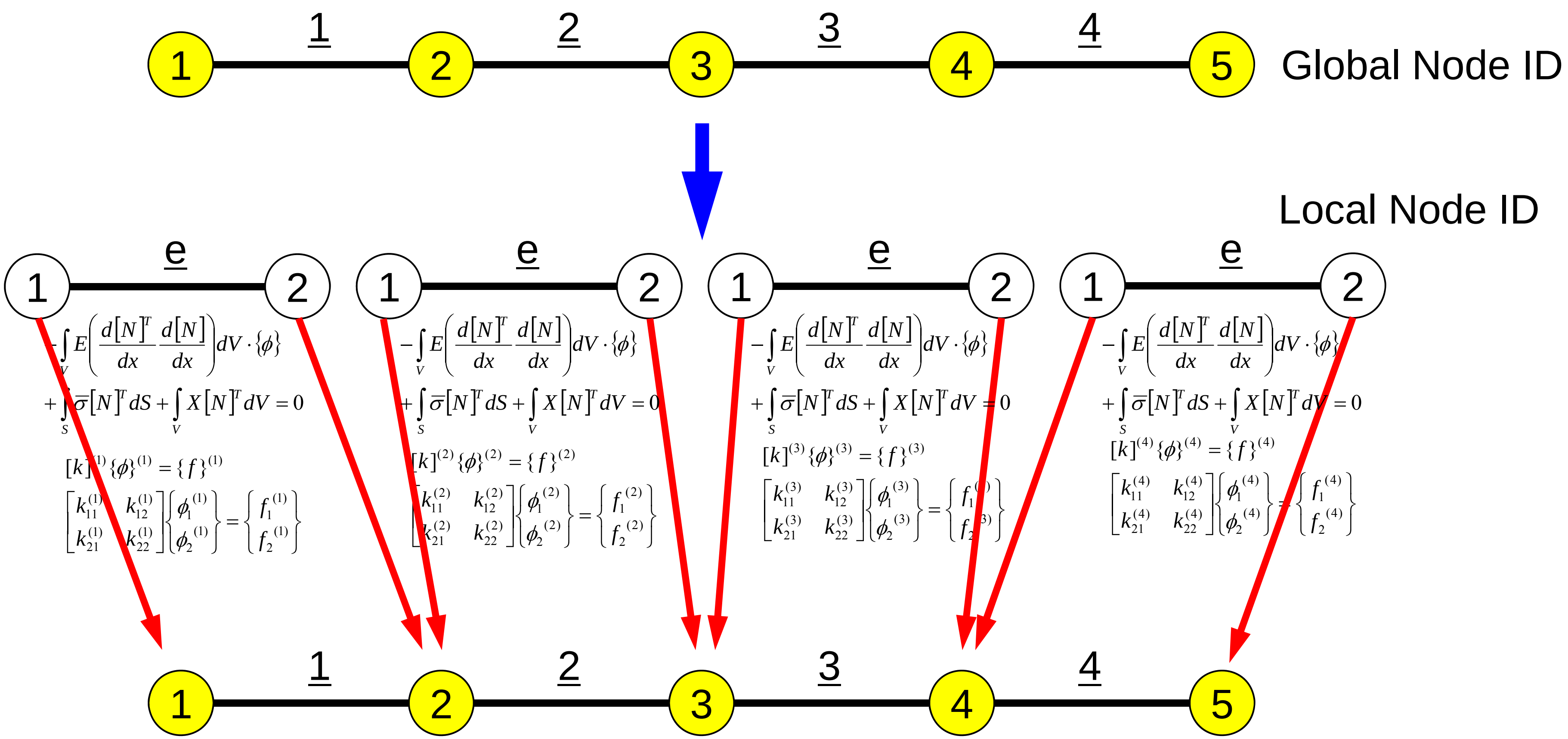
very flexible if each element has different material property, size, etc.

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

$$
\left[K\right] = \sum_{e=1}^{4} \left[k^{(e)}\right] = \begin{bmatrix} +1 & -1 & & & \\ -1 & +1 & & & \\ & & & & \\ & & & & \\ & & & & \end{bmatrix} \times \frac{\lambda^{(1)} A^{(1)}}{L^{(1)}} + \begin{bmatrix} & & & & \\ & +1 & -1 & & \\ & -1 & +1 & & \\ & & & & \\ & & & & \end{bmatrix} \times \frac{\lambda^{(2)} A^{(2)}}{L^{(2)}}
$$

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

Element/Global Operations

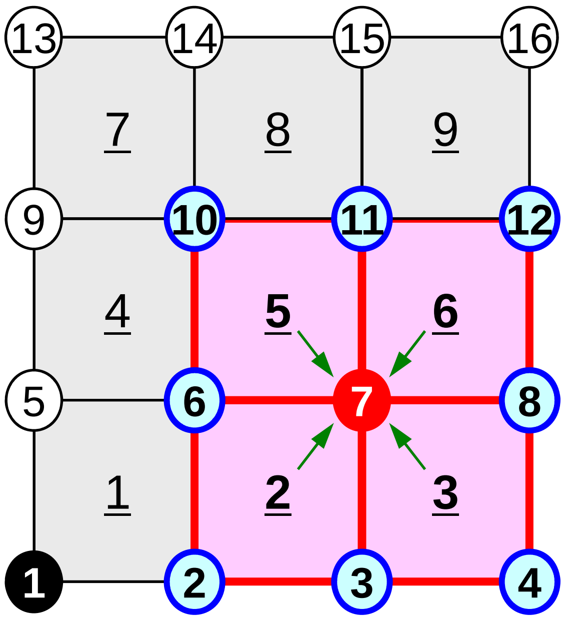


$$[K]\{\Phi\} = \{F\}$$

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

Mapping Information needed,
from element-matrix
to global-matrix.

Accumulation to Global/overall Matrix



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

$$
\begin{bmatrix}
D & X & & & X & X & & & & & & & & & & \\
X & D & X & & X & X & X & & & & & & & & & \\
& X & D & X & & X & X & X & & & & & & & & \\
& & X & D & & & X & X & & & & & & & & \\
X & X & & & D & X & & & X & X & & & & & & \\
X & X & X & & X & D & X & & X & X & X & & & & & \\
& X & X & X & & X & D & X & & X & X & X & & & & \\
& & X & X & & & X & D & & & X & X & & & & \\
& & & & X & X & & & D & X & & & X & X & & \\
& & & & X & X & X & & X & D & X & & X & X & X & \\
& & & & & X & X & X & & X & D & X & & X & X & X \\
& & & & & & X & X & & & X & D & & & X & X \\
& & & & & & & & X & X & & & D & X & & \\
& & & & & & & & X & X & X & & X & D & X & \\
& & & & & & & & & X & X & X & & X & D & X \\
& & & & & & & & & & X & X & & & X & D
\end{bmatrix}
\begin{Bmatrix}
\Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \\ \Phi_5 \\ \Phi_6 \\ \Phi_7 \\ \Phi_8 \\ \Phi_9 \\ \Phi_{10} \\ \Phi_{11} \\ \Phi_{12} \\ \Phi_{13} \\ \Phi_{14} \\ \Phi_{15} \\ \Phi_{16}
\end{Bmatrix}
=
\begin{Bmatrix}
F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \\ F_8 \\ F_9 \\ F_{10} \\ F_{11} \\ F_{12} \\ F_{13} \\ F_{14} \\ F_{15} \\ F_{16}
\end{Bmatrix}
$$

- 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 **Ax=b** is the most important and <u>**expensive**</u> 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 <u>***direct***</u> and <u>***iterative***</u> 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

Iterative Method
反復法

- Stationary Method
  - SOR, Gauss-Seidel, Jacobi
  - Generally slow, impractical

- 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**

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
  - $\{x\}^{\mathrm{T}}[A]\{x\}>0$ for arbitrary $\{x\}$
  - All of diagonal components, eigenvaules 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

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

- Algorithm
  - “Steepest Descent Method”
  - $\boldsymbol{x}^{(i)} = \boldsymbol{x}^{(i-1)} + \boldsymbol{\alpha}_i \boldsymbol{p}^{(i)}$
    - $x^{(i)}$: solution, $p^{(i)}$: search direction, α_i: coefficient
  - Solution $\{x\}$ minimizes $\{\boldsymbol{x}-\boldsymbol{y}\}^{\mathrm{T}}[\boldsymbol{A}]\{\boldsymbol{x}-\boldsymbol{y}\}$, where $\{y\}$ is exact solution.

Procedures of Conjugate Gradient

```
Compute r(0)= b-[A]x(0)
for i= 1, 2, …
    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) + αip(i)
    r(i)= r(i-1) - αiq(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, …
    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) + αip(i)
    r(i)= r(i-1) - αiq(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, …
    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) + αip(i)
    r(i)= r(i-1) - αiq(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, …
    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) + αip(i)
    r(i)= r(i-1) - αiq(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, …
    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) + αip(i)
    r(i)= r(i-1) - αiq(i)
    check convergence |r|
end
```

$x^{(i)}$: Vector

α_i : Scalar

Algorithm of CG Method (1/5)

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

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

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

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

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

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

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

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

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

Algorithm of CG Method (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:

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

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

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

Algorithm of CG Method (3/5)

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

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

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

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

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

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

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

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

Algorithm of CG Method (4/5)

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

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

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

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

Thus, following relation is obtained:

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

Algorithm of CG Method (5/5)

$$\left(p^{(k+1)}, Ap^{(k)}\right) = \left(r^{(k+1)} + \beta_k p^{(k)}, Ap^{(k)}\right) = \left(r^{(k+1)}, Ap^{(k)}\right) + \beta_k\left(p^{(k)}, Ap^{(k)}\right) = 0$$

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

$$\left(p^{(k+1)}, Ap^{(k)}\right) = 0$$ $p^{(k)}$ is “conjugate” for matrix A

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

$$\left(p^{(i)}, Ap^{(j)}\right) = 0 \; (i \neq j)$$

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

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

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 (1/2)

$$\left(p^{(i)}, r^{(k+1)}\right) = 0, \quad i = 0,1,\ldots,k$$

$$x^{(k+1)} = x^{(i+1)} + \sum_{j=i+1}^{k} \alpha_j p^{(j)}$$

$$r^{(k+1)} = b - Ax^{(k+1)} = b - A\left[x^{(i+1)} + \sum_{j=i+1}^{k} \alpha_j p^{(j)}\right]$$

$$= \left[b - Ax^{(i+1)}\right] - \sum_{j=i+1}^{k} \alpha_j Ap^{(j)} = r^{(i+1)} - \sum_{j=i+1}^{k} \alpha_j Ap^{(j)}$$

$$\left(p^{(i)}, r^{(k+1)}\right) = \left(p^{(i)}, r^{(i+1)} - \sum_{j=i+1}^{k} \alpha_j Ap^{(j)}\right)$$

$$= \underbrace{\left(p^{(i)}, r^{(i+1)}\right)}_{=0} - \underbrace{\left(p^{(i)}, \sum_{j=i+1}^{k} \alpha_j Ap^{(j)}\right)}_{=0} = 0$$

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

$$\left(Ap^{(k)}, y - x^{(k+1)}\right)$$
$$= \left(p^{(k)}, Ay - Ax^{(k+1)}\right)$$
$$= \left(p^{(k)}, b - Ax^{(k+1)}\right)$$
$$= \left(p^{(k)}, r^{(k+1)}\right) = 0$$

Proof (2/2)

$$\left(r^{(i)}, r^{(j)}\right) = 0 \; \left(i \neq j\right)$$

$$
\begin{aligned}
0 &= \left(p^{(i)}, r^{(k+1)}\right) = \left(r^{(i)} + \beta_{i-1} p^{(i-1)}, r^{(k+1)}\right) \\
&= \left(\beta_{i-1} p^{(i-1)}, r^{(k+1)}\right) + \left(r^{(i)}, r^{(k+1)}\right) = \left(r^{(i)}, r^{(k+1)}\right)
\end{aligned}
$$

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

$$
\begin{aligned}
\left(p^{(k)}, r^{(k)}\right) &= \left(r^{(k)} + \beta_{k-1} p^{(k-1)}, r^{(k)}\right) \\
&= \left(\beta_{k-1} p^{(k-1)}, r^{(k)}\right) + \left(r^{(k)}, r^{(k)}\right) = \left(r^{(k)}, r^{(k)}\right)
\end{aligned}
$$

$\alpha_k, \ \beta_k$

Usually, we use simpler definitions of $\alpha_k, \ \beta_k$ 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, …
    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) + αip(i)
    r(i)= r(i-1) - αiq(i)
    check convergence |r|
end
```

$x^{(i)}$: Vector

α_i : Scalar

Preconditioning for Iterative Solvers

- Convergence rate of iterative solvers strongly depends on the spectral properties (eigenvalue distribution) of the coefficient matrix **A**.
  - Eigenvalue distribution is small, eigenvalues are close to 1
  - In "ill-conditioned" problems, "condition number" (ratio of max/min eigenvalue if **A** is symmetric) is large.
- A preconditioner **M** (whose properties are similar to those of **A**)transforms the linear system into one with more favorable spectral properties
  - In "ill-conditioned" problems, "condition number" (ratio of max/min eigenvalue if **A** is symmetric) is large.
  - **M** transforms original equation **Ax=b** into **A'x=b'** where $\mathbf{A'=M^{-1}A}$, $\mathbf{b'=M^{-1}b}$
  - If **M~A**, $\mathbf{M^{-1}A}$ is close to identity matrix.
  - If $\mathbf{M^{-1}=A^{-1}}$, this is the best preconditioner (a.k.a. Gaussian Elimination)

Preconditioned 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) + αip(i)
    r(i)= r(i-1) - αiq(i)
    check convergence |r|
end
```

ILU(0), IC(0)

- Widely used Preconditioners for Sparse Matrices
  - Incomplete LU Factorization
  - Incomplete Cholesky Factorization (for Symmetric Matrices)

- 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]z(i-1)= r(i-1)`** is very easy.
- Provides fast convergence for simple problems.
- 1d.f, 1d.c

- More detailed discussions on preconditioning will be provided in “Multicore Programming”.

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

Coef. Matrix derived from FEM

- Sparse Matrix
  - Many “0”’s
- Storing all components (e.g. A(i,j)) is not efficient for sparse matrices
  - A(i,j) is suitable for dense matrices

$$
\begin{bmatrix}
D & X & & & X & X & & & & & & & & & & \\
X & D & X & & X & X & X & & & & & & & & & \\
& X & D & X & & X & X & X & & & & & & & & \\
& & X & D & & & X & X & & & & & & & & \\
X & X & & & D & X & & & X & X & & & & & & \\
X & X & X & & X & D & X & & X & X & X & & & & & \\
& X & X & X & & X & D & X & & X & X & X & & & & \\
& & X & X & & & X & D & & & X & X & & & & \\
& & & & X & X & & & D & X & & & X & X & & \\
& & & & X & X & X & & X & D & X & & X & X & X & \\
& & & & & X & X & X & & X & D & X & & X & X & X \\
& & & & & & X & X & & & X & D & & & X & X \\
& & & & & & & & X & X & & & D & X & & \\
& & & & & & & & X & X & X & & X & D & X & \\
& & & & & & & & & X & X & X & & X & D & X \\
& & & & & & & & & & X & X & & & X & D
\end{bmatrix}
\begin{Bmatrix}
\Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \\ \Phi_5 \\ \Phi_6 \\ \Phi_7 \\ \Phi_8 \\ \Phi_9 \\ \Phi_{10} \\ \Phi_{11} \\ \Phi_{12} \\ \Phi_{13} \\ \Phi_{14} \\ \Phi_{15} \\ \Phi_{16}
\end{Bmatrix}
=
\begin{Bmatrix}
F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \\ F_8 \\ F_9 \\ F_{10} \\ F_{11} \\ F_{12} \\ F_{13} \\ F_{14} \\ F_{15} \\ F_{16}
\end{Bmatrix}
$$

- Number of non-zero off-diagonal components is O(100) in FEM
  - If number of unknowns is 10^8 :
    - A(i,j): $O(10^{16})$ words
    - Actual Non-zero Components : $O(10^{10})$ words
- Only (really) non-zero off-diag. components should be stored on memory

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
```

$$
\begin{bmatrix}
D & X & & & X & X & & & & & & & & & & \\
X & D & X & & X & X & X & & & & & & & & & \\
& X & D & X & & X & X & X & & & & & & & & \\
& & X & D & & & X & X & & & & & & & & \\
X & X & & & D & X & & & X & X & & & & & & \\
X & X & X & & X & D & X & & X & X & X & & & & & \\
& X & X & X & & X & D & X & & X & X & X & & & & \\
& & X & X & & & X & D & & & X & X & & & & \\
& & & & X & X & & & D & X & & & X & X & & \\
& & & & X & X & X & & X & D & X & & X & X & X & \\
& & & & & X & X & X & & X & D & X & & X & X & X \\
& & & & & & X & X & & & X & D & & & X & X \\
& & & & & & & & X & X & & & D & X & & \\
& & & & & & & & X & X & X & & X & D & X & \\
& & & & & & & & & X & X & X & & X & D & X \\
& & & & & & & & & & X & X & & & X & D
\end{bmatrix}
\begin{Bmatrix}
\Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \\ \Phi_5 \\ \Phi_6 \\ \Phi_7 \\ \Phi_8 \\ \Phi_9 \\ \Phi_{10} \\ \Phi_{11} \\ \Phi_{12} \\ \Phi_{13} \\ \Phi_{14} \\ \Phi_{15} \\ \Phi_{16}
\end{Bmatrix}
=
\begin{Bmatrix}
F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \\ F_8 \\ F_9 \\ F_{10} \\ F_{11} \\ F_{12} \\ F_{13} \\ F_{14} \\ F_{15} \\ F_{16}
\end{Bmatrix}
$$

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

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

```
{Y}= [A]{X}

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)

$$
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{bmatrix}
1.1 & 2.4 & 0 & 0 & 3.2 & 0 & 0 & 0 \\
4.3 & 3.6 & 0 & 2.5 & 0 & 3.7 & 0 & 9.1 \\
0 & 0 & 5.7 & 0 & 1.5 & 0 & 3.1 & 0 \\
0 & 4.1 & 0 & 9.8 & 2.5 & 2.7 & 0 & 0 \\
3.1 & 9.5 & 10.4 & 0 & 11.5 & 0 & 4.3 & 0 \\
0 & 0 & 6.5 & 0 & 0 & 12.4 & 9.5 & 0 \\
0 & 6.4 & 2.5 & 0 & 0 & 1.4 & 23.1 & 13.1 \\
0 & 9.5 & 1.3 & 9.6 & 0 & 3.1 & 0 & 51.3
\end{bmatrix}
\end{array}
$$

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

対角成分

```
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.	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

NPLU= 25
(=index(N))

index(i-1)+1th~index(i) th
Non-Zero Off-Diag. Components corresponding to *i*-th row

Compressed Row Storage (CRS)

Row	Diag.	Off-Diag.				# Non-Zero Off-Diag.	index
							index(0)= 0
1	1.1 ①	2.4 ②,1	3.2 ⑤,2			2	index(1)= 2
2	3.6 ②	4.3 ①,3	2.5 ④,4	3.7 ⑥,5	9.1 ⑧,6	4	index(2)= 6
3	5.7 ③	1.5 ⑤,7	3.1 ⑦,8			2	**index(3)= 8**
4	9.8 ④	4.1 ②,9	2.5 ⑤,10	2.7 ⑥,11		**3**	**index(4)= 11**
5	11.5 ⑤	3.1 ①,12	9.5 ②,13	10.4 ③,14	4.3 ⑦,15	4	index(5)= 15
6	12.4 ⑥	6.5 ③,16	9.5 ⑦,17			2	index(6)= 17
7	23.1 ⑦	6.4 ②,18	2.5 ③,19	1.4 ⑥,20	13.1 ⑧,21	4	index(7)= 21
8	51.3 ⑧	9.5 ②,22	1.3 ③,23	9.6 ④,24	3.1 ⑥,25	4	index(8)= 25

NPLU= 25
(=index(N))

index(i-1)+1^th^**~index(i)** ^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)

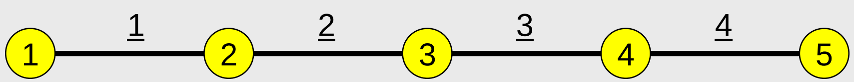
Name	Type	Size	I/O	Definition
X	R	N	O	Location of Each Node
PHI	R	N	O	Temperature of Each Node
Rhs	R	N	O	RHS Vector
Diag	R	N	O	Diagonal Components
W	R	(N,4)	O	Work Array for CG
Amat	R	NPLU	O	Off-Diagonal Components (Value)
Index	I	0:N	O	Number of Non-Zero Off-Diagonals at Each ROW
Item	I	NPLU	O	Off-Diagonal Components (Corresponding Column ID)
Icelnod	I	2*NE	O	Node ID for Each Element
Kmat	R	(2,2)	O	Element Matrix [*k*]
Emat	R	(2,2)	O	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)

      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
```

Control Data `input.dat`

4	**NE** (Number of Elements)
1.0 1.0 1.0 1.0	Δ**x** (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)

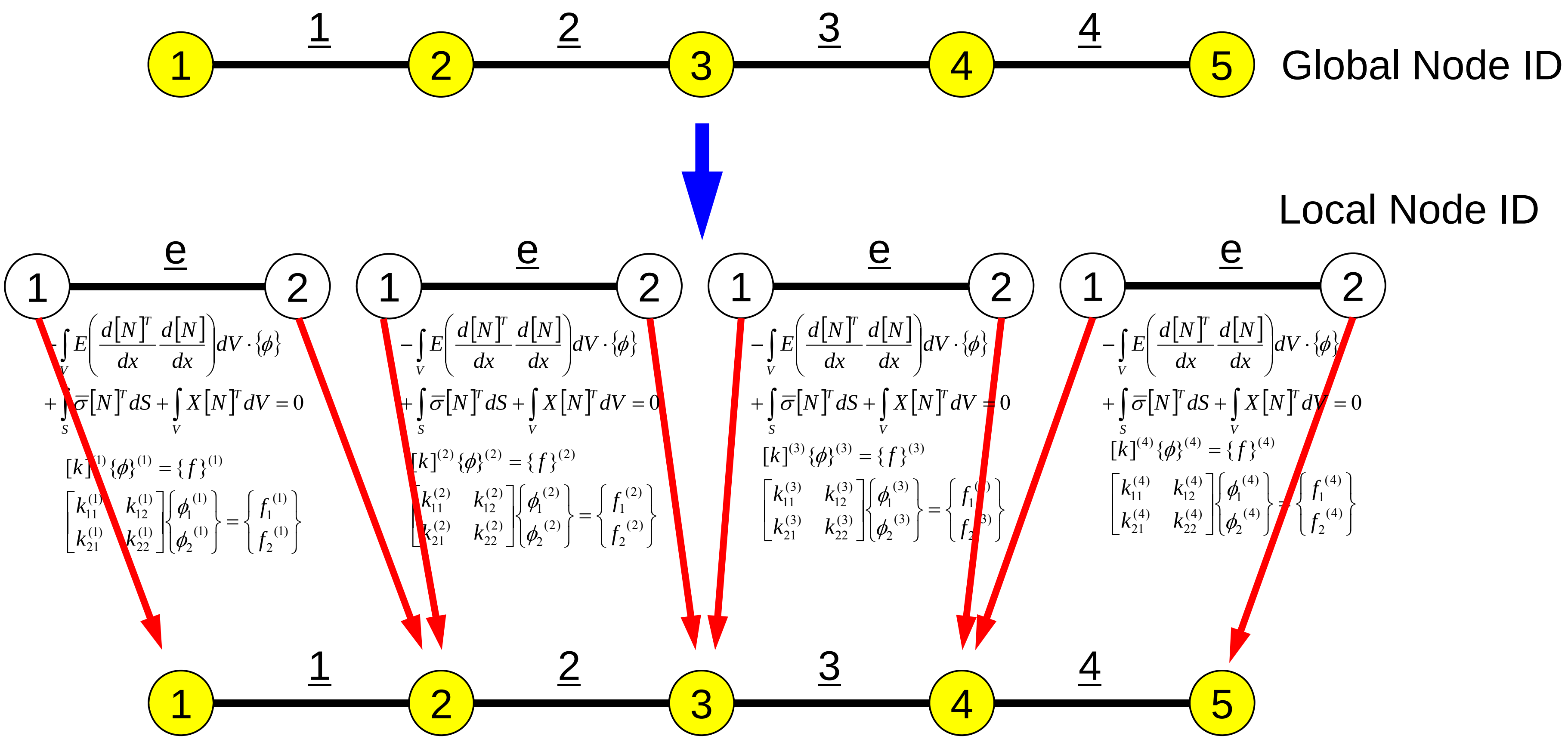
Program: 1d.f (2/6)

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

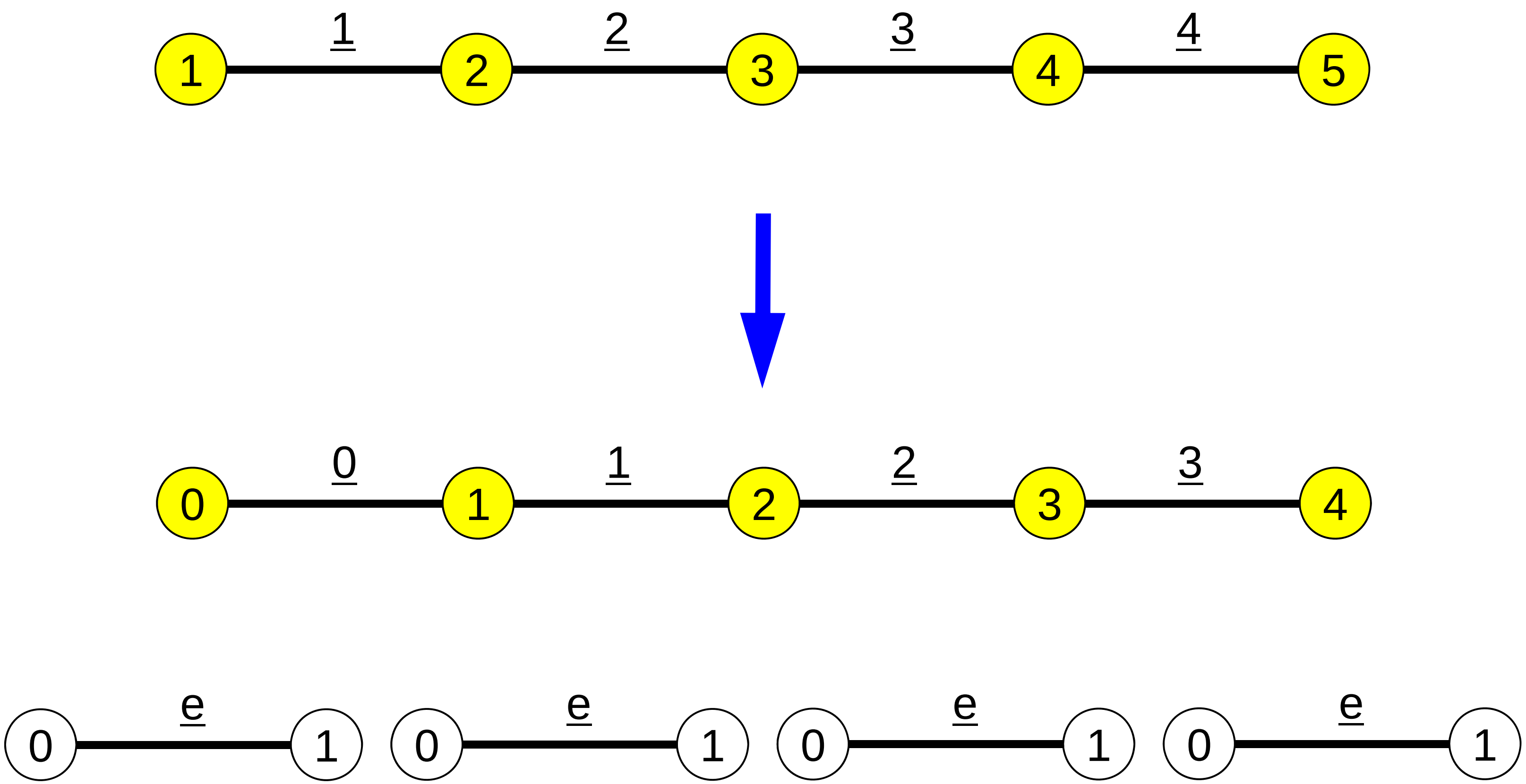


$$[K]\{\Phi\}=\{F\}$$

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

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

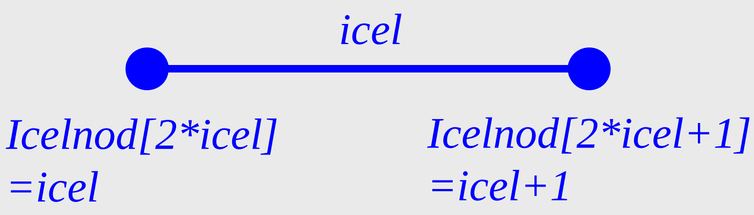
Attention: In C program, node and element ID's start from 0.



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.)

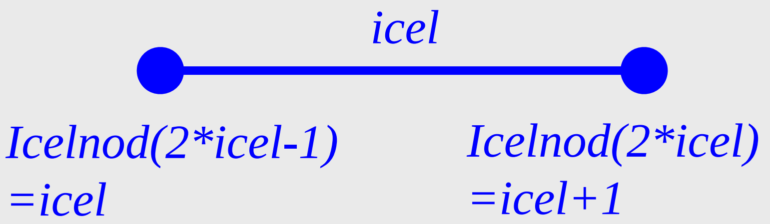
```
      do i= 1, N
        X(i)= dfloat(i-1)*dX
      enddo

      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
```

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

[Kmat]

Program: 1d.f (4/6)

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

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

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]

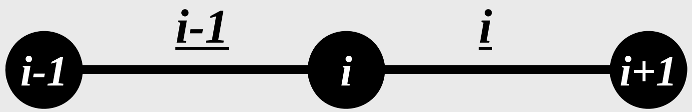
						# Non-Zero Off-Diag.	index(0)= 0
1	1.1 ①	2.4 ②	3.2 ⑤			2	index(1)= 2
2	3.6 ②	4.3 ①	2.5 ④	3.7 ⑥	9.1 ⑧	4	index(2)= 6
3	5.7 ③	1.5 ⑤	3.1 ⑦			2	index(3)= 8
4	9.8 ④	4.1 ②	2.5 ⑤	2.7 ⑥		3	index(4)= 11
5	11.5 ⑤	3.1 ①	9.5 ②	10.4 ③	4.3 ⑦	4	index(5)= 15
6	12.4 ⑥	6.5 ③	9.5 ⑦			2	index(6)= 17
7	23.1 ⑦	6.4 ②	2.5 ③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



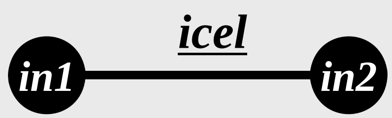
						# Non-Zero Off-Diag.	index(0)= 0
1	1.1 ①	2.4 ②	3.2 ⑤			2	index(1)= 2
2	3.6 ②	4.3 ①	2.5 ④	3.7 ⑥	9.1 ⑧	4	index(2)= 6
3	5.7 ③	1.5 ⑤	3.1 ⑦			2	index(3)= 8
4	9.8 ④	4.1 ②	2.5 ⑤	2.7 ⑥		3	index(4)= 11
5	11.5 ⑤	3.1 ①	9.5 ②	10.4 ③	4.3 ⑦	4	index(5)= 15
6	12.4 ⑥	6.5 ③	9.5 ⑦			2	index(6)= 17
7	23.1 ⑦	6.4 ②	2.5 ③	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)+1th~index(i) th
Non-Zero Off-Diag. Components corresponding to *i*-th row

Program: 1d.f (5/6)

Element Matrix ~ Global Matrix

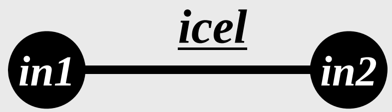
```
!C +-----------------+
!C | MATRIX ASSEMBLE |
!C +-----------------+
!C===
      do icel= 1, NE
        in1= ICELNOD(2*icel-1)
        in2= ICELNOD(2*icel  )
        X1 = X(in1)
        X2 = X(in2)
        DL = dabs(X2-X1)

        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===
```

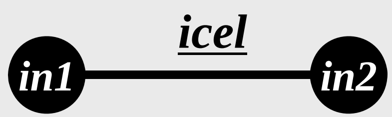


$$[Emat]=[k]^{(e)}=\frac{\lambda A}{L}\begin{bmatrix}+1 & -1\\ -1 & +1\end{bmatrix}=\frac{\lambda A}{L}[Kmat]$$

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===
```



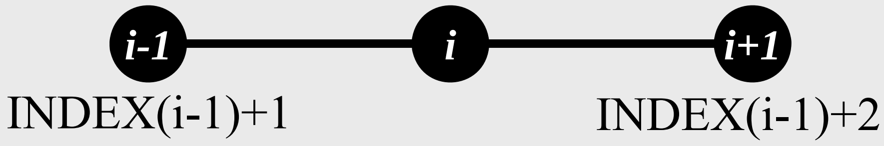
$$[Emat] = [k]^{(e)} = \frac{EA}{L}\begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}$$

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= 0.50d0*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



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

k1

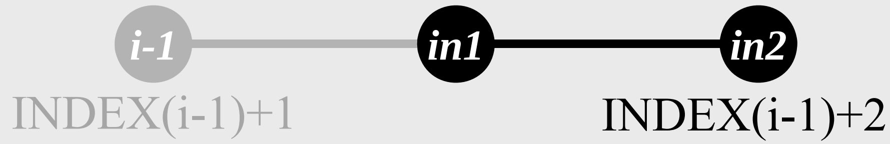
k2

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



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

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

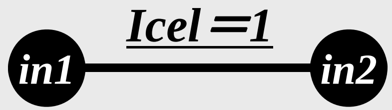
$$[Emat]=[k]^{(e)}=\frac{\lambda A}{L}\begin{bmatrix}+1 & -1\\ -1 & +1\end{bmatrix}$$

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

INDEX(i-1)+1

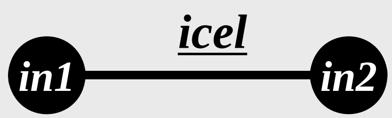
$$[Emat]=[k]^{(e)}=\frac{\lambda A}{L}\begin{bmatrix}+1 & -1\\ -1 & +1\end{bmatrix}$$

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===
```



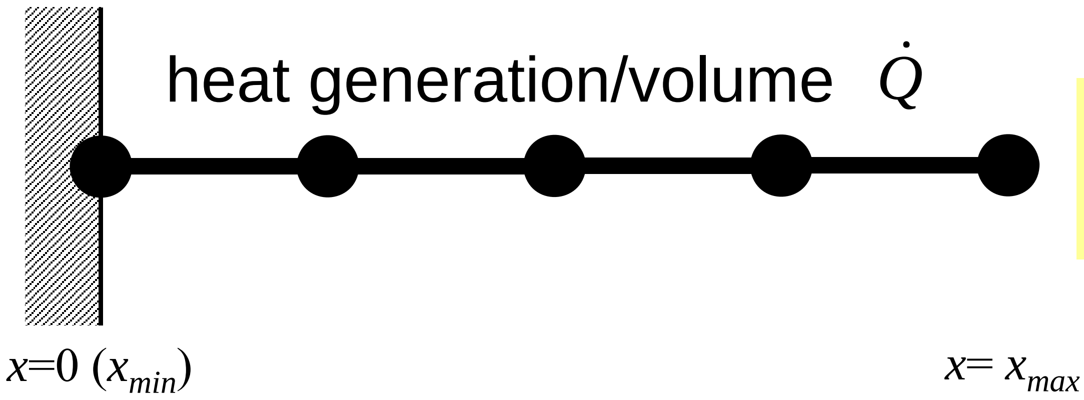
$$\int_V \dot{Q}[N]^T dV = \dot{Q}A\int_0^L \begin{bmatrix} 1-x/L \\ x/L \end{bmatrix} dx = \frac{\dot{Q}AL}{2}\begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$$

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===
```

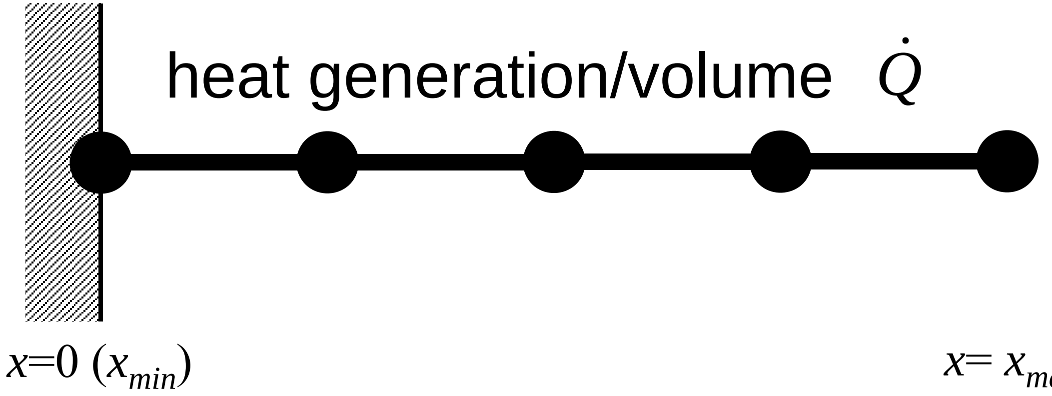
1D Steady State Heat Conduction



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

- 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)

(Linear) Equation at x=0
$T_1 = 0$ (or $T_0 = 0$)



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

- 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} : $\dfrac{\partial T}{\partial x} = 0$ (Insulated)

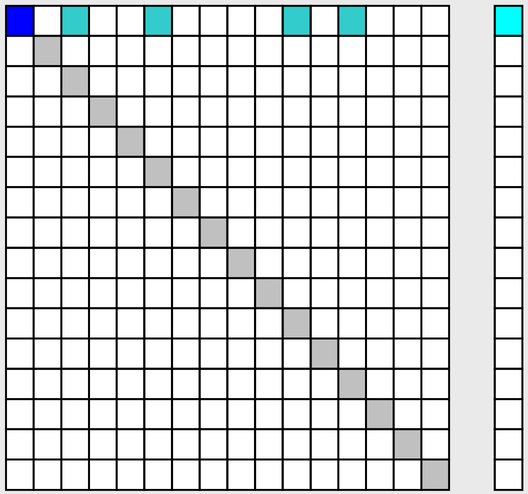
Program: 1d.f (6/6)

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.



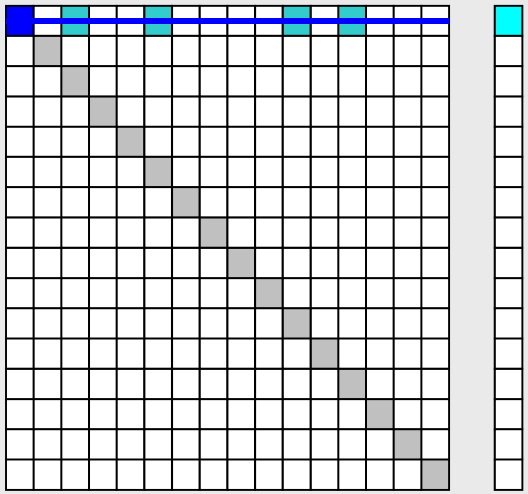
Program: 1d.f (6/6)
Dirichlet B.C. @ X=0

```
!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)

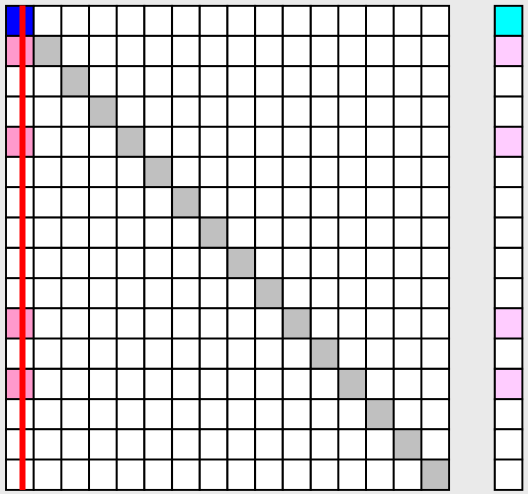
Dirichlet B.C. @ X=0

```
!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===

!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===
```

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

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

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===
```

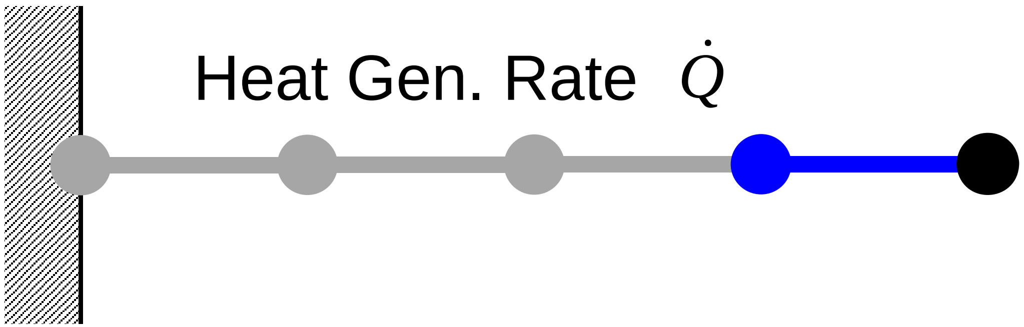
$$Diag_j \phi_j + \sum_{k=Index[j],k\neq k_s}^{Index[j+1]-1} Amat_k \, \phi_{Item[k]}$$

$$= Rhs_j - Amat_{k_s} \phi_{Item[k_s]}$$

$$= Rhs_j - Amat_{k_s} \phi_{\min} \quad where\ Item[k_s] = 0$$

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)



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

x=0 (x_{min})

x= x_{max}

$$T=0 @ x=0$$

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

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

Surface 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 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) + αip(i)
    r(i)= r(i-1) - αiq(i)
    check convergence |r|
end
```

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

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
```

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

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

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

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

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

CG Solver (4/6)

```
!C
!C-- {p} = {z} if      ITER=1
!C   BETA= RHO / RHO1  otherwise

      if ( iter.eq.1 ) then
        do i= 1, N
          W(i,P)= W(i,Z)
        enddo
       else
         BETA= RHO / RHO1
         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 $r^{(0)}= b-[A]x^{(0)}$
<u>for</u> i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  <u>if</u> i=1
    $p^{(1)}= z^{(0)}$
  <u>else</u>
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
  <u>endif</u>
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence $|r|$
<u>end</u>

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

<u>for</u> i= 1, 2, …

    solve $[M]z^{(i-1)}= r^{(i-1)}$

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

    <u>if</u> i=1

      $p^{(1)}= z^{(0)}$

    <u>else</u>

      $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$

      $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$

    <u>endif</u>

    $q^{(i)}= [A]p^{(i)}$

    $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$

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

    $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$

    check convergence |r|

<u>end</u>

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
```

ρ_{i-2}

```
    enddo
900 continue
```

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

Control Data **input.dat**

4	**NE** (Number of Elements)
1.0 1.0 1.0 1.0	Δ**x** (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 $r^{(0)}= b-[A]x^{(0)}$
<u>for</u> i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  <u>if</u> i=1
    $p^{(1)}= z^{(0)}$
  <u>else</u>
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
  <u>endif</u>
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
<u>end</u>

Finite Element Procedures

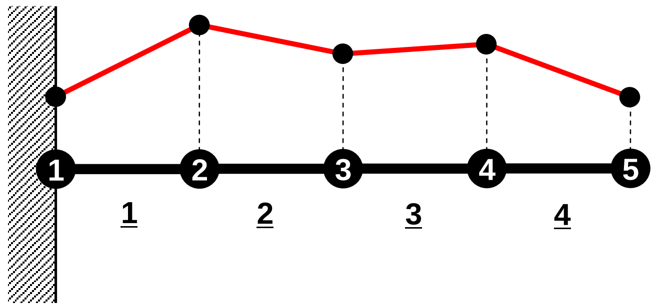
- Initialization
  - Control Data
  - Node, Connectivity of Elements (N: Node#, NE: Elem#)
  - Initialization of Arrays (Global/Element Matrices)
  - Element-Global Matrix Mapping (Index, Item)
- Generation of Matrix
  - Element-by-Element Operations (do icel= 1, NE)
    - Element matrices
    - Accumulation 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
```

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




```
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
```

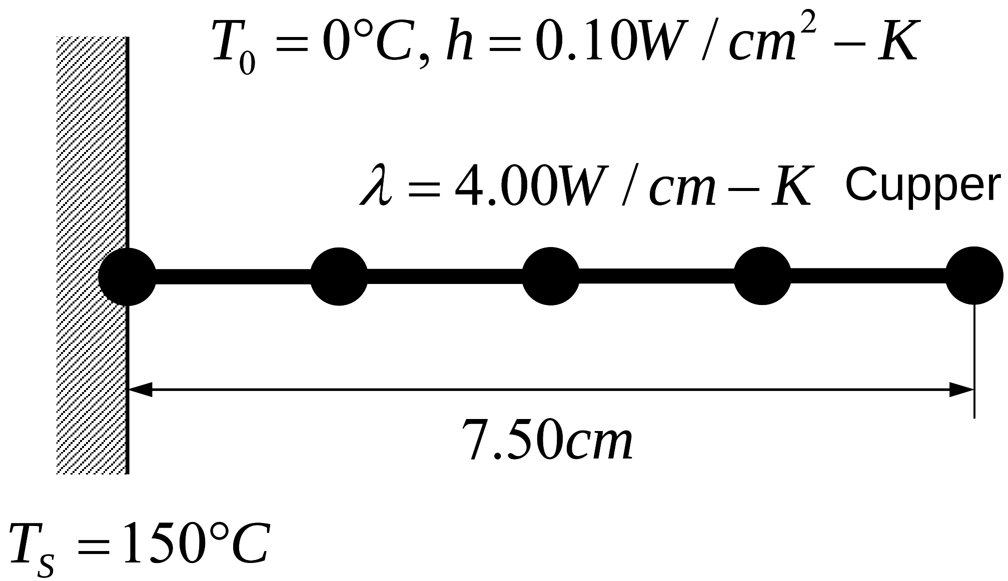
Remedies for Higher Accuracy

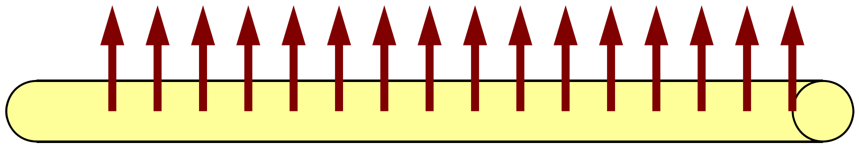
- Finer Meshes
- Higher Order Shape/Interpolation Function（高次補間関数・形状関数）
  - Higher-Order Element（高次要素）
  - Linear-Element, 1st-Order Element: Lower Order（低次要素）
- Formulation which assures continuity of n-th order derivatives
  - C^n Continuity（C^n連続性）

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=1cm

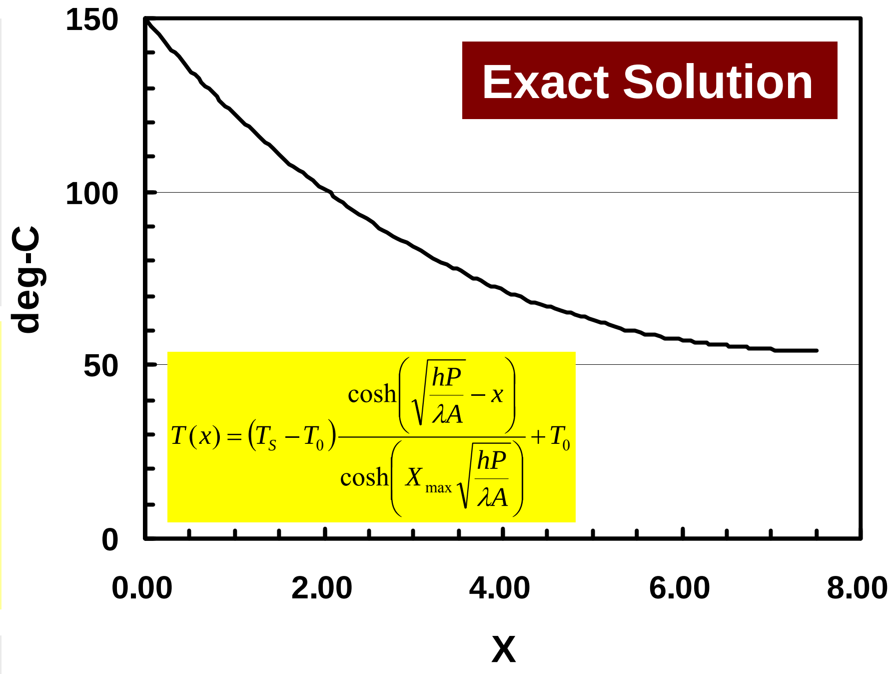
- Boundary Condition
  - x=0 : Fixed Temperature
  - x=7.5 : Insulated
- Convective Heat Transfer on Cylindrical Surface
  - $q= h (T-T_0)$
  - q : Heat Flux
    - Heat Flow/Unit Surface Area/sec.

Example: 1D Heat Transfer (2/2)

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

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

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



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