1D Code for Static Linear-Elastic Problems (1/2)

Kengo Nakajima

Technical & Scientific Computing I (4820-1027)

Seminar on Computer Science I (4810-1204)

- 1D-code for Static Linear-Elastic Problems by Galerkin FEM
- Sparse Linear Solver
  - Conjugate Gradient Method
  - Preconditioning
- Storage of Sparse Matrices
- Program
- Higher-order Elements
- Numerical Integration, Isoparametric Elements
- Report #1

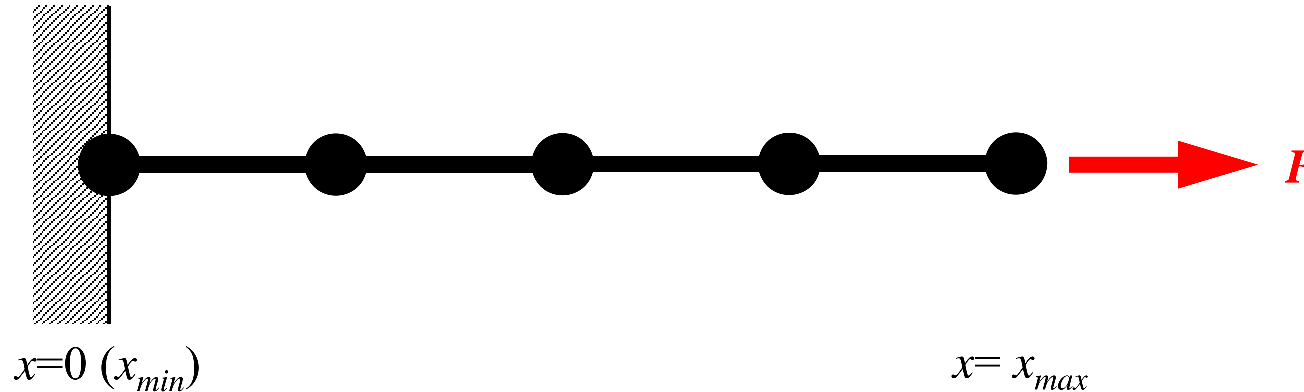
Keywords

- 1D Static Linear Elastic Problems
- Galerkin Method
- Linear Element
- Preconditioned Conjugate Gradient Method

1D Static Linear Elastic Problem（Truss：トラス）

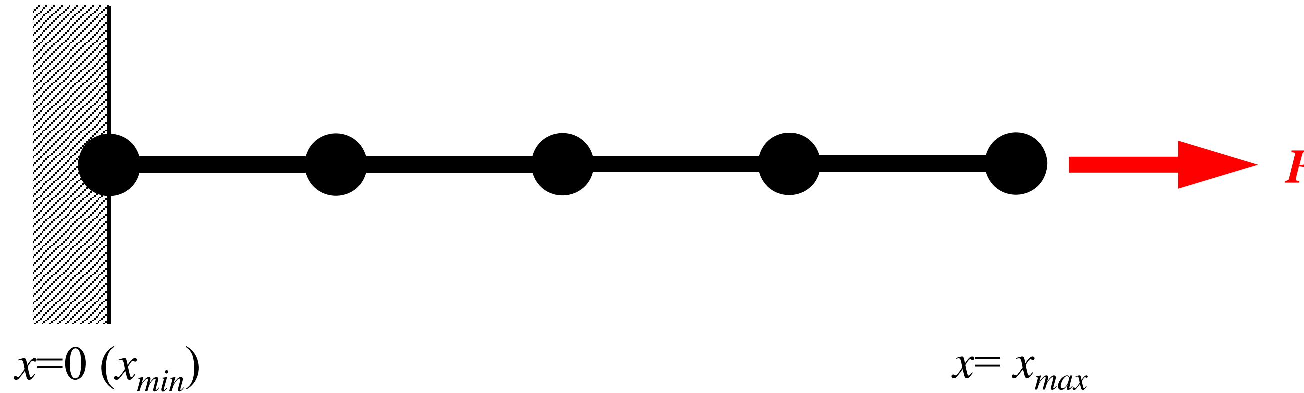
- Only deforms in x-direction (displacement: u)
  - Uniform: Sectional Area A, Young's Modulus E
  - Boundary Conditions (B.C.)
    - $x=0$: $u=0$ (fixed)
    - $x=x_{max}$: F (axial force)
- Truss: NO bending deformation by G-force

1D Static Linear Elastic Problem



- Only deforms in x-direction (displacement: u)
  - Uniform: Sectional Area A, Young's Modulus E
  - Boundary Conditions (B.C.)
    - x=0 : u=0 (fixed)
    - $x=x_{max}$: F (axial force)
- Truss: NO bending deformation by G-force

1D Static Linear Elastic Problem



Equilibrium Equation	$\frac{\partial \sigma_x}{\partial x} + X = 0$
Strain~ Displacement	$\varepsilon_x = \frac{\partial u}{\partial x}$
Stress~ Strain	$\sigma_x = E\varepsilon_x$

$$\frac{\partial}{\partial x}\left(E \frac{\partial u}{\partial x} \right) + X = 0$$

Governing Equation for u

Procedures for Computation

- solve equations for displacement u

$$\frac{\partial}{\partial x}\left(E \frac{\partial u}{\partial x} \right) + X = 0$$

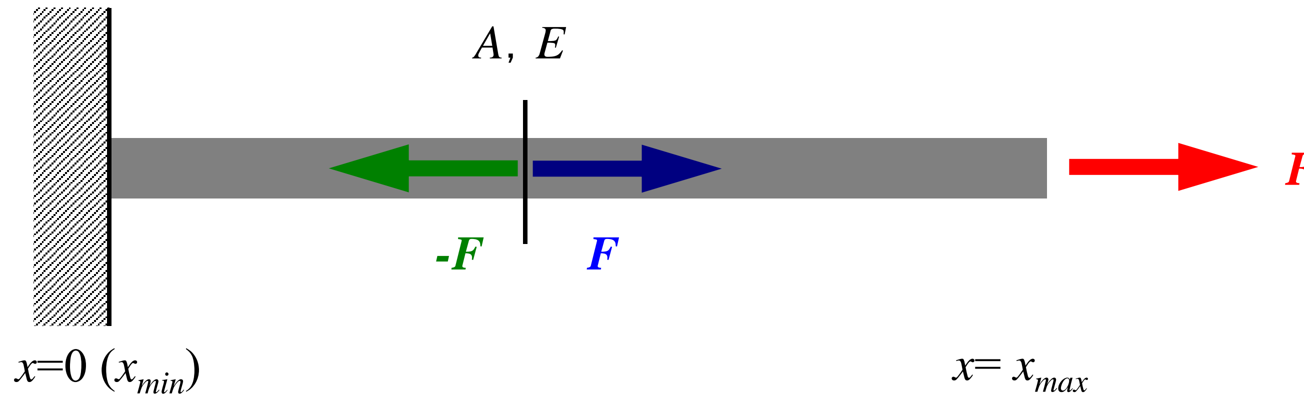
- calc. strain

$$\varepsilon_x = \frac{\partial u}{\partial x}$$

- calc. stress

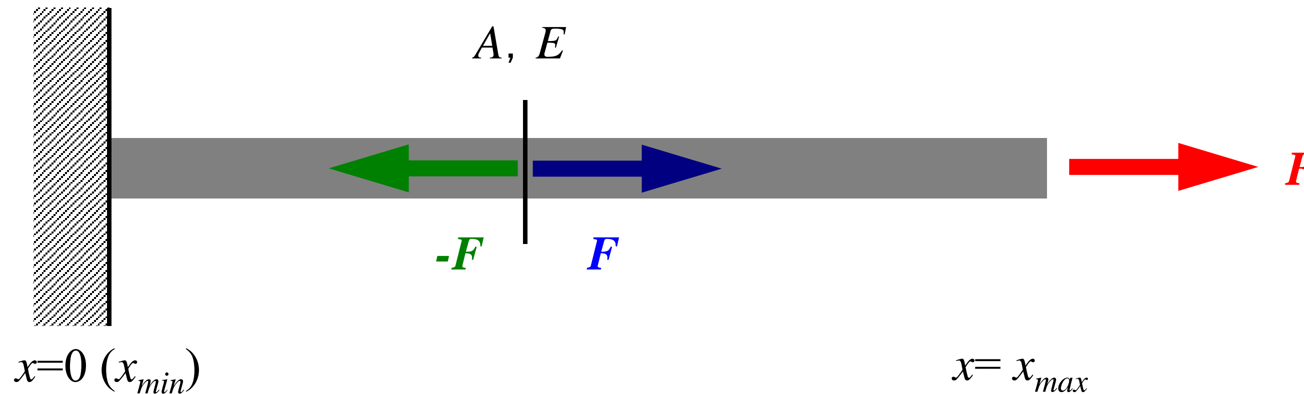
$$\sigma_x = E\varepsilon_x$$

Results Expected



- Equilibrium State: Uniform axial force at any section
  - Uniform stress: $\sigma = F/A$
  - Uniform strain: $\varepsilon = \sigma/E$
  - Uniform displacement for each element, if size of each element is uniform (deformation rate= ε)

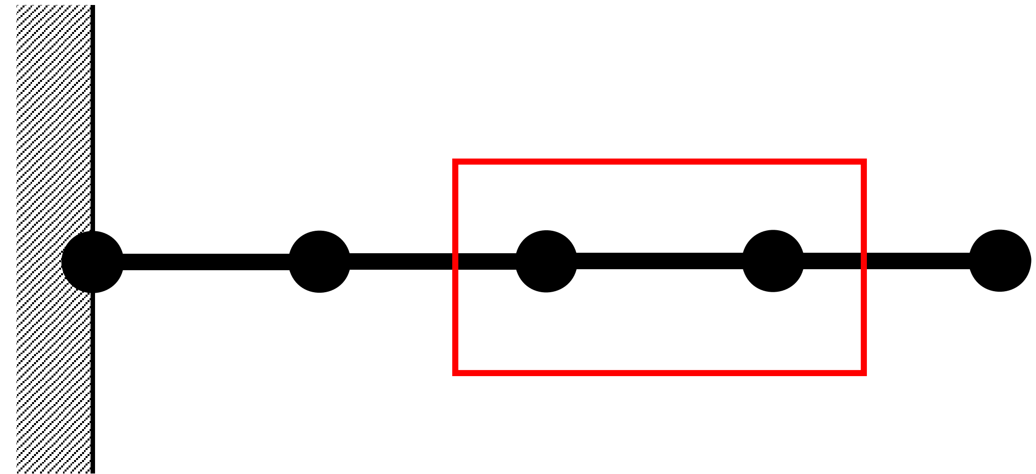
Results Expected



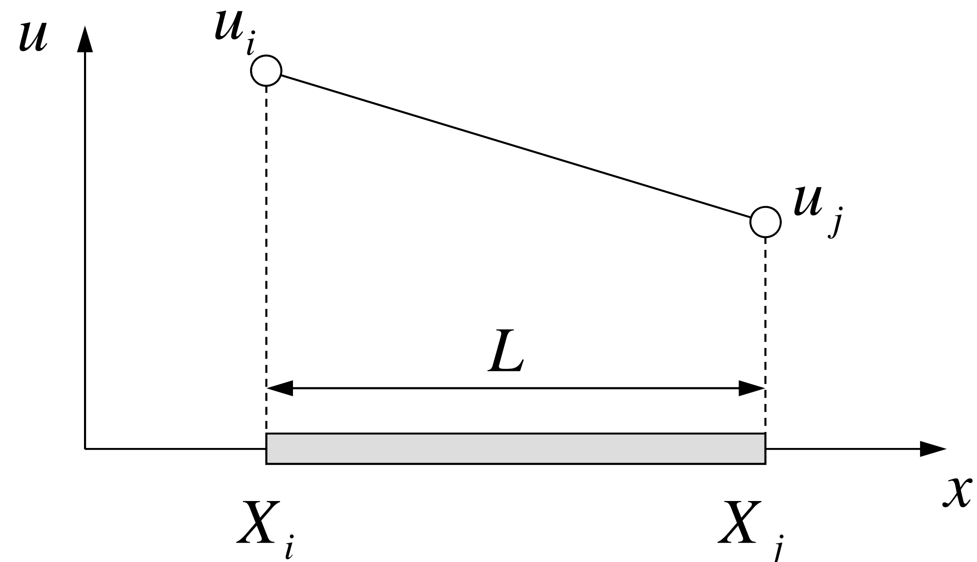
- $x_{max}=10$, $F=5$, $A=2$, $E=10$.
- Equilibrium State: Uniform axial force at any section
  - Uniform stress: $\sigma = F/A = 2.5$
  - Uniform strain: $\varepsilon = \sigma/E = 0.25$
  - Uniform displacement for each element, if size of each element is uniform (deformation rate= ε) $u= 2.5@\ x=x_{max}$

1D Linear Element (1/4)
一次元線形要素

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



  - u_i Displacement at i
  - u_j Displacement at j



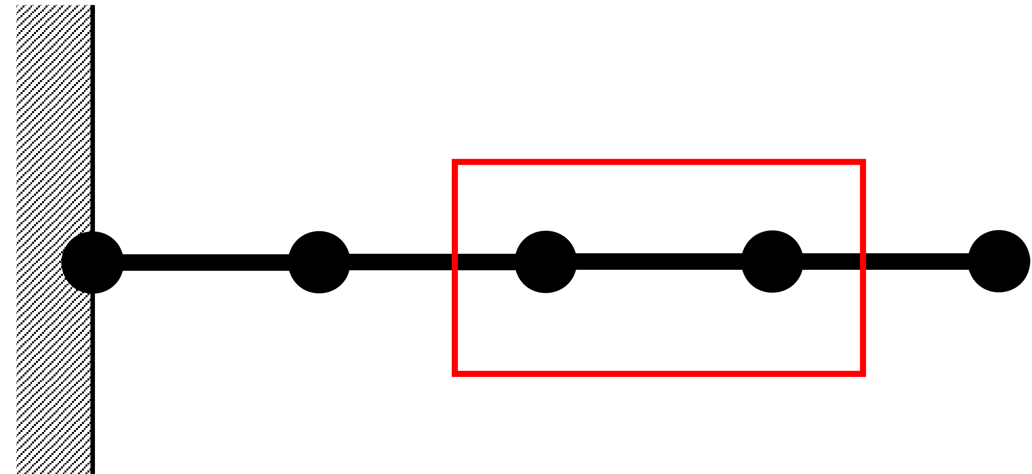
  - Displacement u on each element is linear function of x (Piecewise Linear):

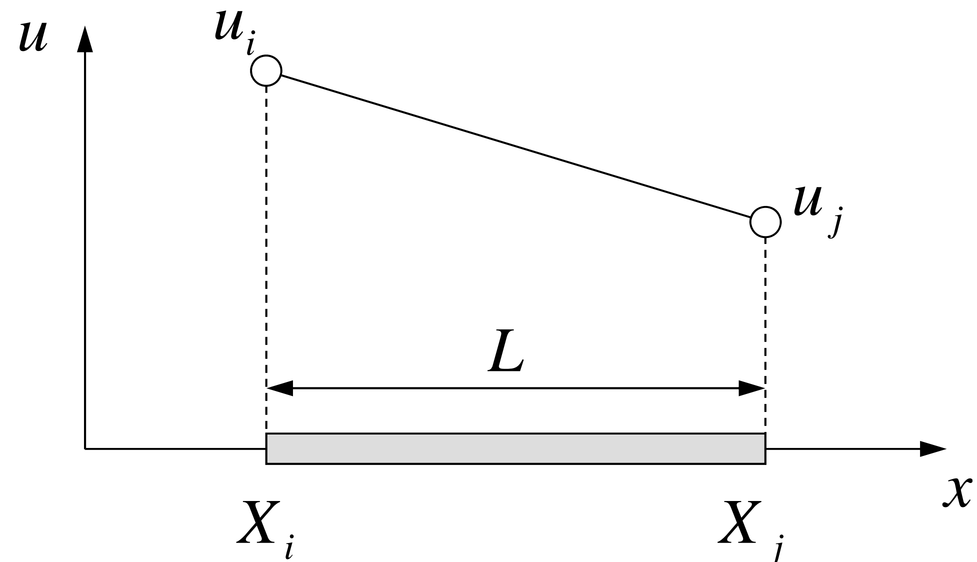
$$u = \alpha_1 + \alpha_2 x$$

1D Linear Element (1/4)

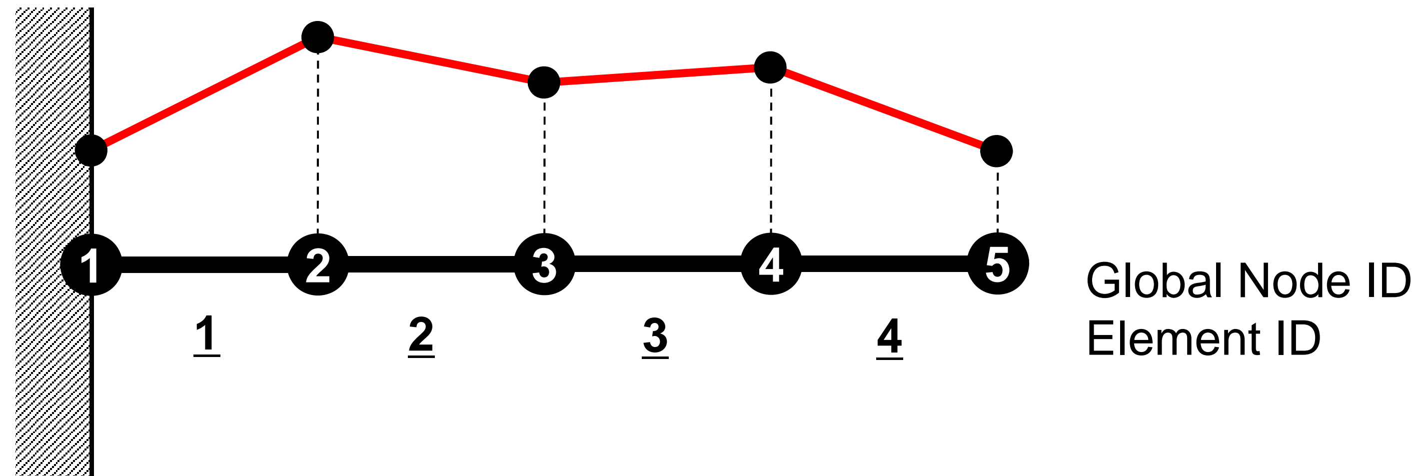
- 1D Linear Element
  - Length= L
    - Node (Vertex)
    - Element
  - u_i Displacement at i
  - u_j Displacement at j
  - Displacement u on each element is linear function of x (Piecewise Linear):

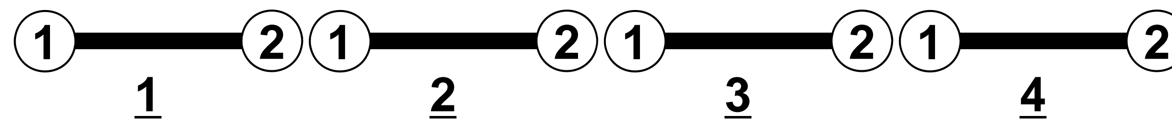
$$u = \alpha_1 + \alpha_2 x$$





Piecewise Linear





Local Node ID
for each elem.

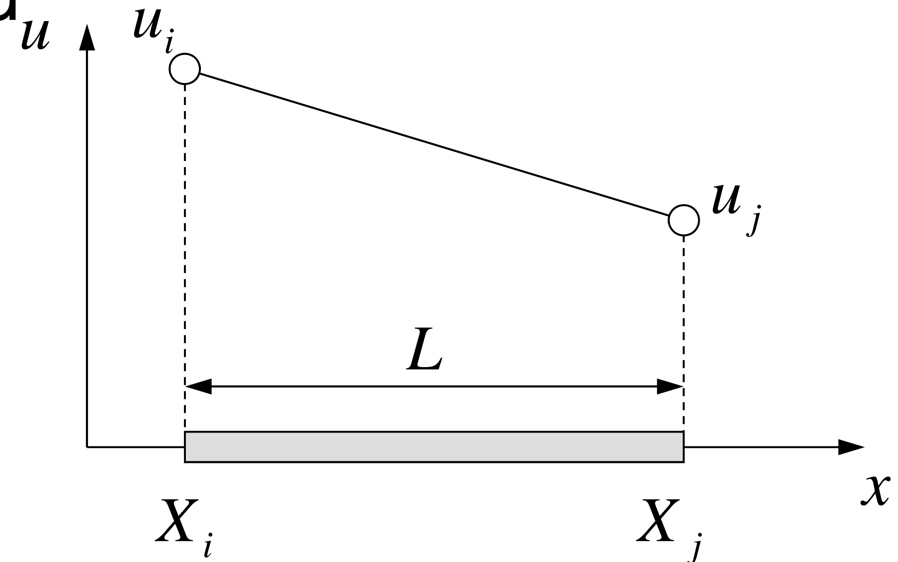
Strain and stress are constant in each element (might be discontinuous at each “node”)

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

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

$$u = u_i \ @\ x = X_i, \quad u = u_j \ @\ x = X_j$$

$$u_i = \alpha_1 + \alpha_2 X_i, \quad u_j = \alpha_1 + \alpha_2 X_j$$



- Coefficients:

$$\alpha_1 = \frac{u_i X_j - u_j X_i}{L}, \alpha_2 = \frac{u_j - u_i}{L}$$

- u can be written as follows, according to u_i and u_j:

$$u = \left(\frac{X_j - x}{L}\right) u_i + \left(\frac{x - X_i}{L}\right) u_j$$

N_i N_j

N_i, N_j

Shape Function（形状関数）or

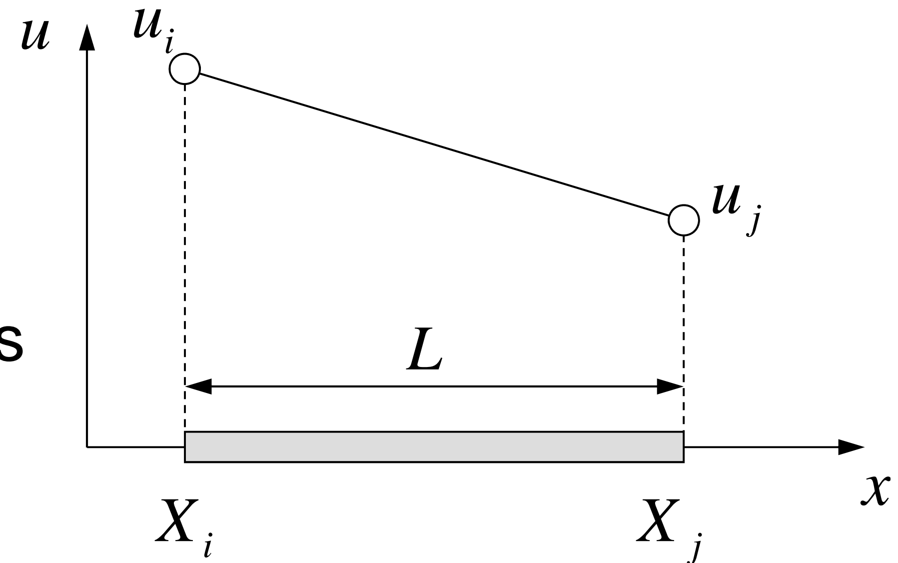
Interpolation Function（内挿〃）

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): Displacement at each node

$$u = N_i u_i + N_j u_j$$

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

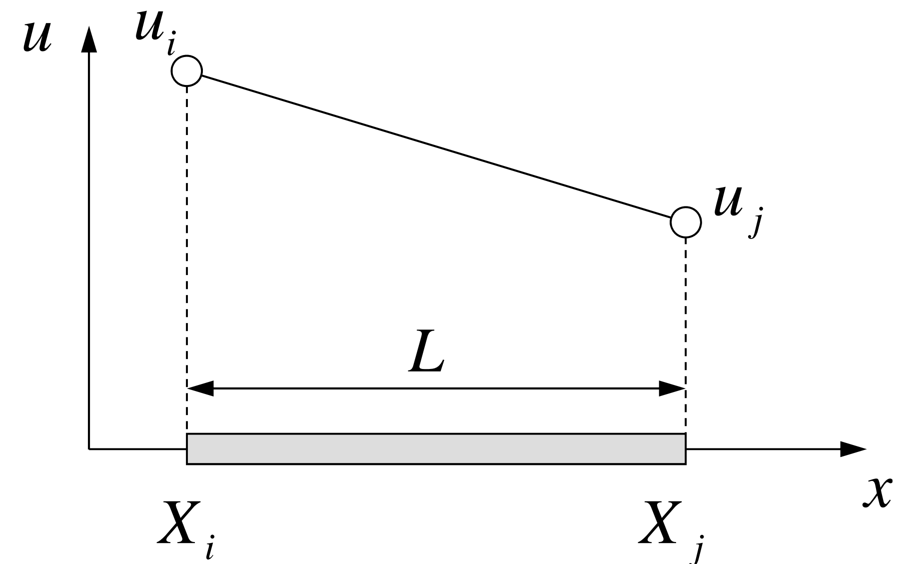
Ψ_i Trial/Test Function (known function of position, defined in domain and at boundary. "Basis" in linear algebra.

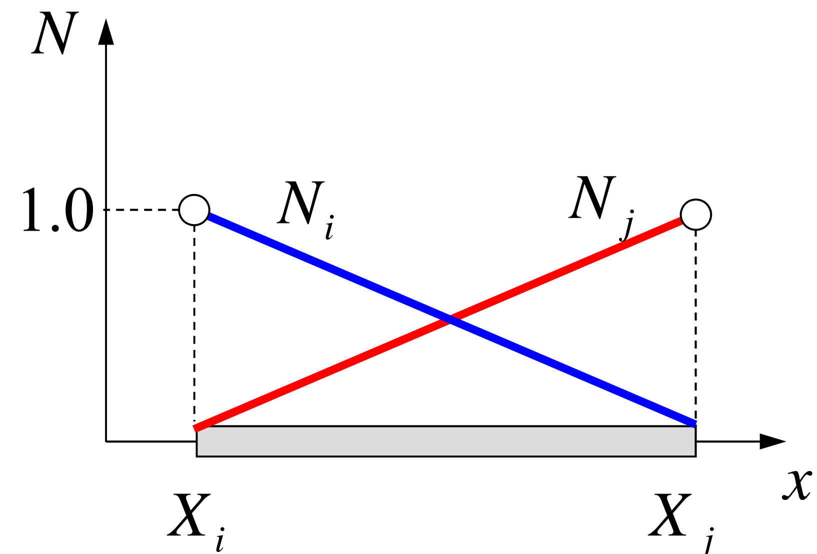
a_i Coefficients (unknown)

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

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

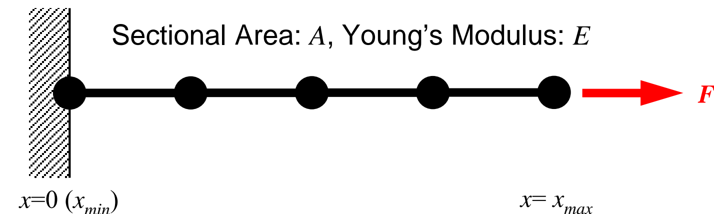
$$N_i = \left(\frac{X_j - x}{L} \right), \quad N_j = \left(\frac{x - X_i}{L} \right)$$





Galerkin Method (1/4)

- Governing Equation for 1D Static Linear-Elastic Problems



$$E\left(\frac{d^2u}{dx^2}\right)+X=0$$

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

Distribution of Displacement in Each Elem. (Matrix Form), ϕ : Displacement at Each Node

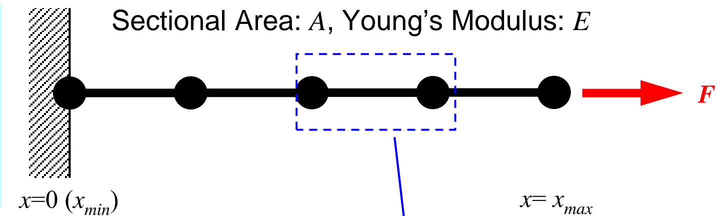
- Following integral equation is obtained at each element by Galerkin method, where $[N]$'s are also weighting functions:

$$\int_V [N]^T\left\{E\left(\frac{d^2u}{dx^2}\right)+X\right\}dV=0$$

Galerkin Method (2/4)

- Green's Theorem (1D)

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



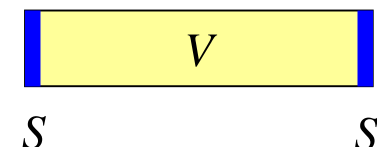
- Apply this to the 1st part of eqn with 2nd-order diff.:

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

- Consider the following terms:

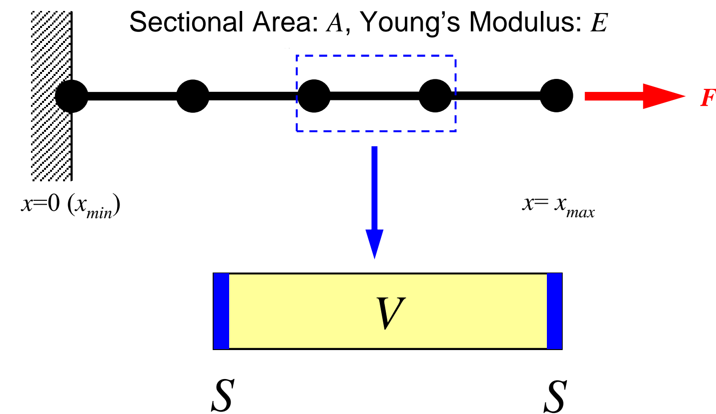
$$u = [N]\{\phi\}, \quad \frac{du}{dx} = \frac{d[N]}{dx}\{\phi\} \qquad \overline{\sigma} = E\frac{du}{dx}$$

Stress on Surfaces of Each Element



Galerkin Method (3/4)

- Finally following eqn is obtained by considering term due to body forces (X)

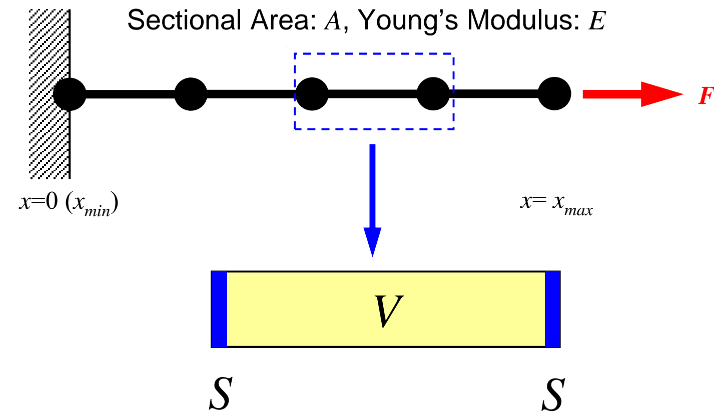


$$
-\int_V E\left(\frac{d[N]^T}{dx}\frac{d[N]}{dx}\right)dV\cdot\{\phi\}
+\int_S \bar{\sigma}[N]^T dS+\int_V X[N]^T dV=0
$$

- 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)

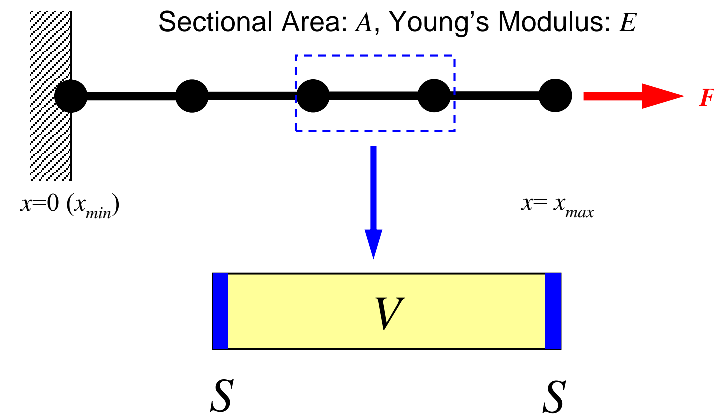
$$-\int_V E\left(\frac{d[N]^T}{dx}\frac{d[N]}{dx}\right)dV\cdot\{\phi\}$$
$$+\int_S \bar{\sigma}[N]^T dS + \int_V X[N]^T dV = 0$$



- 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）



$$
-\int_V E\left(\frac{d[N]^T}{dx}\frac{d[N]}{dx}\right)dV\cdot\{\phi\}
+\int_S \bar{\sigma}[N]^T dS+\int_V X[N]^T dV=0
$$

where $\bar{\sigma}=E\dfrac{du}{dx}$

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

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

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

$$\left[f\right]^{(e)}=\int_V X\left[N\right]^T dV+\int_S \overline{\sigma}\left[N\right]^T dS$$

Integration over Each Element: $[k]$

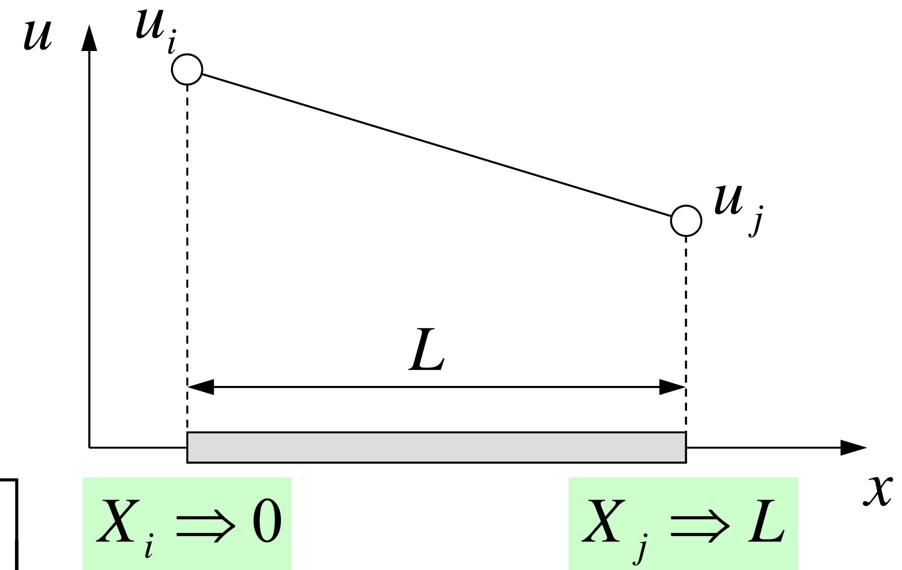
$$N_i = \left(\frac{X_j - x}{L}\right), \quad N_j = \left(\frac{x - X_i}{L}\right) \qquad \frac{dN_i}{dx} = \left(\frac{-1}{L}\right), \quad \frac{dN_j}{dx} = \left(\frac{1}{L}\right)$$

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

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

2x1 matrix   1x2 matrix

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



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

A: Sectional Area
L: Length

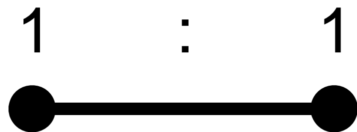
Integration over Each Element: $\{f\}$ (1/2)

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

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

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

Body Force



A: Sectional Area

L: Length

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

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

$$\int_V X[N]^T dV = XA\int_0^L \begin{bmatrix} 1 - x/L \\ x/L \end{bmatrix} dx = \frac{XAL}{2}\begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$$ Body Force

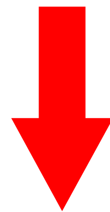
$$\int_S \overline{\sigma}[N]^T dS = \overline{\sigma}A\Big|_{x=L} = \overline{\sigma}A\begin{Bmatrix} 0 \\ 1 \end{Bmatrix}$$ Surface Force

when surface force acts on
only this surface.

Global Equations

- Accumulate Element Equations:

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



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

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

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

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

ECCS 2012 System

Creating Directory

```
>$ cd Documents      create your directory under this (good for Windows)
>$ mkdir fem1        your favorite name
>$ cd fem1
```

This is your “top” directory, and is called **<$fem1>** in this class.

1D Code for Static Linear-Elastic Problems

```
>$ cd <$fem1>
>$ cp /home03/skengon/Documents/class/fem1/1d.tar .
>$ tar xvf 1d.tar
>$ cd 1d
```

Compile & GO !

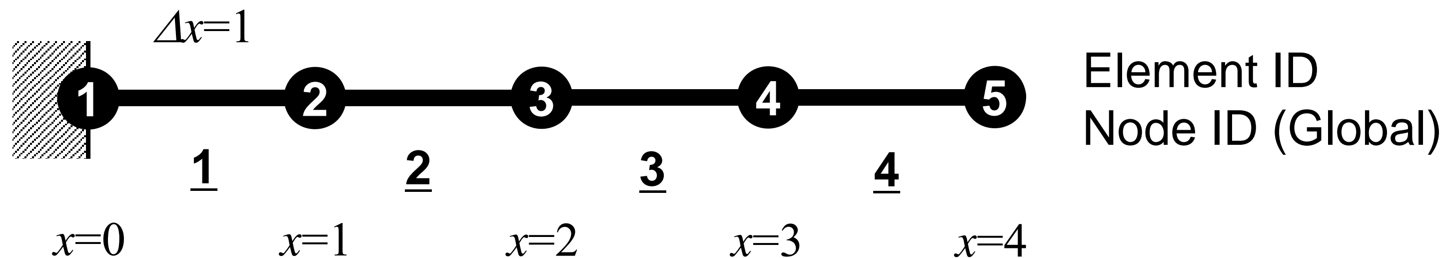
```
>$ cd <$fem1>/1d
>$ cc -O 1d.c        (or g95 -O 1d.f)
>$ ./a.out
```

Control Data `input.dat`

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

$$\sigma = \frac{F}{A} = \frac{1}{1} = 1$$

$$\frac{du}{dx} = \varepsilon = \frac{\sigma}{E} = \frac{1}{1} = 1$$

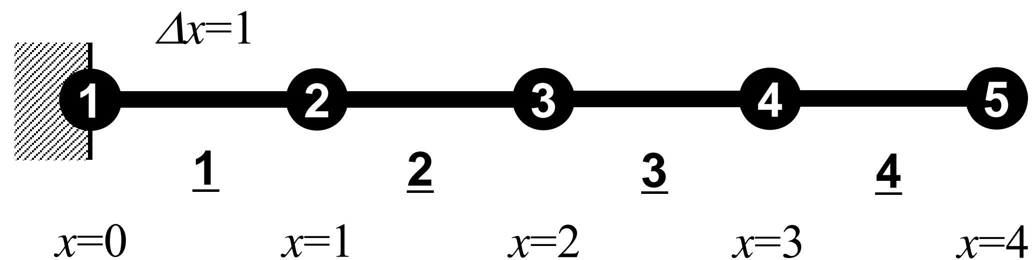


Results

```
>$ ./a.out

 4 iters, RESID=    0.000000E+00 U(N)=     4.000000E+00

### DISPLACEMENT       at each node (computed)
       1     0.000000E+00
       2     1.000000E+00
       3     2.000000E+00
       4     3.000000E+00
       5     4.000000E+00

### STRESS             at each element (computed, theoretical)
       1     1.000000E+00     1.000000E+00
       2     1.000000E+00     1.000000E+00
       3     1.000000E+00     1.000000E+00
       4     1.000000E+00     1.000000E+00
```

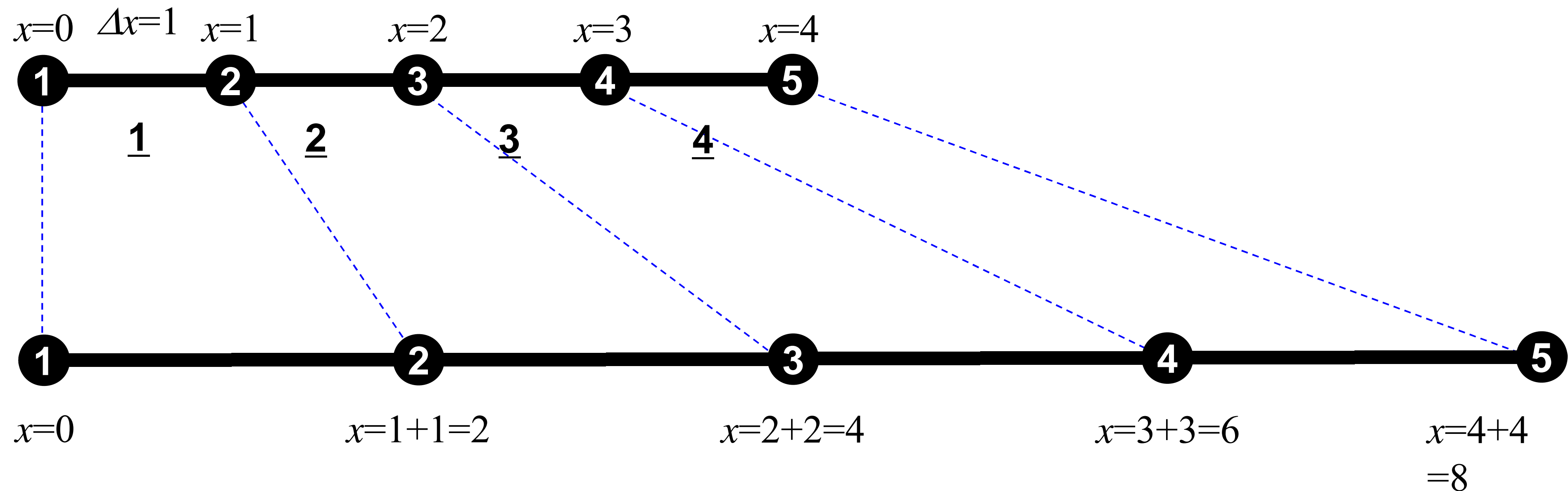
$$\sigma = \frac{F}{A} = \frac{1}{1} = 1$$

$$\frac{du}{dx} = \varepsilon = \frac{\sigma}{E} = \frac{1}{1} = 1$$



Strain=1.00 (100%)

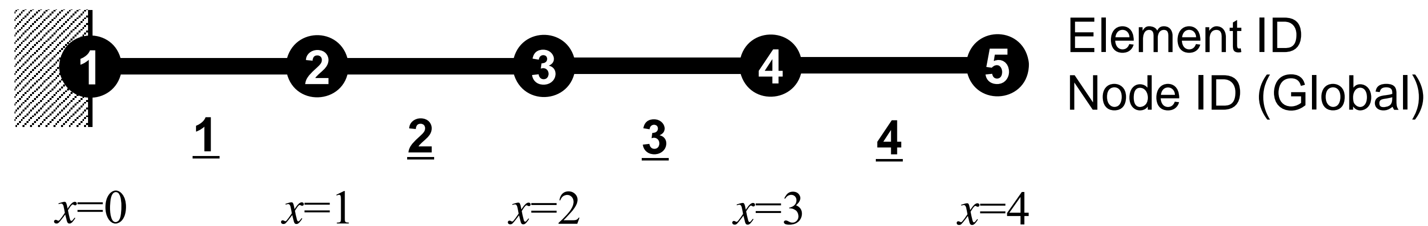
Length of each element is doubled compared to initial condition




```
### DISPLACEMENT        at each node (computed)
           1      0.000000E+00
           2      1.000000E+00
           3      2.000000E+00
           4      3.000000E+00
           5      4.000000E+00
```

Element Eqn's/Accumulation (1/3)

- 4 elements, 5 nodes



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

$$[k]^{(1)} = \frac{EA}{L}\begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \qquad \{f\}^{(1)} = \frac{XAL}{2}\begin{Bmatrix} 1 \\ 1 \end{Bmatrix} + \int_S \bar{\sigma}[N]^T dS = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}$$

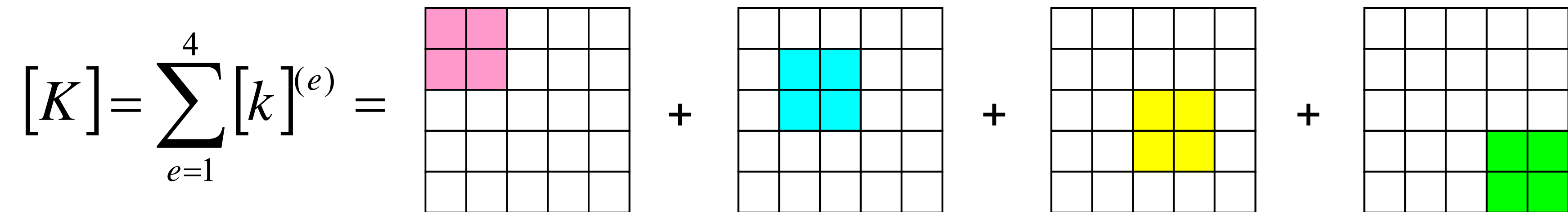
X=0 in this case

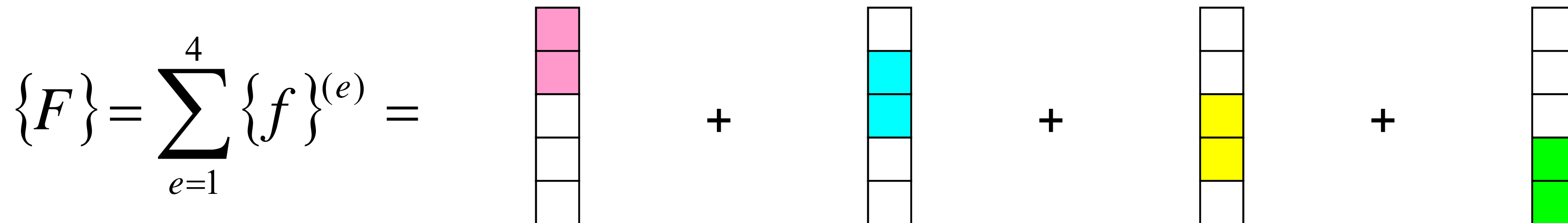
- As for Element-4:

$$[k]^{(4)} = \frac{EA}{L}\begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \qquad \{f\}^{(4)} = \int_S \bar{\sigma}[N]^T dS = \bar{\sigma}A\begin{Bmatrix} 0 \\ 1 \end{Bmatrix} = \begin{Bmatrix} 0 \\ F \end{Bmatrix}$$

Element Eqn's/Accumulation (2/3)

- Element-by-Element Accumulation:





Element Eqn's/Accumulation (3/3)

- Relations to FDM

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

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

$$= \begin{bmatrix} +1 & -1 & & & \\ -1 & +2 & -1 & & \\ & -1 & +2 & -1 & \\ & & -1 & +2 & -1 \\ & & & -1 & +1 \end{bmatrix} \times \frac{EA}{L}$$

$$-\int_V \left(\frac{d^2u}{dx^2}\right) dV = -\int_V \left(\frac{u_{i+1} - 2u_i + u_{i-1}}{L^2}\right) dV$$

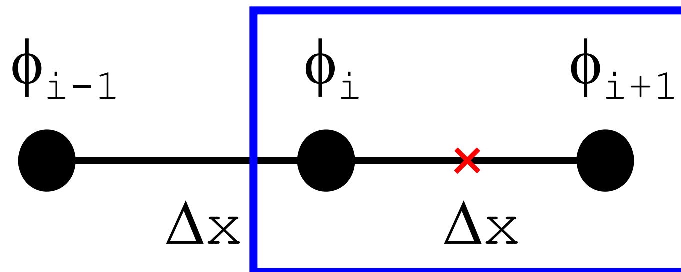
$$= -\left(\frac{u_{i+1} - 2u_i + u_{i-1}}{L^2}\right) \cdot AL = -\left(u_{i+1} - 2u_i + u_{i-1}\right) \cdot \frac{A}{L}$$

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.

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

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

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

Element/Global Operations

1 2 3 4

1 2 3 4 5 Global Node ID

Local Node ID

e e e e

1 2 1 2 1 2 1 2

$$-\int_V E\left(\frac{d[N]^T}{dx}\frac{d[N]}{dx}\right)dV\cdot\{\phi\}+\int_S \bar{\sigma}[N]^T dS+\int_V X[N]^T dV=0$$

$$[k]^{(1)}\{\phi\}^{(1)}=\{f\}^{(1)}$$

$$\begin{bmatrix}k_{11}^{(1)} & k_{12}^{(1)}\\ k_{21}^{(1)} & k_{22}^{(1)}\end{bmatrix}\begin{Bmatrix}\phi_1^{(1)}\\ \phi_2^{(1)}\end{Bmatrix}=\begin{Bmatrix}f_1^{(1)}\\ f_2^{(1)}\end{Bmatrix}$$

$$-\int_V E\left(\frac{d[N]^T}{dx}\frac{d[N]}{dx}\right)dV\cdot\{\phi\}+\int_S \bar{\sigma}[N]^T dS+\int_V X[N]^T dV=0$$

$$[k]^{(2)}\{\phi\}^{(2)}=\{f\}^{(2)}$$

$$\begin{bmatrix}k_{11}^{(2)} & k_{12}^{(2)}\\ k_{21}^{(2)} & k_{22}^{(2)}\end{bmatrix}\begin{Bmatrix}\phi_1^{(2)}\\ \phi_2^{(2)}\end{Bmatrix}=\begin{Bmatrix}f_1^{(2)}\\ f_2^{(2)}\end{Bmatrix}$$

$$-\int_V E\left(\frac{d[N]^T}{dx}\frac{d[N]}{dx}\right)dV\cdot\{\phi\}+\int_S \bar{\sigma}[N]^T dS+\int_V X[N]^T dV=0$$

$$[k]^{(3)}\{\phi\}^{(3)}=\{f\}^{(3)}$$

$$\begin{bmatrix}k_{11}^{(3)} & k_{12}^{(3)}\\ k_{21}^{(3)} & k_{22}^{(3)}\end{bmatrix}\begin{Bmatrix}\phi_1^{(3)}\\ \phi_2^{(3)}\end{Bmatrix}=\begin{Bmatrix}f_1^{(3)}\\ f_2^{(3)}\end{Bmatrix}$$

$$-\int_V E\left(\frac{d[N]^T}{dx}\frac{d[N]}{dx}\right)dV\cdot\{\phi\}+\int_S \bar{\sigma}[N]^T dS+\int_V X[N]^T dV=0$$

$$[k]^{(4)}\{\phi\}^{(4)}=\{f\}^{(4)}$$

$$\begin{bmatrix}k_{11}^{(4)} & k_{12}^{(4)}\\ k_{21}^{(4)} & k_{22}^{(4)}\end{bmatrix}\begin{Bmatrix}\phi_1^{(4)}\\ \phi_2^{(4)}\end{Bmatrix}=\begin{Bmatrix}f_1^{(4)}\\ f_2^{(4)}\end{Bmatrix}$$

1 2 3 4

1 2 3 4 5

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

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

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

Around the FEM in 3 minutes

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
- Higher-order Elements
- Numerical Integration, Isoparametric Elements
- Report #1

Large-Scale Linear Equations in Scientific Applications

- Solving large-scale linear equations **Ax=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

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**

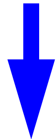
Non-Stationary/Krylov Subspace Method (1/2)
非定常法・クリロフ部分空間法

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

Compute $\mathbf{x_0}$, $\mathbf{x_1}$, $\mathbf{x_2}$, ..., $\mathbf{x_k}$ by the following iterative procedures:

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

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



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

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

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:

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

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

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

Procedures of Conjugate Gradient

```
Compute r(0)= b-[A]x(0)
for i= 1, 2, …
    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

$x^{(i)}$: 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 $r^{(0)}= b-[A]x^{(0)}$
<u>for</u> i= 1, 2, …
    $z^{(i-1)}= r^{(i-1)}$
    $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
    <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>

$x^{(i)}$: Vector
α_i : Scalar

Derivation of CG Algorithm (1/5)

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

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

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

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

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

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

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

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

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

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:

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

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

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

Derivation of CG Algorithm (3/5)

Residual vector at $(k+1)$-th iteration: $r^{(k+1)} = b - Ax^{(k+1)},\ r^{(k)} = b - Ax^{(k)}$

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

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

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

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

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:

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

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

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

Thus, following relation is obtained:

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

Derivation of CG Algorithm (5/5)

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

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

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

```
Compute p(0)=r(0)= b-[A]x(0)
for i= 1, 2, …
    calc. αi-1
    x(i)= x(i-1) + αi-1p(i-1)
    r(i)= r(i-1) - αi-1[A]p(i-1)

    check convergence |r|
    (if not converged)
    calc. βi-1
    p(i)= r(i) + βi-1 p(i-1)
end
```

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

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

Properties of CG Algorithm

Following "conjugate（共役）" 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.

Top 10 Algorithms in the 20th Century (SIAM)

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

Proof (1/3)

Mathematical Induction
数学的帰納法

$$\left(r^{(i)}, r^{(j)}\right) = 0 \; (i \neq j)$$ <u>Orthogonal</u>

$$\left(p^{(i)}, Ap^{(j)}\right) = 0 \; (i \neq j)$$ <u>Conjugate</u>

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

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

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

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

Proof (2/3)

Mathematical Induction
数学的帰納法

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

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

if $i < k$

$$\left(r^{(k+1)}, r^{(i)}\right) = \left(r^{(i)}, r^{(k+1)}\right) \overset{(2)}{=} \left(r^{(i)}, r^{(k)} - \alpha_k Ap^{(k)}\right)$$
$$\overset{(*)}{=} -\alpha_k\left(r^{(i)}, Ap^{(k)}\right) \overset{(4)}{=} -\alpha_k\left(p^{(i)} - \beta_{i-1}p^{(i-1)}, Ap^{(k)}\right)$$
$$= -\alpha_k\left(p^{(i)}, Ap^{(k)}\right) + \alpha_k\beta_{i-1}\left(p^{(i-1)}, Ap^{(k)}\right) \overset{(*)}{=} 0$$

if $i = k$

$$\left(r^{(k+1)}, r^{(k)}\right) \overset{(2)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(r^{(k)}, \alpha_k Ap^{(k)}\right)$$
$$\overset{(3)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(p^{(k)} - \beta_{k-1}p^{(k-1)}, \alpha_k Ap^{(k)}\right)$$
$$\overset{(*)}{=} \left(r^{(k)}, r^{(k)}\right) - \alpha_k\left(p^{(k)}, Ap^{(k)}\right) \overset{(1)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(p^{(k)}, r^{(k)}\right)$$
$$\overset{(3)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(\beta_{k-1}p^{(k-1)} + r^{(k)}, r^{(k)}\right)$$
$$= -\beta_{k-1}\left(p^{(k-1)}, r^{(k)}\right) \overset{(2)}{=} -\beta_{k-1}\left(p^{(k-1)}, r^{(k-1)} - \alpha_{k-1}Ap^{(k-1)}\right)$$
$$= -\beta_{k-1}\left\{\left(p^{(k-1)}, r^{(k-1)}\right) - \alpha_{k-1}\left(p^{(k-1)}, Ap^{(k-1)}\right)\right\} \overset{(1)}{=} 0$$

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

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

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

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

Proof (3/3)

Mathematical Induction

数学的帰納法

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

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

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

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

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

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

<u>if $i < k$</u>

$$\left(p^{(k+1)}, Ap^{(i)}\right) \overset{(3)}{=} \left(r^{(k+1)} + \beta_k p^{(k)}, Ap^{(i)}\right)$$
$$\overset{(*)}{=} \left(r^{(k+1)}, Ap^{(i)}\right)$$
$$\overset{(2)}{=} \frac{1}{\alpha_k}\left(r^{(k+1)}, r^{(i)} - r^{(i-1)}\right) = 0$$

<u>if $i = k$</u>

$$\left(p^{(k+1)}, Ap^{(k)}\right) \overset{(3)}{=} \left(r^{(k+1)}, Ap^{(k)}\right) + \beta_k\left(p^{(k)}, Ap^{(k)}\right)$$
$$\overset{(4)}{=} 0$$

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

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

$$\overset{(3)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(p^{(k)} - \beta_{k-1} p^{(k-1)}, \alpha_k A p^{(k)}\right)$$

$$\overset{(*)}{=} \left(r^{(k)}, r^{(k)}\right) - \alpha_k \left(p^{(k)}, A p^{(k)}\right) \overset{(1)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(p^{(k)}, r^{(k)}\right) = 0$$

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

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

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

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

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

$\alpha_k,\ \beta_k$

Usually, we use simpler definitions of $\alpha_k,\ \beta_k$ 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

$$\beta_{i-1} = \frac{\left(r^{(i-1)}, r^{(i-1)}\right)}{\left(r^{(i-2)}, r^{(i-2)}\right)} \qquad \begin{matrix} (= \rho_{i-1}) \\ (= \rho_{i-2}) \end{matrix}$$

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

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（条件数）.
- A preconditioner **M** (whose properties are similar to those of **A**)transforms the linear system into one with more favorable spectral properties（前処理）
  - **M** transforms original equation **Ax=b** into **A'x=b'** where **A'=M^{-1}A**, **b'=M^{-1}b**
  - If **M~A**, **M^{-1}A** is close to identity matrix.
  - If **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（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]z(i-1)= r(i-1)` ($[M]z^{(i-1)} = r^{(i-1)}$) is very easy.
- Provides fast convergence for simple problems.
- 1d.f, 1d.c

- More detailed discussions on preconditioning will be provided in “3D code”.

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

- Program

- Higher-order Elements
- Numerical Integration, Isoparametric Elements
- Report #1

Coef. Matrix derived from FEM

- Sparse Matrix
  - Many “0”’s
- Storing all components (e.g. $\mathrm{A}(i,j)$) is not efficient for sparse matrices
  - $\mathrm{A}(i,j)$ 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 :
    - $\mathrm{A}(i,j)$: $O(10^{16})$ 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)

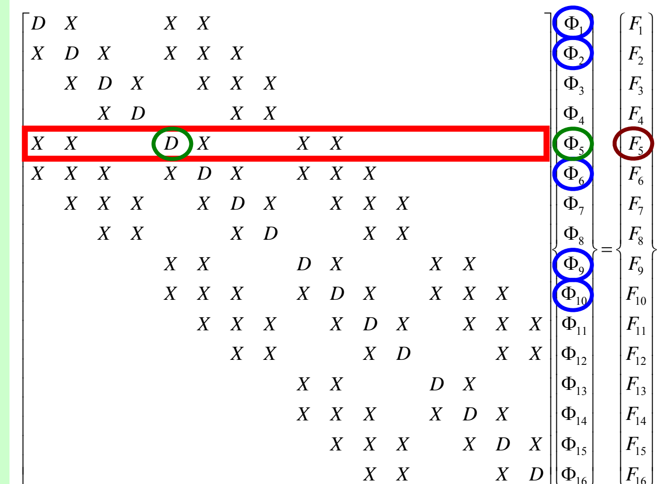
Only non-zero components are stored according to “Compressed Row Storage”.

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
```



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)

$$
\begin{array}{c} \\ ❶ \\ ❷ \\ ❸ \\ ❹ \\ ❺ \\ ❻ \\ ❼ \\ ❽ \end{array}
\begin{array}{c}
\begin{array}{cccccccc} ❶ & ❷ & ❸ & ❹ & ❺ & ❻ & ❼ & ❽ \end{array} \\
\begin{bmatrix}
1.1 & 2.4 & 0 & 0 & 3.2 & 0 & 0 & 0 \\
4.3 & 3.6 & 0 & 2.5 & 0 & 3.7 & 0 & 9.1 \\
0 & 0 & 5.7 & 0 & 1.5 & 0 & 3.1 & 0 \\
0 & 4.1 & 0 & 9.8 & 2.5 & 2.7 & 0 & 0 \\
3.1 & 9.5 & 10.4 & 0 & 11.5 & 0 & 4.3 & 0 \\
0 & 0 & 6.5 & 0 & 0 & 12.4 & 9.5 & 0 \\
0 & 6.4 & 2.5 & 0 & 0 & 1.4 & 23.1 & 13.1 \\
0 & 9.5 & 1.3 & 9.6 & 0 & 3.1 & 0 & 51.3
\end{bmatrix}
\end{array}
$$

Compressed Row Storage (CRS): C Numbering starts from 0 in program

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

```
N= 8
```

Diagonal Components

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

Compressed Row Storage (CRS): C

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

Compressed Row Storage (CRS): C

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

NPLU= 25
(=Index[N])

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

Compressed Row Storage (CRS): C

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

NPLU= 25
(=Index[N])

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

Compressed Row Storage (CRS)：C

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

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

Compressed Row Storage (CRS) : C

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

Diag (i) Diagonal Components (REAL, i=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}

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

- 1D-code for Static Linear-Elastic Problems by Galerkin FEM
- Sparse Linear Solver
  - Conjugate Gradient Method
  - Preconditioning
- Storage of Sparse Matrices
- Program
- Higher-order Elements
- Numerical Integration, Isoparametric Elements
- Report #1

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
- Calculation of Stress

Program: 1d.c (1/7)
variables and arrays

```
/*
//   1D Solid Mechanics for Truss Elements solved by
//   CG (Conjugate Gradient) Method
//
//   d/dx(EdU/dx) + F = 0
//   U=0@x=0
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <assert.h>

int main(){
          int NE, N, NPLU, IterMax, errno;
          int R, Z, Q, P, DD;

          double dX, Resid, Eps, Area, F, Young;
          double X1, X2, U1, U2, DL, Strain, Sigma, Ck;
          double *U, *Rhs, *X;
          double *Diag, *AMat;
          double **W;

          int *Index, *Item, *Icelnod;
          double Kmat[2][2], Emat[2][2];

          int i, j, in1, in2, k, icel, k1, k2, jS;
          int iter;
          FILE *fp;
          double BNorm2, Rho, Rho1=0.0, C1, Alpha, DNorm2;
          int ierr = 1;
```

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
F	R		I	Axial Force F at X=Xmax
Young	R		I	Young’s Modulus
X1,X2,U1,U2	R		O	Location/Displacement at Local Nodes

Variable/Arrays (2/2)

Name	Type	Size	I/O	Definition
DL, Ck	R		O	Coef's for Element Matrix
Strain, Stress	R		O	Element Strain, Element Stress
X	R	N	O	Location of Each Node
U	R	N	O	Displacement of Each Node
Rhs	R	N	O	RHS Vector
Diag	R	N	O	Diagonal Components
W	R	[4][N]	O	Work Array for CG
Amat	R	NPLU	O	Off-Diagonal Components (Value)
Index	I	N+1	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

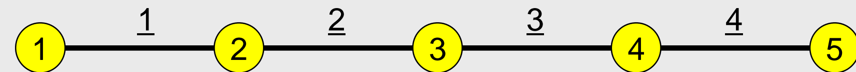
Program: 1d.c (2/7)

Initialization, Allocation of Arrays

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

        N= NE + 1;

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

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

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

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

input.dat

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



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

Program: 1d.c (2/7)

Initialization, Allocation of Arrays

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

          N= NE + 1;

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

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

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

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

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

Element/Global Operations

Global Node ID: 1 —1— 2 —2— 3 —3— 4 —4— 5

Local Node ID: 1 —e— 2 (for each element)

$$-\int_V E\left(\frac{d[N]^T}{dx}\frac{d[N]}{dx}\right)dV\cdot\{\phi\}+\int_S \bar{\sigma}[N]^T dS+\int_V X[N]^T dV=0$$

$$[k]^{(1)}\{\phi\}^{(1)}=\{f\}^{(1)}\qquad \begin{bmatrix} k_{11}^{(1)} & k_{12}^{(1)} \\ k_{21}^{(1)} & k_{22}^{(1)} \end{bmatrix}\begin{Bmatrix} \phi_1^{(1)} \\ \phi_2^{(1)} \end{Bmatrix}=\begin{Bmatrix} f_1^{(1)} \\ f_2^{(1)} \end{Bmatrix}$$

$$[k]^{(2)}\{\phi\}^{(2)}=\{f\}^{(2)}\qquad \begin{bmatrix} k_{11}^{(2)} & k_{12}^{(2)} \\ k_{21}^{(2)} & k_{22}^{(2)} \end{bmatrix}\begin{Bmatrix} \phi_1^{(2)} \\ \phi_2^{(2)} \end{Bmatrix}=\begin{Bmatrix} f_1^{(2)} \\ f_2^{(2)} \end{Bmatrix}$$

$$[k]^{(3)}\{\phi\}^{(3)}=\{f\}^{(3)}\qquad \begin{bmatrix} k_{11}^{(3)} & k_{12}^{(3)} \\ k_{21}^{(3)} & k_{22}^{(3)} \end{bmatrix}\begin{Bmatrix} \phi_1^{(3)} \\ \phi_2^{(3)} \end{Bmatrix}=\begin{Bmatrix} f_1^{(3)} \\ f_2^{(3)} \end{Bmatrix}$$

$$[k]^{(4)}\{\phi\}^{(4)}=\{f\}^{(4)}\qquad \begin{bmatrix} k_{11}^{(4)} & k_{12}^{(4)} \\ k_{21}^{(4)} & k_{22}^{(4)} \end{bmatrix}\begin{Bmatrix} \phi_1^{(4)} \\ \phi_2^{(4)} \end{Bmatrix}=\begin{Bmatrix} f_1^{(4)} \\ f_2^{(4)} \end{Bmatrix}$$

1 —1— 2 —2— 3 —3— 4 —4— 5

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

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

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

Program: 1d.c (2/7)

Initialization, Allocation of Arrays

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

        N= NE + 1;

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

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

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

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

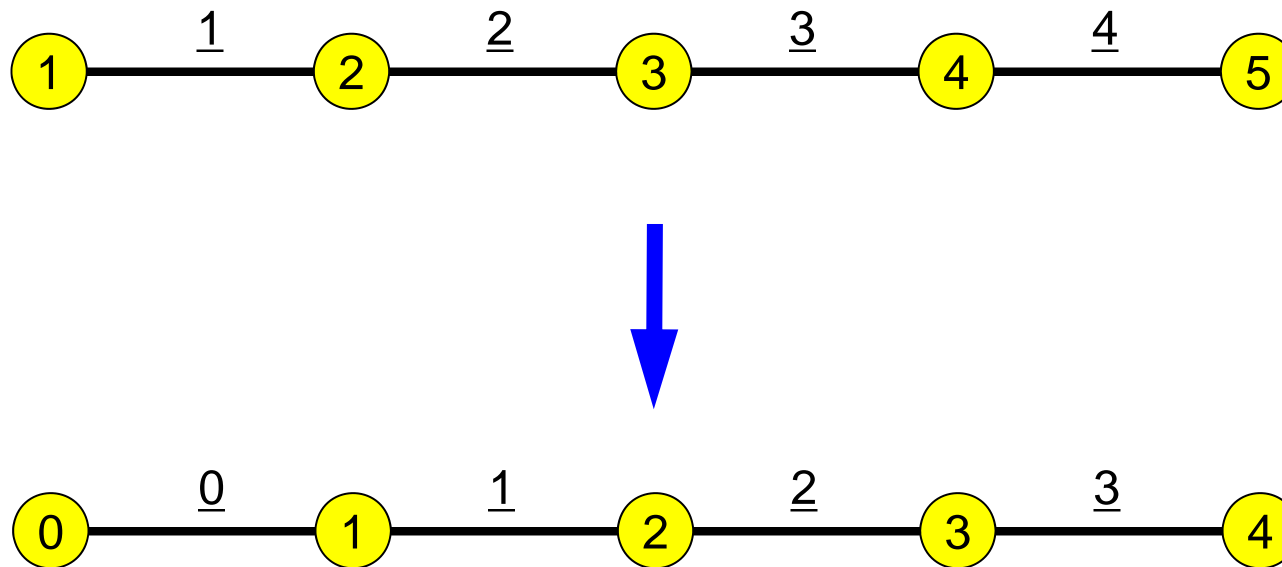
`Amat`: Non-Zero Off-Diag. Comp.
`Item`: Corresponding Column ID

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

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

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

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



Program: 1d.c (3/7)

Initialization, Allocation of Arrays (cont.)

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

  for(i=0;i<N;i++)    U[i] = 0.0;
  for(i=0;i<N;i++) Diag[i] = 0.0;
  for(i=0;i<N;i++)  Rhs[i] = 0.0;
  for(k=0;k<2*N-2;k++)  AMat[k] = 0.0;
  for(i=0;i<N;i++) X[i]= i*dX;
  for(icel=0;icel<NE;icel++){
            Icelnod[2*icel  ]= icel;
            Icelnod[2*icel+1]= icel+1;
  }

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

Program: 1d.c (3/7)

Initialization, Allocation of Arrays (cont.)

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

          for(i=0;i<N;i++)     U[i] = 0.0;
          for(i=0;i<N;i++)  Diag[i] = 0.0;
          for(i=0;i<N;i++)   Rhs[i] = 0.0;
          for(k=0;k<2*N-2;k++)   AMat[k] = 0.0;
          for(i=0;i<N;i++) X[i]= i*dX;
          for(icel=0;icel<NE;icel++){
                    Icelnod[2*icel  ]= icel;
                    Icelnod[2*icel+1]= icel+1;
          }

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

X: X-coordinate component of each node

Program: 1d.c (3/7)

Initialization, Allocation of Arrays (cont.)

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

  for(i=0;i<N;i++)    U[i] = 0.0;
  for(i=0;i<N;i++) Diag[i] = 0.0;
  for(i=0;i<N;i++)  Rhs[i] = 0.0;
  for(k=0;k<2*N-2;k++)  AMat[k] = 0.0;
  for(i=0;i<N;i++) X[i]= i*dX;
  for(icel=0;icel<NE;icel++){
          Icelnod[2*icel  ]= icel;
          Icelnod[2*icel+1]= icel+1;
  }

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

Program: 1d.c (3/7)

Initialization, Allocation of Arrays (cont.)

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

  for(i=0;i<N;i++)     U[i] = 0.0;
  for(i=0;i<N;i++)  Diag[i] = 0.0;
  for(i=0;i<N;i++)   Rhs[i] = 0.0;
  for(k=0;k<2*N-2;k++)   AMat[k] = 0.0;
  for(i=0;i<N;i++) X[i]= i*dX;
  for(icel=0;icel<NE;icel++){
            Icelnod[2*icel  ]= icel;
            Icelnod[2*icel+1]= icel+1;
  }

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

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

[Kmat]

Program: 1d.c (4/7)

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

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

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

        NPLU= Index[N];

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

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

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

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

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

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

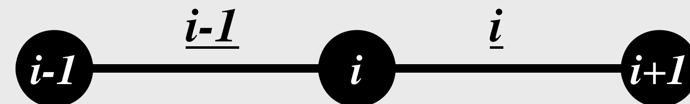
Program: 1d.c (4/7)

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

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

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

        NPLU= Index[N];

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

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

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



Program: 1d.c (5/7)

Element Matrix ~ Global Matrix

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

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

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

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

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

Program: 1d.c (5/7)
Element Matrix ~ Global Matrix

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

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

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

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

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

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

Program: 1d.c (5/7)
Element Matrix ~ Global Matrix

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

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

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

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

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

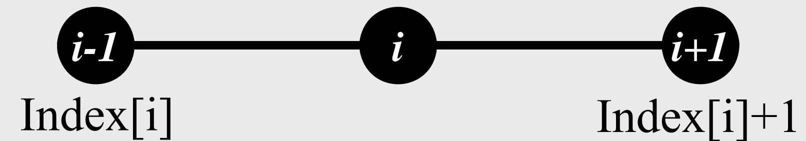
$$[Emat] = [k]^{(e)} = \frac{EA}{L}\begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}$$

Program: 1d.c (5/7)

Element Matrix ~ Global Matrix

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

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

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

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

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

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



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

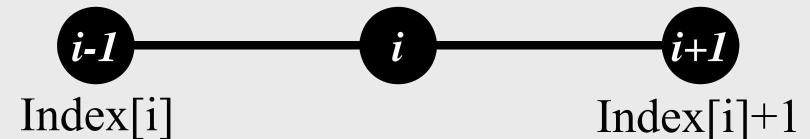
k1 (upper-right −1), k2 (lower-left −1)

Program: 1d.c (5/7)

Element Matrix ~ Global Matrix

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

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

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

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

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

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



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

k1

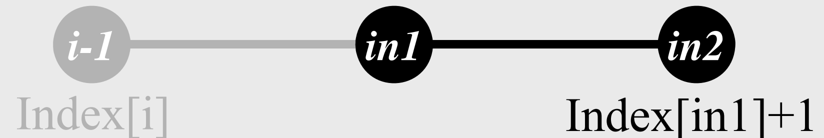
k2

General Elements: k1

"in2" as a off-diag. component of "in1"

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

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

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

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

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

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



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

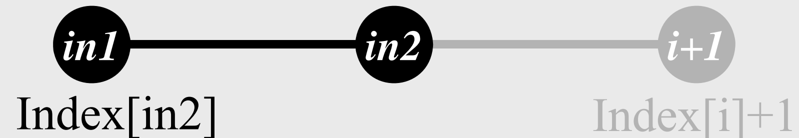
k1

General Elements: k2

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

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

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

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

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

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

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



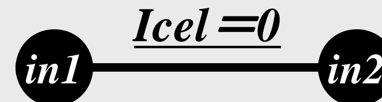
$$[Emat] = [k]^{(e)} = \frac{EA}{L}\begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}$$

k2

0-th Element: k1

"in2" as a off-diag. component of "in1"

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

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

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

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

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



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

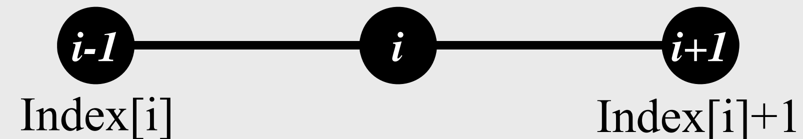
$$[Emat]=[k]^{(e)}=\frac{EA}{L}\begin{bmatrix}+1 & -1\\ -1 & +1\end{bmatrix}$$

k1

Program: 1d.c (5/7)
Element Matrix ~ Global Matrix

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

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

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

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

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

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



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

Program: 1d.c (6/7)
Boundary Conditions

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

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

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

/* X=Xmax */
        i=N-1;
        Rhs[i]= F;
```

1D Static Linear Elastic Problem

- Only deforms in x-direction (displacement: u)
  - Uniform: Sectional Area A, Young's Modulus E
  - Boundary Conditions (B.C.)
    - x=0 : u=0 (fixed)
    - x=x_{max} : F (axial force)
- Truss: NO bending deformation by G-force

(Linear) Equation at x=0
u_1= 0 (or u_0= 0)

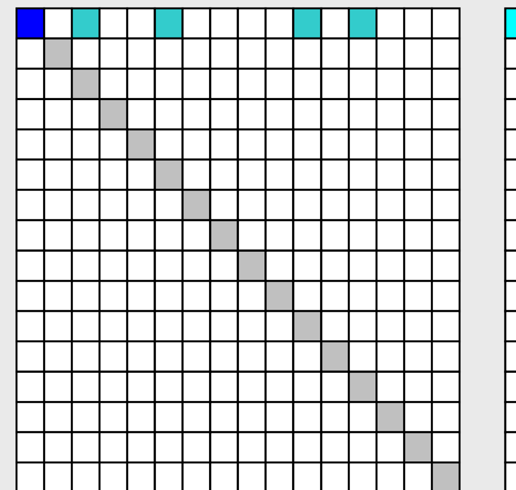
- Only deforms in x-direction (displacement: u)
  - Uniform: Sectional Area A, Young's Modulus E
  - Boundary Conditions (B.C.)
    - x=0 : u=0 (fixed)
    - $x=x_{max}$: F (axial force)
- Truss: NO bending deformation by G-force

Program: 1d.c (6/7)
Boundary Conditions

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

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

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

/* X=Xmax */
        i=N-1;
        Rhs[i]= F;
```

$u_1=0$

Diagonal Component=1
RHS=0
Off-Diagonal Components= 0.



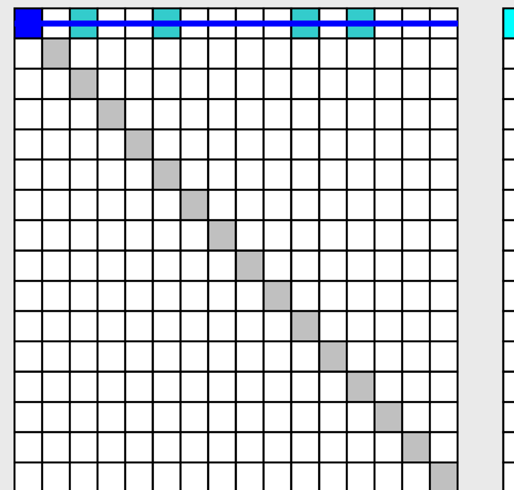
Program: 1d.c (6/7)
Boundary Conditions

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

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

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

/* X=Xmax */
        i=N-1;
        Rhs[i]= F;
```

$u_1=0$

Diagonal Component=1
RHS=0
Off-Diagonal Components= 0.

Erase !



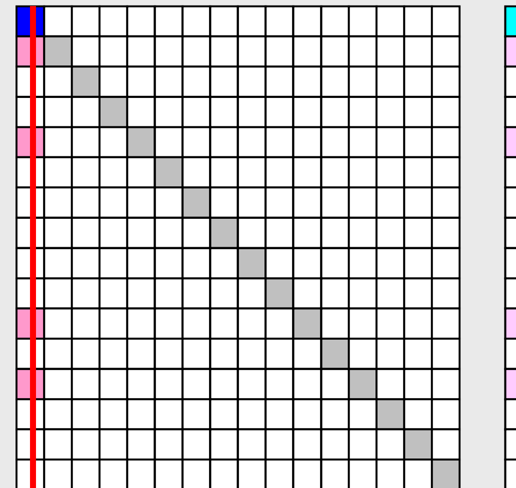
Program: 1d.c (6/7)
Boundary Conditions

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

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

          for(k=0;k<NPLU;k++){
            if(Item[k]==0){AMat[k]=0.0;
            }}
/* X=Xmax */
        i=N-1;
        Rhs[i]= F;
```

$u_1=0$

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)

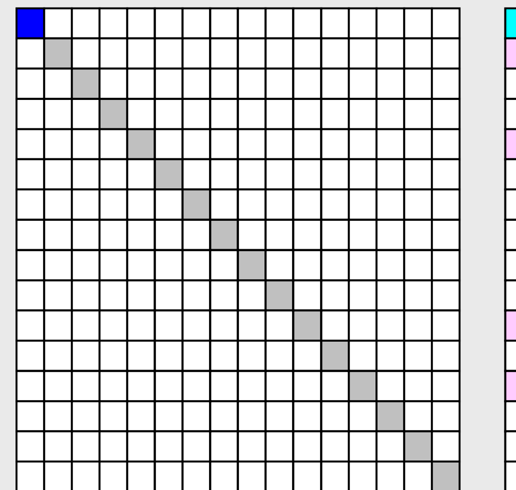
Program: 1d.c (6/7)
Boundary Conditions

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

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

          for(k=0;k<NPLU;k++){
            if(Item[k]==0){AMat[k]=0.0;
            }}
/* X=Xmax */
        i=N-1;
        Rhs[i]= F;
```

$u_1=0$

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 $u_1 \neq 0$

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

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

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

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

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

if $u_1 \neq 0$

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

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

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

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

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

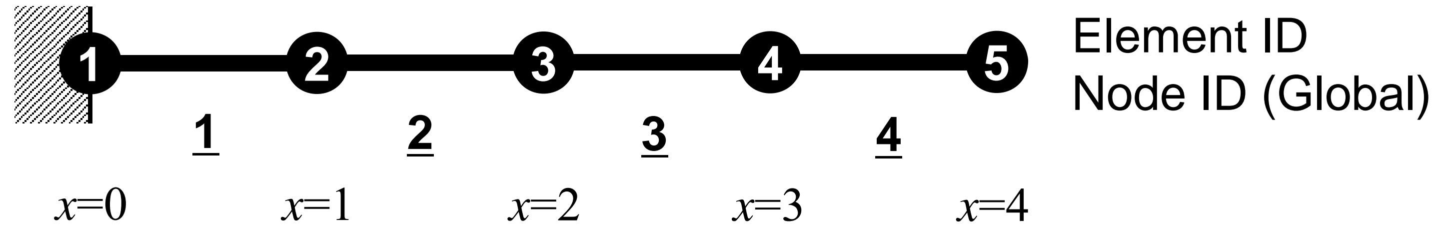
$$= Rhs_j - Amat_{k_s} u_{Item[k_s]} = Rhs_j - Amat_{k_s} u_{\min} \quad where\ Item[k_s]=0$$

Program: 1d.c (6/7)

Boundary Conditions

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

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

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

/* X=Xmax */
        i= N-1;
        Rhs[i]= F;
```

Element Eqn's/Accumulation



- As for Element-4:

$$[k]^{(4)} = \frac{EA}{L}\begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \qquad \{f\}^{(4)} = \int_S \bar{\sigma}[N]^T dS = \bar{\sigma}A\begin{Bmatrix} 0 \\ 1 \end{Bmatrix} = \begin{Bmatrix} 0 \\ F \end{Bmatrix}$$

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

+ + +

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

+ + +

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)

```
/*
// +---------------+
// | CG iterations |
// +---------------+
*/
        R = 0;
        Z = 1;
        Q = 1;
        P = 2;
        DD= 3;

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

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

CG Solver (1/6)

```
/*
// +----------------+
// | CG iterations |
// +----------------+
*/
        R = 0;
        Z = 1;
        Q = 1;
        P = 2;
        DD= 3;

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

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

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

CG Solver (2/6)

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

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

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)

```
  for(iter=1;iter<=IterMax;iter++){

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

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

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

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

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

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

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

  Alpha = Rho / C1;

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

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

```
  DNorm2 = 0.0;
  for(i=0;i<N;i++){
      DNorm2 += W[R][i] * W[R][i];
  }

  Resid = sqrt(DNorm2/BNorm2);

  if((iter)%1000 == 0){
      printf("%8d%s%16.6e¥n",iter, "
              iters, RESID=", Resid);
   }

   if(Resid <= Eps){ierr = 0; break;}

   Rho1 = Rho;
}
```

ρ_{i-2}

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

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>

```
input.dat

4                        NE
1.0  1.0  1.0  1.0       Δx, F, A, E
100                      Max Iterations
1.e-8                    Eps
```

Program: 1d.c (7/7)

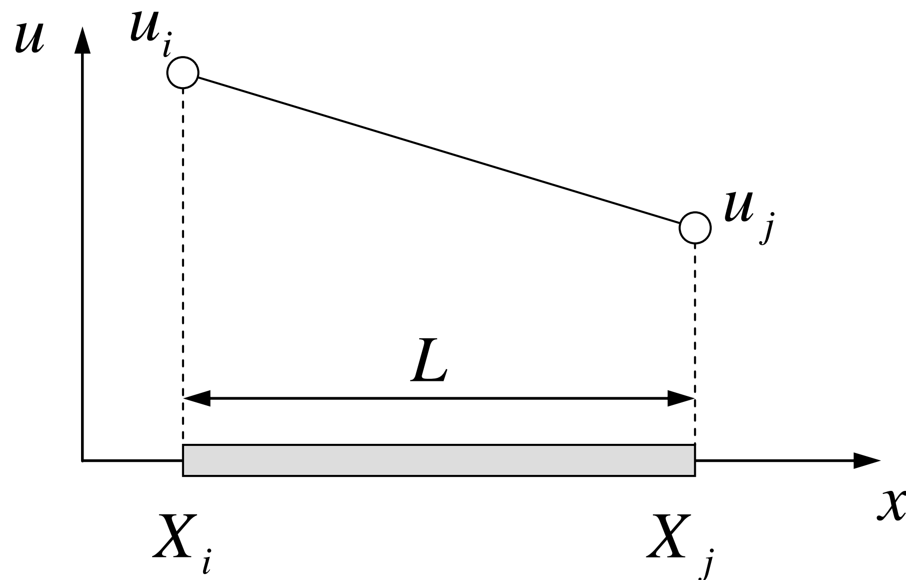
Element Stress

```
/*
// +-----------------+
// | STRESS recovery |
// +-----------------+
*/
        printf("¥n%s¥n", "### STRESS");

        for(icel=0;icel<NE;icel++){
             in1= Icelnod[2*icel];
             in2= Icelnod[2*icel+1];
             X1 = X[in1];
             X2 = X[in2];
             U1 = U[in1];
             U2 = U[in2];
             DL = fabs(X2-X1);

          Strain= (U2-U1)/DL;
          Sigma = Young*Strain;

          printf("%8d%16.6E%16.6E¥n", icel+1, Sigma, F/Area);

          }
```

Element Stress

Constant for 1D linear element: Constant Strain

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



$$u = N_i u_i + N_j u_j$$

$$\varepsilon = \frac{du}{dx} = \frac{d}{dx}\left(N_i u_i + N_j u_j\right)$$

$$= \frac{dN_i}{dx} u_i + \frac{dN_j}{dx} u_j$$

$$= \left(-\frac{1}{L}\right) u_i + \left(\frac{1}{L}\right) u_j = \boxed{\frac{u_j - u_i}{L}}$$

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
- Calculation of Stress