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

Kengo Nakajima
Information Technology Center

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

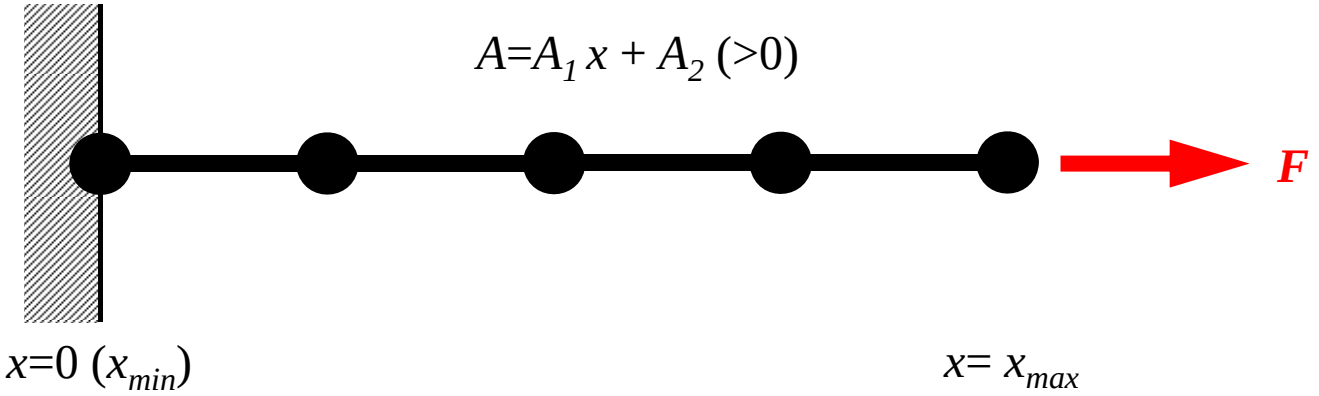
- Higher-order Elements
- Numerical Integration, Isoparametric Elements
- Report #1

- Non-Uniform Sectional Area
  - <$fem1>/1darea/a1.c
- 2nd-Order/Quadratic Element
  - <$fem1>/1d/1d2.c

1D Static Linear Elastic Problem

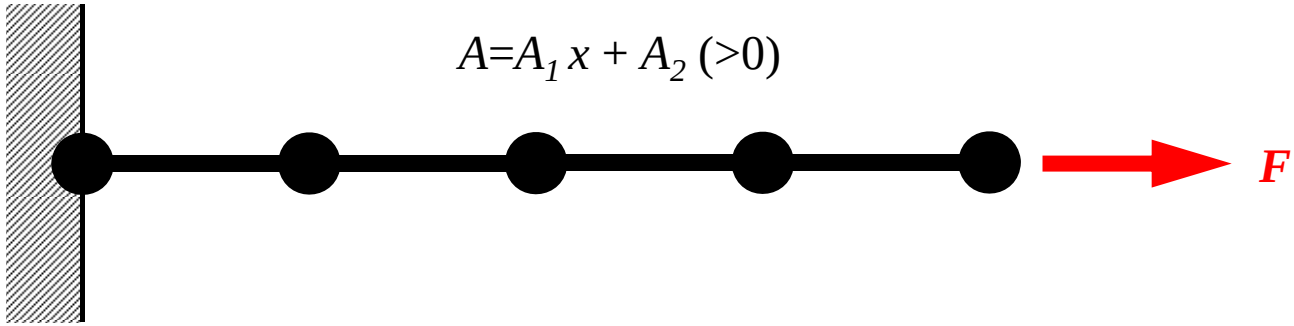
- Only deforms in x-direction (displacement: u)
  - Sectional Area $A=A_1 x + A_2$ (>0)
  - Uniform Young's Modulus E
  - Boundary Conditions (B.C.)
    - $x=0$: $u=0$ (fixed)
    - $x=x_{max}$: F (axial force)
- Truss: NO bending deformation by G-force

1D Static Linear Elastic Problem



- Only deforms in x-direction (displacement: u)
  - Sectional Area $A=A_1 x + A_2$ (>0)
  - Uniform 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

$$\sigma_x=E\varepsilon_x$$

Analytical Solution

$$\sigma_x = E\varepsilon_x = \frac{F}{A}$$

$$E\frac{du}{dx} = \frac{F}{A_1 x + A_2}$$

$$Eu = \frac{F}{A_1}\log\left(A_1 x + A_2\right) + C \qquad C = -\frac{F}{A_1}\log\left(A_2\right) \quad \because u = 0 @ x = 0$$

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

Files of Programs

1D Code for Static Linear-Elastic Problems

```
>$ cd <$fem1>
>$ cp /home03/skengon/Documents/class/fem1/1d2.tar .
>$ tar xvf 1d2.tar
>$ cd 1darea
```

Compile & GO !

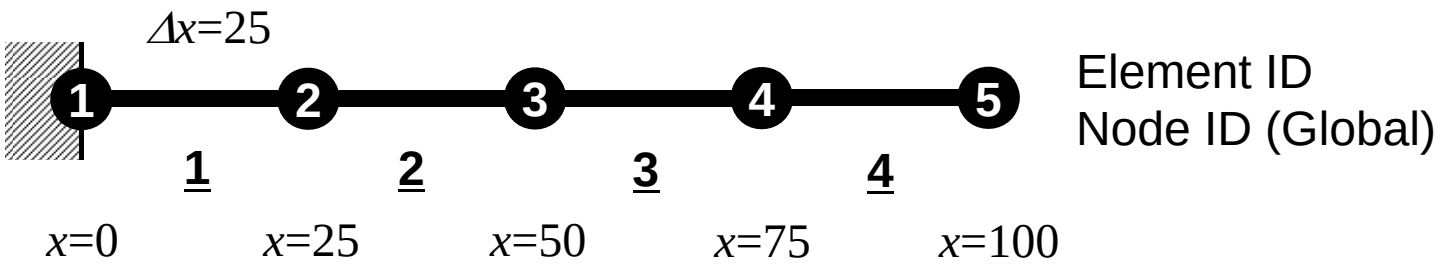
```
>$ cd <$fem1>/1darea
>$ cc –O a1.c      (or g95 –O a1.f)
>$ ./a.out
```

input.dat

```
4
25.0   5.e4   -0.105 12   5.e6
100
1.e-8
```

NE (Number of Elements)
Δx, **F**, **A_1**, **A_2**, **E**
Number of MAX. Iterations for CG Solver
Convergence Criteria for CG Solver

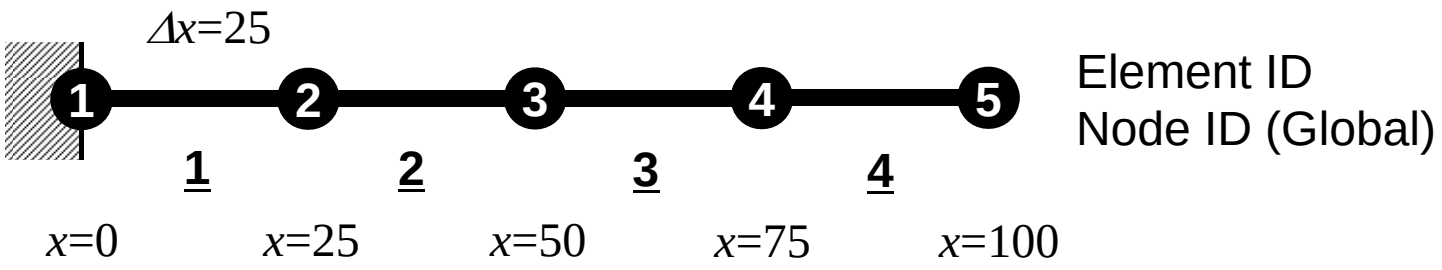
$$x = 0 \quad A_1 x + A_2 = 12.$$
$$x = 100 \quad A_1 x + A_2 = 1.5$$



FEM results differs from analytical ones.

```
>$ ./a.out

  4 iters, RESID=    2.079723E-16 U(N)=     1.892655E-01

### DISPLACEMENT
       1    0.000000E+00   -0.000000E+00
       2    2.339181E-02    2.351048E-02
       3    5.439956E-02    5.479659E-02
       4    1.003766E-01    1.016991E-01
       5    1.892655E-01    1.980421E-01
```

FEM **Analytical**



Program: a1.c (1/2)
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, A1, A2, F, Young;
        double X1, X2, U1, U2, DL, Strain, Sigma, Ck, XX, Disp;
        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, A1, A2	R		I	Sectional Area of Element Area= A1*x + A2
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

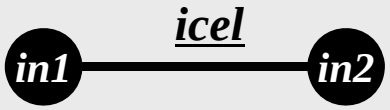
Variable/Arrays (2/2)

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: a1.c (2/2)

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);

          XX = 0.5 * (X1+X2);
          Area= A1*XX + A2;

          if(Area<= 0.) {
                fprintf(stderr, "ERROR: Area<0: ¥n");
                return -1;
          }

          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];
```



Integration on Each Element: $[k]$

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

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

$$= E\int_{X_i}^{X_j}\begin{bmatrix}-1/L\\ 1/L\end{bmatrix}\begin{bmatrix}-1/L, 1/L\end{bmatrix} A\,dx$$

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

$$= \frac{E}{L^2}\int_{X_i}^{X_j}\begin{bmatrix}+1 & -1\\ -1 & +1\end{bmatrix} A dx = \frac{E}{L^2}\begin{bmatrix}+1 & -1\\ -1 & +1\end{bmatrix}\int_{X_i}^{X_j}\left(A_1 x + A_2\right)dx$$

$$= \frac{E}{L^2}\begin{bmatrix}+1 & -1\\ -1 & +1\end{bmatrix}\left[\frac{1}{2}A_1 x^2 + A_2 x\right]_{X_i}^{X_j}$$

$$= \frac{E\left(\frac{1}{2}A_1\left(X_j{}^2 - X_i{}^2\right) + A_2\left(X_j - X_i\right)\right)}{L^2}\begin{bmatrix}+1 & -1\\ -1 & +1\end{bmatrix}$$

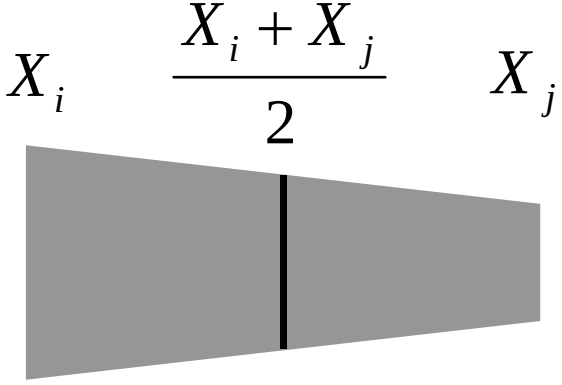
Sectional Area@X=L/2 is multiplied

$$\frac{E\left(\frac{1}{2}A_1\left(X_j{}^2 - X_i{}^2\right) + A_2\left(X_j - X_i\right)\right)}{L^2}\begin{bmatrix}+1 & -1\\ -1 & +1\end{bmatrix}$$

$$= \frac{E\left(\frac{1}{2}A_1\left(X_i + X_j\right)\left(X_j - X_i\right) + A_2\left(X_j - X_i\right)\right)}{L^2}\begin{bmatrix}+1 & -1\\ -1 & +1\end{bmatrix}$$

$$X_j - X_i = L$$

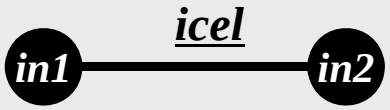
$$= \frac{E\left(A_1\frac{\left(X_i + X_j\right)}{2} + A_2\right)}{L}\begin{bmatrix}+1 & -1\\ -1 & +1\end{bmatrix}$$



Program: a1.c (2/2)

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);

          XX = 0.5 * (X1+X2);
          Area= A1*XX + A2;

          if(Area<= 0.) {
                fprintf(stderr, "ERROR: Area<0: ¥n");
                return -1;
          }

          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];
```



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

$$= \frac{E \cdot A(XX)}{L} [Kmat]$$

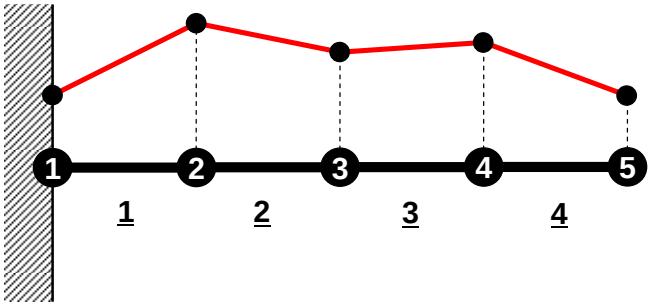
- Non-Uniform Sectional Area
  - <$fem1>/1darea/a1.c
- 2nd-Order/Quadratic Element
  - <$fem1>/1d/1d2.c

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
```

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



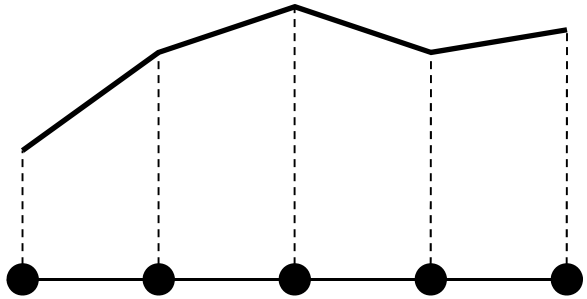

```
NE=20, dX=5

 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）

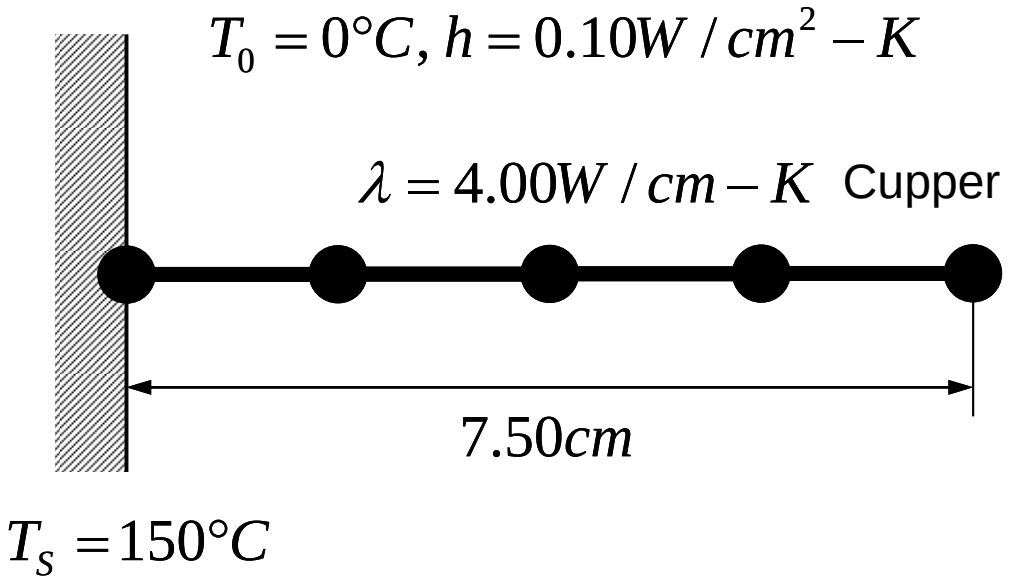
Remedies for Higher Accuracy

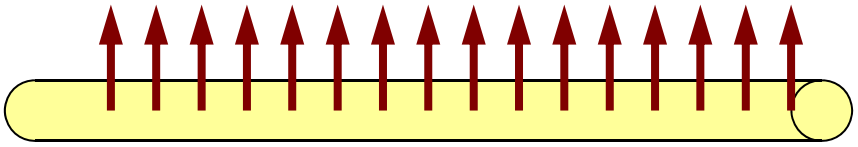
- Finer Meshes
- Higher Order Shape/Interpolation Function
- Formulation which assures continuity of n-th order derivatives: C^n Continuity
- Linear Elements
  - Piecewise Linear
  - C^0 Continuity
    - Only dependent variables are continuous at element boundary
- In this section …
- Second Order/Quadratic Elem's
  - Suitable for discretization of curves
  - C^0 Continuity



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

Example: 1D Heat Transfer (1/2)





Convective Heat Transfer on Cylindrical Surface

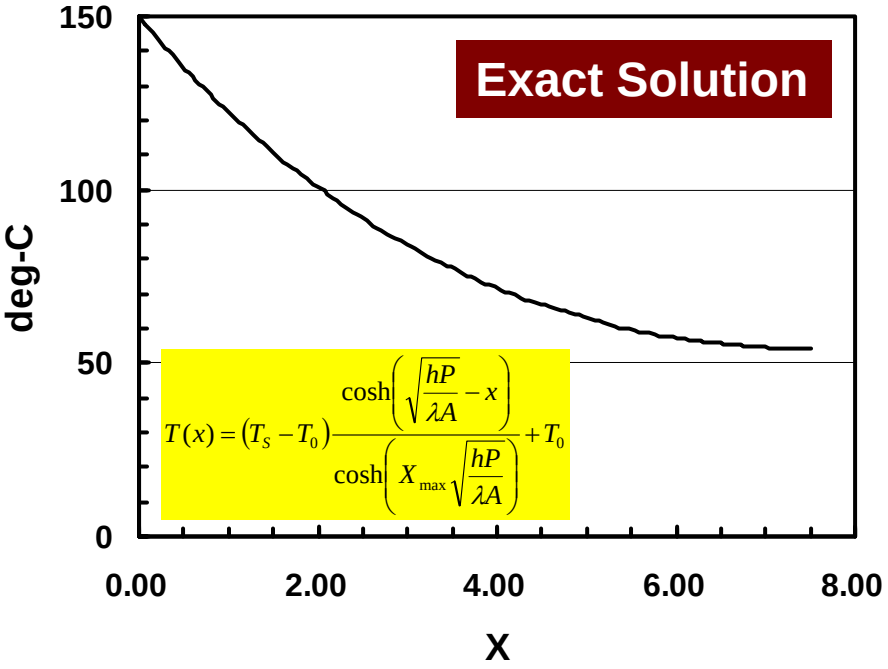
- Temp. Thermal Fins
- Circular Sectional Area, r=1cm
- Boundary Condition
  - x=0 : Fixed Temperature
  - x=7.5 : Insulated
- Convective Heat Transfer on Cylindrical Surface
  - $q = h\,(T-T_0)$
  - q : Heat Flux
    - Heat Flow/Unit Surface Area/sec.

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.

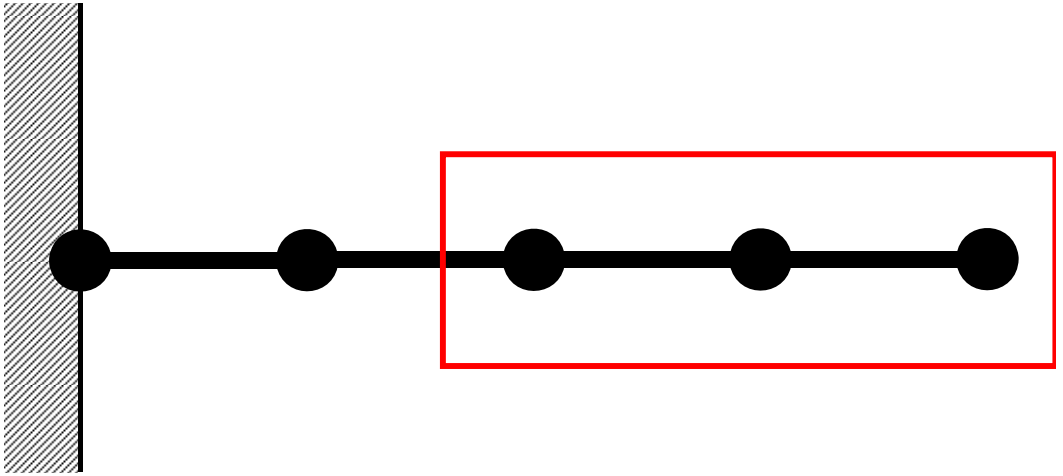
Let's go back to the initial problem with constant A

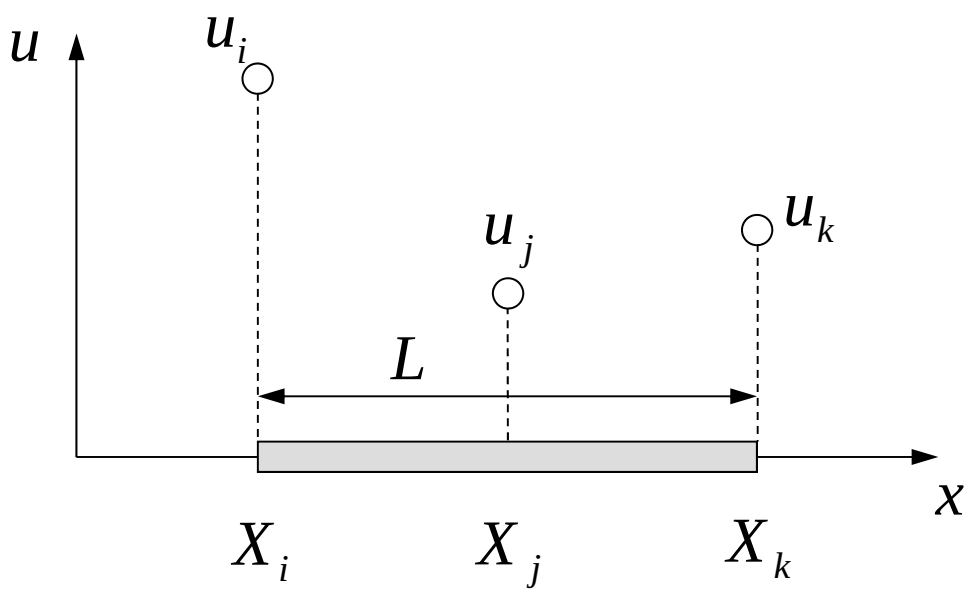
- Only deforms in x-direction (displacement: u)
  - Uniform: Sectional Area A, Young's Modulus E
  - Boundary Conditions (B.C.)
    - $x=0$: $u=0$ (fixed)
    - $x=x_{max}$: F (axial force)
- Truss: NO bending deformation by G-force

1D Quadratic Element (1/2)
一次元二次要素

- Length: L
- Three Nodes (i,j,k)
  - j : midpoint of i and k
  - j : intermediate node
    （中間節点）
- Displacement u is defined over the element, as follows:

$$u = \alpha_1 + \alpha_2 x + \alpha_3 x^2$$





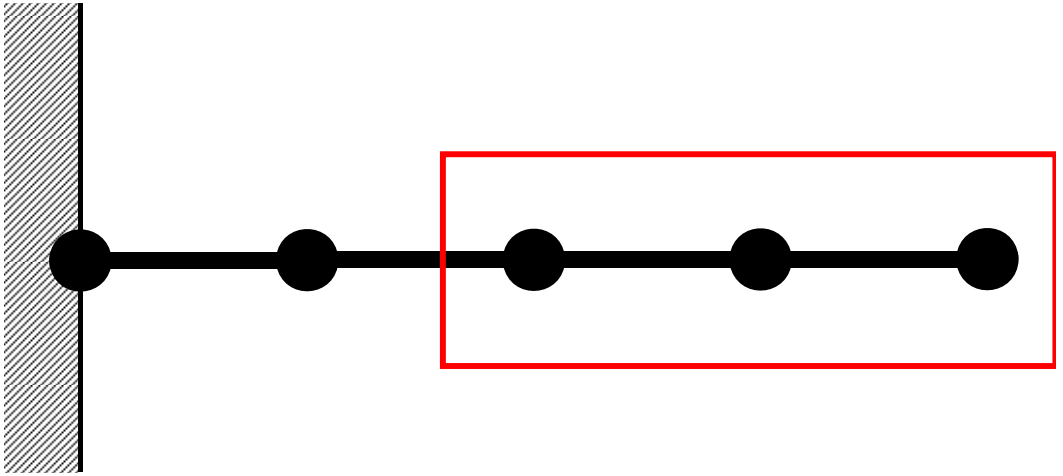
1D Quadratic Element (1/2)

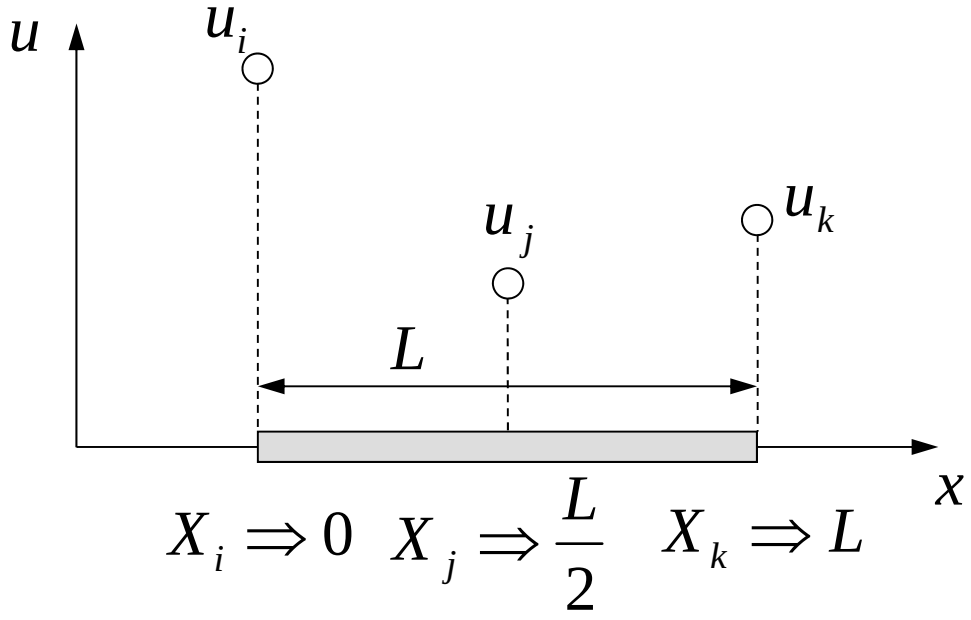
- Length: L
- Three Nodes (i,j,k)
  - j : midpoint of i and k
  - Local Coordinate System where X_i=0
- Displacement u is defined over the element, as follows:

$$u = \alpha_1 + \alpha_2 x + \alpha_3 x^2$$

$$u_i = \alpha_1, \quad u_j = \alpha_1 + \frac{L}{2}\alpha_2 + \frac{L^2}{4}\alpha_3$$

$$u_k = \alpha_1 + L\alpha_2 + L^2\alpha_3$$



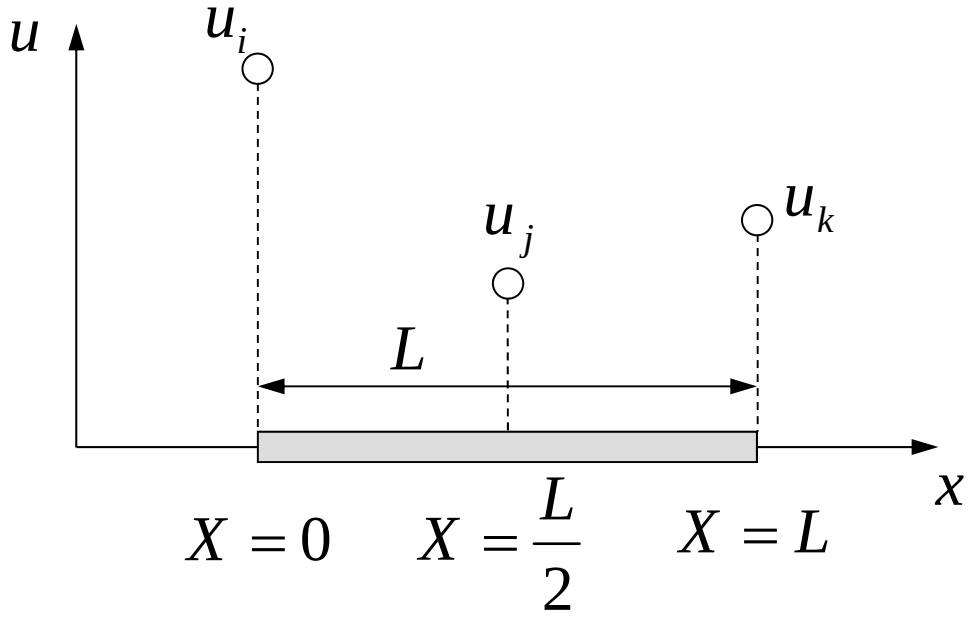


1D Quadratic Element (2/2)

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

$$\alpha_1 = u_i,\ \alpha_2 = \frac{4u_i - 3u_j - u_k}{L},$$

$$\alpha_3 = \frac{2}{L^2}\left(u_i - 2u_j + u_k\right)$$



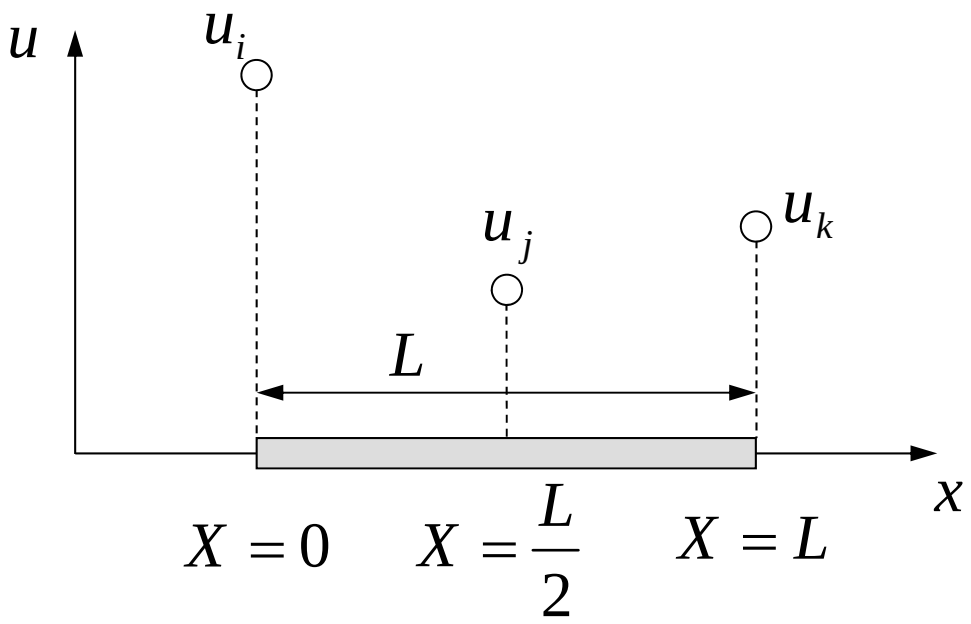
- Shape Functions are as follows:

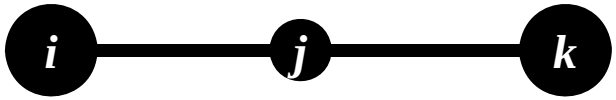
$$u = N_i u_i + N_j u_j + N_k u_k$$

$$= \left(1 - \frac{2x}{L}\right)\left(1 - \frac{x}{L}\right)u_i + \left(\frac{4x}{L}\right)\left(1 - \frac{x}{L}\right)u_j + \left(-\frac{x}{L}\right)\left(1 - \frac{2x}{L}\right)u_k$$

Confirm values at X=0, X=L/2 & X=L.

1D Quadratic Element



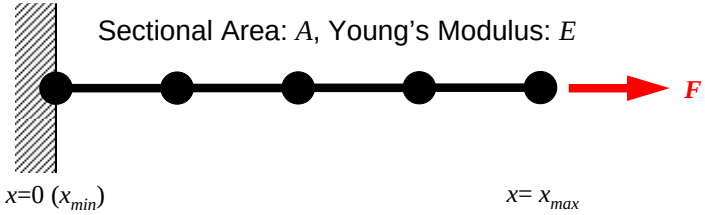


Intermediate Node
（中間節点）

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

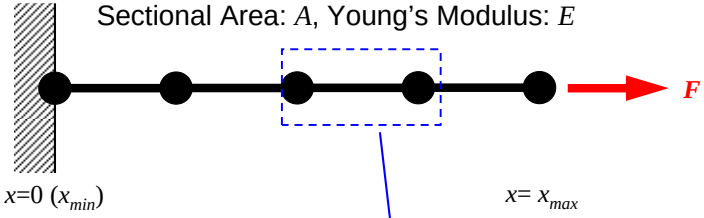
- Following integral equation is obtained at each element by Galerkin method, where $[N]$'s are also weighting functions:

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

Galerkin Method (2/4)

- Green's Theorem (1D)

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



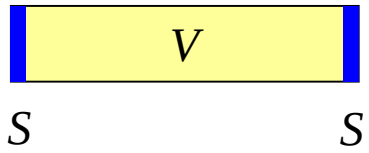
- Apply this to the 1st part of eqn with 2nd-order diff.:

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

- Consider the following terms:

$$u = [N]\{\phi\} \qquad \bar{\sigma} = E\frac{du}{dx}$$

Stress on Surfaces of Each Element

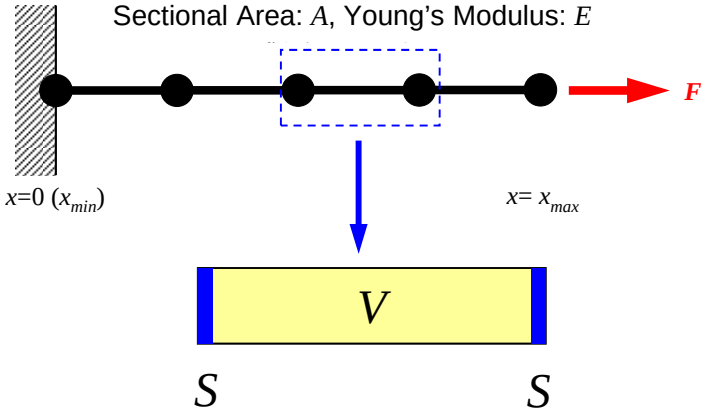


Galerkin Method (3/4)

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

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

$$+\int_S \bar{\sigma}[N]^T dS+\int_V X[N]^T dV=0$$

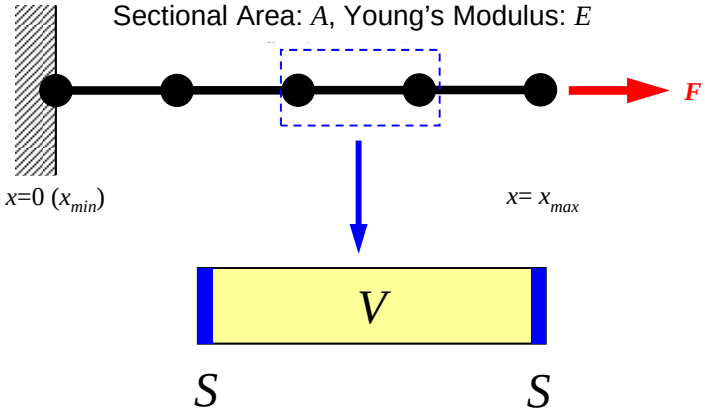


- This is called "weak form". Original PDE consists of terms with 2nd-order diff., but this "weak form" only includes 1st-order diff by Green's theorem.
  - Requirements for shape functions are "weaker" in "weak form". Linear functions can describe effects of 2nd-order differentiation.

Galerkin Method (4/4)

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

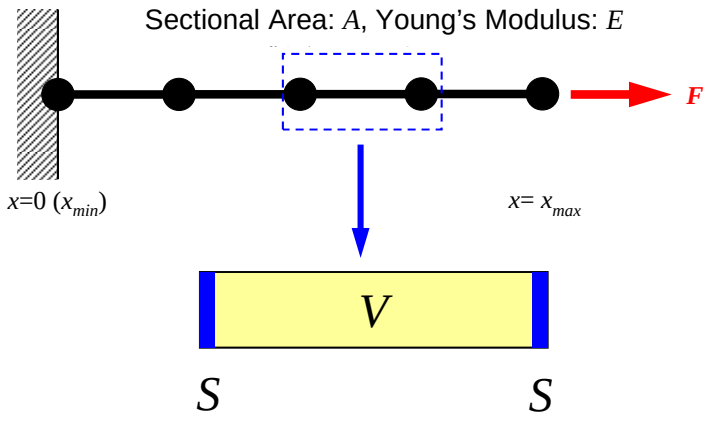
$$+\int_S \bar{\sigma}[N]^T dS+\int_V X[N]^T dV=0$$



- 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)
  - Essential B.C.



- Derivatives of Unknowns (Neumann)
  - Naturally satisfied in weak form
  - Secondary B.C.
  - Natural B.C

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

$$+\int_S \bar{\sigma}[N]^T dS+\int_V X[N]^T dV=0$$

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

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

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

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

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

So far, nothing has changed from piecewise linear case.

Integration over Each Element: $[k]$ (1/2)

$$N_i = \left(1-\frac{2x}{L}\right)\left(1-\frac{x}{L}\right) \qquad \frac{dN_i}{dx} = \left(\frac{4x}{L^2}-\frac{3}{L}\right)$$

$$N_j = \left(\frac{4x}{L}\right)\left(1-\frac{x}{L}\right) \qquad \frac{dN_j}{dx} = \left(\frac{4}{L}-\frac{8x}{L^2}\right)$$

$$N_k = \left(-\frac{x}{L}\right)\left(1-\frac{2x}{L}\right) \qquad \frac{dN_k}{dx} = \left(\frac{4x}{L^2}-\frac{1}{L}\right)$$

Derivative:
1st-order function of x
in each element.

Integration over Each Element: $[k]$ (2/2)

$$\int_V E\left(\frac{d[N]^T}{dx}\frac{d[N]}{dx}\right)dV = \int_0^L \begin{bmatrix} dN_i/dx \\ dN_j/dx \\ dN_k/dx \end{bmatrix} E\left[\frac{dN_i}{dx}, \frac{dN_j}{dx}, \frac{dN_k}{dx}\right] A\,dx$$

$$= EA\int_0^L \begin{bmatrix} \frac{dN_i}{dx}\frac{dN_i}{dx} & \frac{dN_i}{dx}\frac{dN_j}{dx} & \frac{dN_i}{dx}\frac{dN_k}{dx} \\ \frac{dN_j}{dx}\frac{dN_i}{dx} & \frac{dN_j}{dx}\frac{dN_j}{dx} & \frac{dN_j}{dx}\frac{dN_k}{dx} \\ \frac{dN_k}{dx}\frac{dN_i}{dx} & \frac{dN_k}{dx}\frac{dN_j}{dx} & \frac{dN_k}{dx}\frac{dN_k}{dx} \end{bmatrix} dx = \frac{EA}{6L}\begin{bmatrix} +14 & -16 & +2 \\ -16 & +32 & -16 \\ +2 & -16 & +14 \end{bmatrix}$$

Integration over Each Element: $\{f\}$

$$\int_V X\left[N\right]^T dV = XA\int_0^L \begin{bmatrix} N_i \\ N_j \\ N_k \end{bmatrix} dx = XA\int_0^L \begin{bmatrix} 1-\frac{3x}{L}+\frac{2x^2}{L^2} \\ \frac{4x}{L}-\frac{4x^2}{L^2} \\ -\frac{x}{L}+\frac{2x^2}{L^2} \end{bmatrix} dx = \frac{XAL}{6}\begin{Bmatrix} 1 \\ 4 \\ 1 \end{Bmatrix}$$

1 : 4 : 1

Body Force

Piecewise Linear Case: Ratio was 1:1

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

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

Body Force

A: Sectional Area

L: Length

Integration over Each Element: $\{f\}$

$$\int_V X[N]^T dV = XA\int_0^L \begin{bmatrix} N_i \\ N_j \\ N_k \end{bmatrix} dx = XA\int_0^L \begin{bmatrix} 1-\frac{3x}{L}+\frac{2x^2}{L^2} \\ \frac{4x}{L}-\frac{4x^2}{L^2} \\ -\frac{x}{L}+\frac{2x^2}{L^2} \end{bmatrix} dx = \frac{XAL}{6}\begin{Bmatrix} 1 \\ 4 \\ 1 \end{Bmatrix}$$

1 : 4 : 1

Body Force

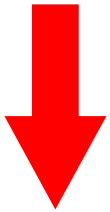
$$\int_V \bar{\sigma}[N]^T dS = \bar{\sigma}A\big|_{x=L} = \bar{\sigma}A\begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix}$$

Surface Force

Global Equations

- Accumulate Element Equations:

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



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

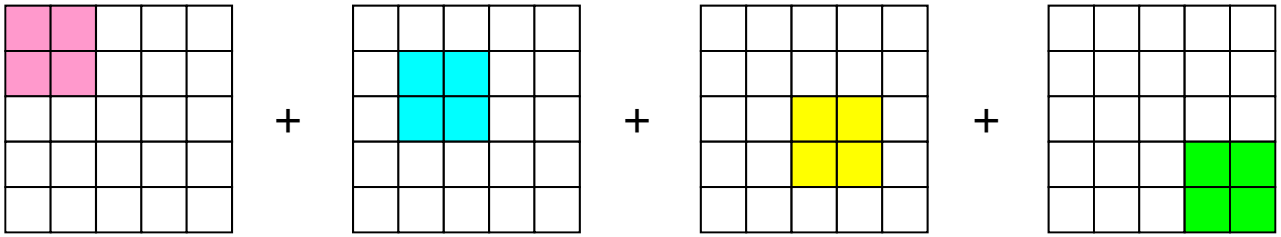
$$[K] = \sum [k], \quad \{F\} = \sum \{f\}$$

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

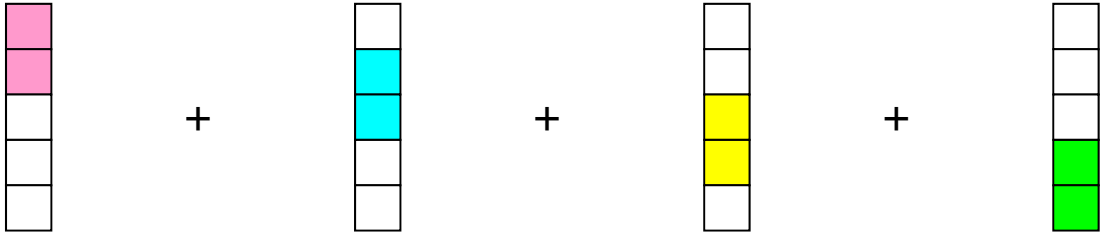
This is the final linear equations (global equations) to be solved.

Element Eqn's/Accumulation Piecewise Linear Element

$$[K]=\sum_{i=1}^{4}\left[k^{(i)}\right]=$$

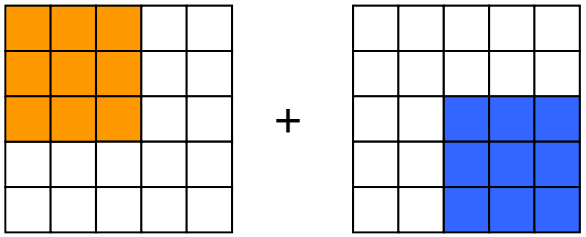


$$\{F\}=\sum_{i=1}^{4}\left\{f^{(i)}\right\}=$$

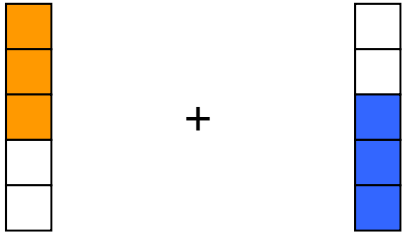


Element Eqn's/Accumulation Quadratic Interpolation, 2-elements

$$[K] = \sum_{i=1}^{2} [k^{(i)}] =$$



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



Compile & GO !

```
>$ cd <$fem1>/1d
>$ cc –O 1d2.c       (or g95 –O 1d2.f)
>$ ./a.out
```

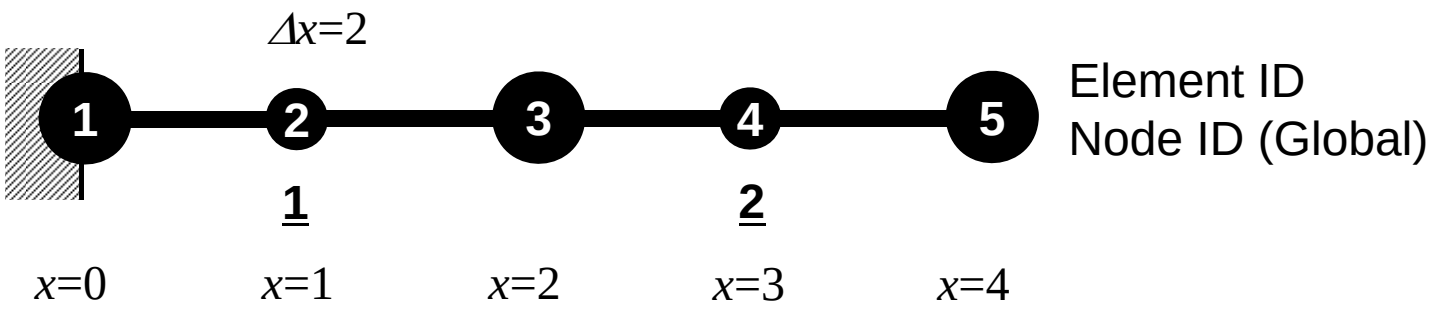
input2.dat

```
2
2.0  1.0  1.0  1.0
100
1.e-8
```

NE (Number of Elements)
Δx (Length of Each Elem.: **L**) , **F, A, E**
Number of MAX. Iterations for CG Solver
Convergence Criteria for CG Solver

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

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

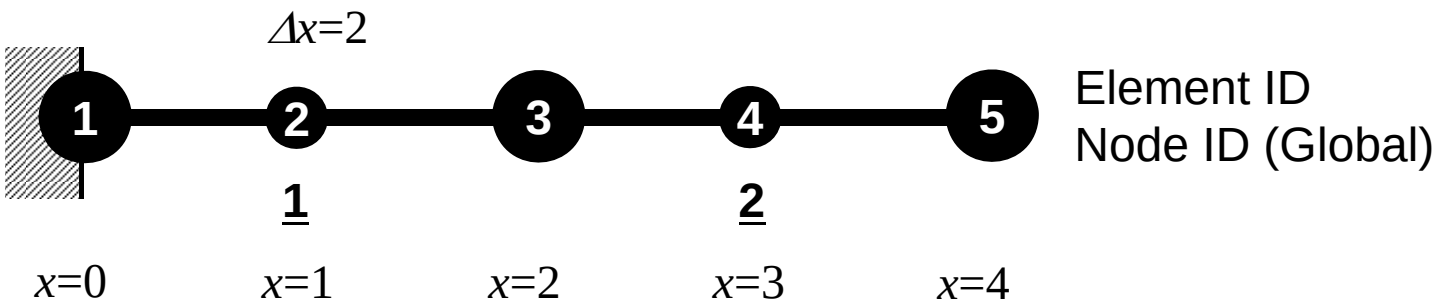


Results

```
>$ ./a.out

  4 iters, RESID=    1.914442e-15

### DISPLACEMENT
       1    0.000000E+00    0.000000E+00
       2    1.000000E+00    1.000000E+00
       3    2.000000E+00    2.000000E+00
       4    3.000000E+00    3.000000E+00
       5    4.000000E+00    4.000000E+00

                  FEM          Exact Solution
```

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

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



Program: 1d2.c (1/7)
variables and arrays

```
/*
//   1D Solid Mechanics for Truss Elements solved by
//   CG (Conjugate Gradient) Method using 2nd-order Elements
//
//   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, NPLU0;
        int R, Z, Q, P, DD;

        double dX, Resid, Eps, Area, F, Young;
        double X1, X2, X3, U1, U2, U3, DL, Strain, Sigma, Ck;
        double *U, *Rhs, *X;
        double *Diag, *AMat;
        double **W;

        int *Index, *Item, *IceInod;
        double Kmat[3][3], Emat[3][3];

        int i, j, in1, in2, in3, k, icel, k12, k13, k21, k23, k31, k32, 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	3*NE	O	Node ID for Each Element
Kmat	R	[3][3]	O	Element Matrix [*k*]
Emat	R	[3][3]	O	Element Matrix

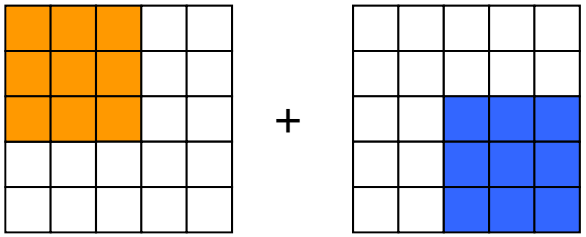
Program: 1d2.c (2/7)

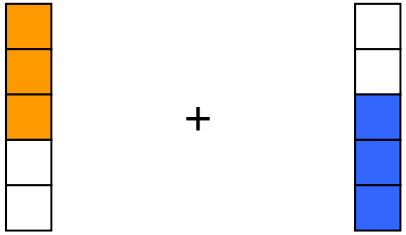
Initialization, Allocation of Arrays

```
/*
// +-------+
// | INIT. |
// +-------+
*/
        fp = fopen("input2.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    = 2*NE + 1;
        NPLU0= 2*2 + NE*2 + (NE-1)*4;

        U    = calloc(N, sizeof(double));
        X    = calloc(N, sizeof(double));
        Diag = calloc(N, sizeof(double));
        AMat = calloc(NPLU0, sizeof(double));
        Rhs = calloc(N, sizeof(double));
        Index= calloc(N+1, sizeof(int));
        Item = calloc(NPLU0, sizeof(int));
        IcelNod= calloc(3*NE, sizeof(int));
```

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

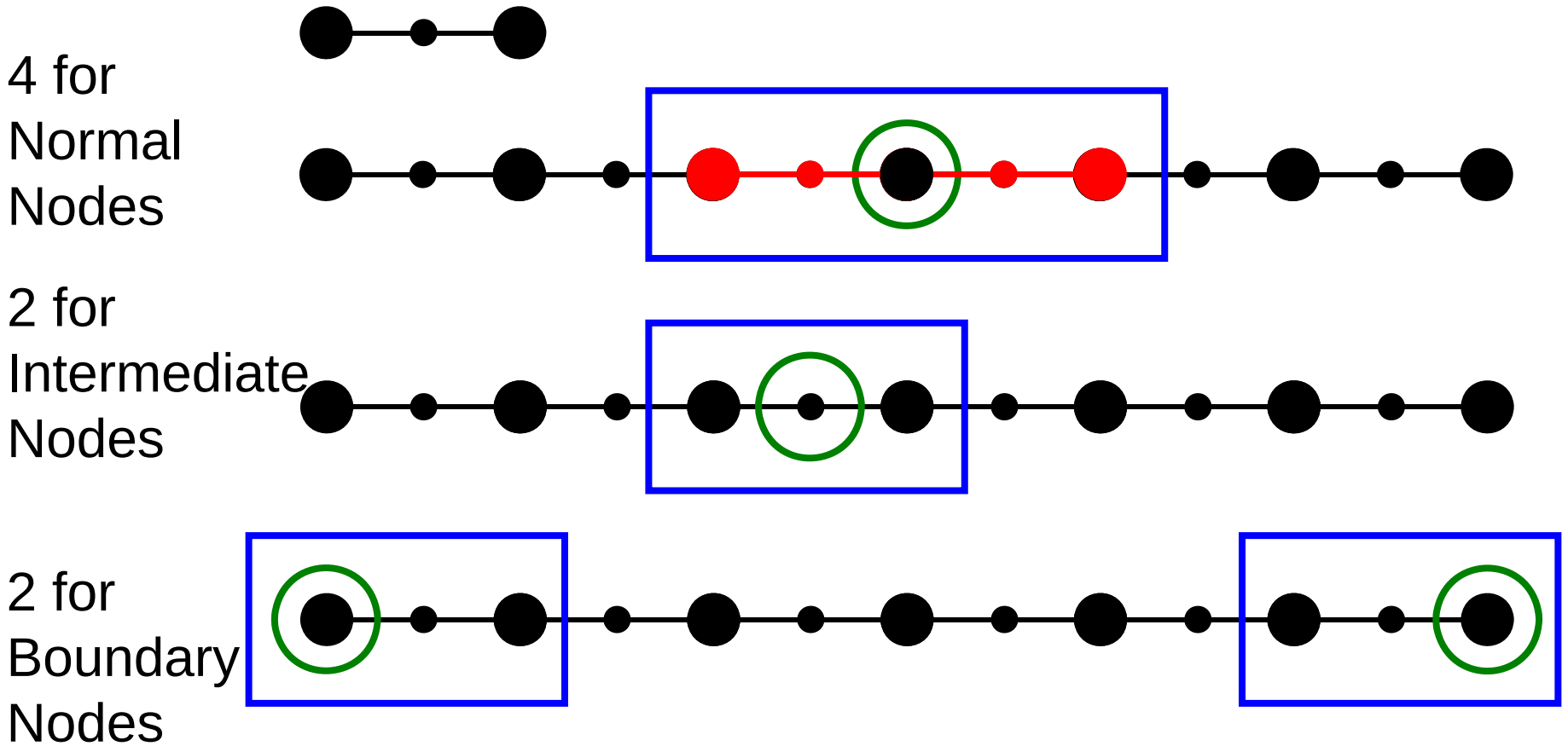
Non-Zero Off-Diagonal Components Nodes are connected to node on each element each other

$$[K]=\sum_{i=1}^{2}\left[k^{(i)}\right]=$$  +

$$\{F\}=\sum_{i=1}^{4}\left\{f^{(i)}\right\}=$$  +

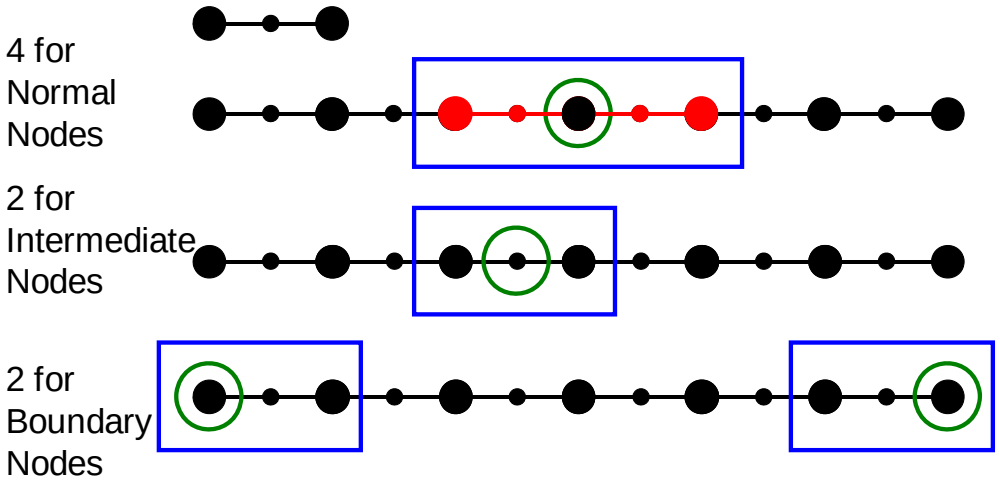
Number of Non-Zero Off-Diagonals

Different number of connections according to location of each node



Number of Non-Zero Off-Diagonals

Different number of connections according to location of each node



- Boundary Node# 2
- Intermediate Node# NE
- Normal Node# NE+1-2= NE-1
  - except Boundary Nodes
- Total Node#: N= 2+NE+NE-1= 2*NE + 1
- Non-Zero Off-Diag.#: NPLU= 2*2 + 2*NE + 4*(NE-1)

Program: 1d2.c (3/7)

Initialization, Allocation of Arrays (cont.)

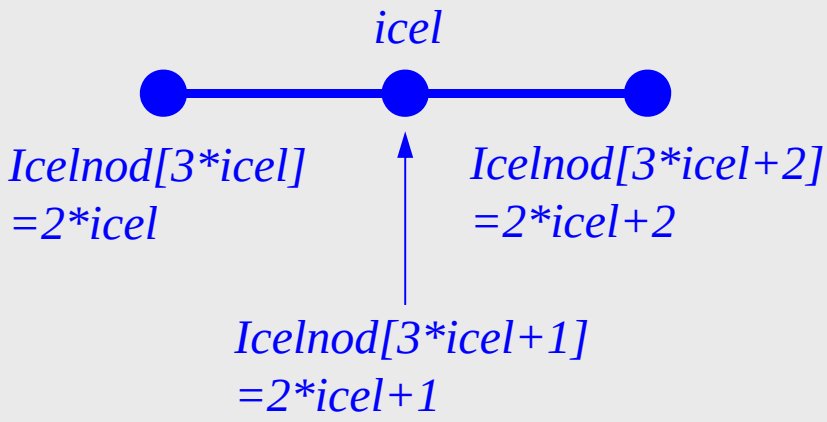
```
 W = (double **)malloc(sizeof(double *)*4);
if(W == NULL) {
        fprintf(stderr, "Error: %s¥n", strerror(errno));
        return -1;
}
for(i=0; i<4; i++) {
        W[i] = (double *)malloc(sizeof(double)*N);
        if(W[i] == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                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<NPLU0;k++)  AMat[k] = 0.0;
for(i=0;i<N;i++) X[i]= i*dX*0.5;
for(icel=0;icel<NE;icel++){
        Icelnod[3*icel  ]= 2*icel;
        Icelnod[3*icel+1]= 2*icel+1;
        Icelnod[3*icel+2]= 2*icel+2;
}
Kmat[0][0]= +14.0/6.;
Kmat[0][1]= -16.0/6.;
Kmat[0][2]=  +2.0/6.;
Kmat[1][0]= -16.0/6.;
Kmat[1][1]= +32.0/6.;
Kmat[1][2]= -16.0/6.;
Kmat[2][0]=  +2.0/6.;
Kmat[2][1]= -16.0/6.;
Kmat[2][2]= +14.0/6.;
```

X: X-coordinate component of each node

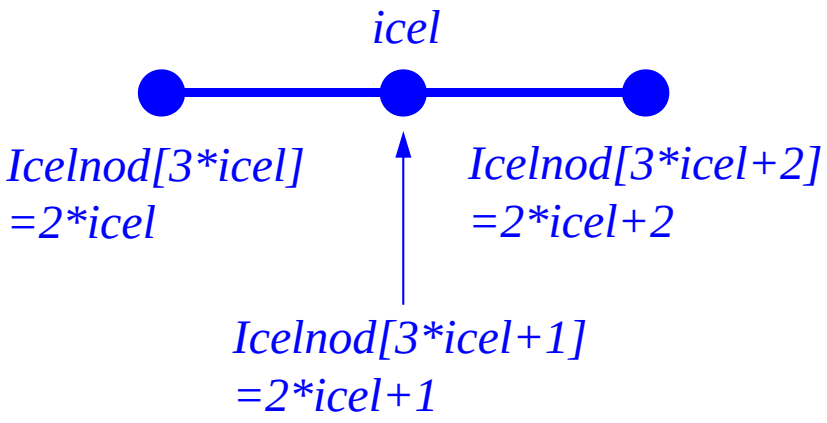
Program: 1d2.c (3/7)

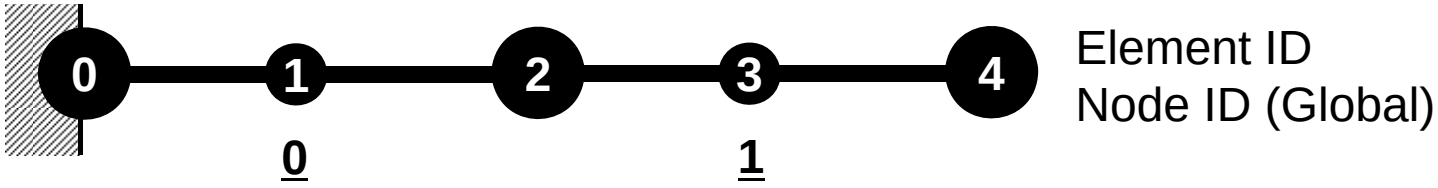
Initialization, Allocation of Arrays (cont.)

```
 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<NPLU0;k++)  AMat[k] = 0.0;
for(i=0;i<N;i++) X[i]= i*dX*0.5;
for(icel=0;icel<NE;icel++){
        Icelnod[3*icel  ]= 2*icel;
        Icelnod[3*icel+1]= 2*icel+1;
        Icelnod[3*icel+2]= 2*icel+2;
}
Kmat[0][0]= +14.0/6.;
Kmat[0][1]= -16.0/6.;
Kmat[0][2]=  +2.0/6.;
Kmat[1][0]= -16.0/6.;
Kmat[1][1]= +32.0/6.;
Kmat[1][2]= -16.0/6.;
Kmat[2][0]=  +2.0/6.;
Kmat[2][1]= -16.0/6.;
Kmat[2][2]= +14.0/6.;
```



Element ID, Local/Internal Node ID





Program: 1d2.c (3/7)

Initialization, Allocation of Arrays (cont.)

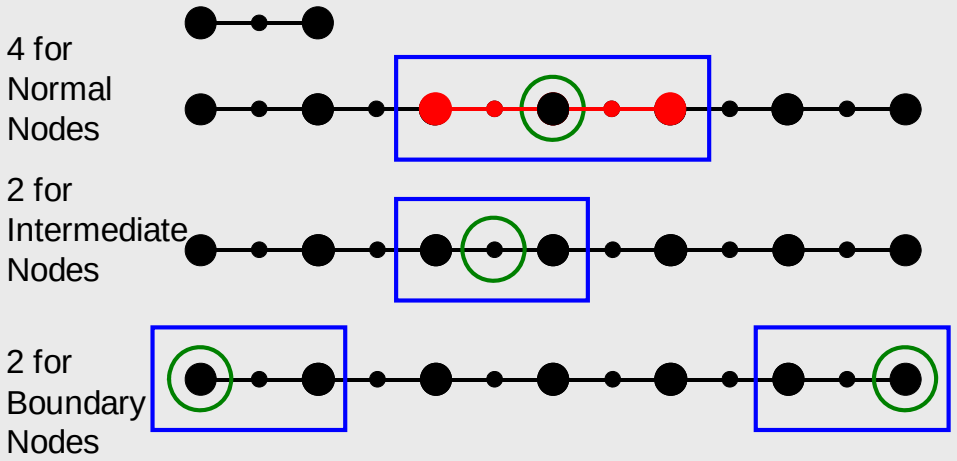
```
 W = (double **)malloc(sizeof(double *)*4);
if(W == NULL) {
        fprintf(stderr, "Error: %s¥n", strerror(errno));
        return -1;
}
for(i=0; i<4; i++) {
        W[i] = (double *)malloc(sizeof(double)*N);
        if(W[i] == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                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<NPLU0;k++)  AMat[k] = 0.0;
for(i=0;i<N;i++) X[i]= i*dX*0.5;
for(icel=0;icel<NE;icel++){
        Icelnod[3*icel  ]= 2*icel;
        Icelnod[3*icel+1]= 2*icel+1;
        Icelnod[3*icel+2]= 2*icel+2;
}
Kmat[0][0]= +14.0/6.;
Kmat[0][1]= -16.0/6.;
Kmat[0][2]=  +2.0/6.;
Kmat[1][0]= -16.0/6.;
Kmat[1][1]= +32.0/6.;
Kmat[1][2]= -16.0/6.;
Kmat[2][0]=  +2.0/6.;
Kmat[2][1]= -16.0/6.;
Kmat[2][2]= +14.0/6.;
```

$$[k]^{(e)} = \frac{EA}{L}\frac{1}{6}\begin{bmatrix} +14 & -16 & +2 \\ -16 & +32 & -16 \\ +2 & -16 & +14 \end{bmatrix}$$

Program: 1d2.c (4/7)

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

```
/*
// +--------------+
// | CONNECTIVITY |
// +--------------+
*/
        Index[0]= 0;
        for(i=1;i<N;i++) {
          if (i%2==1) {Index[i]=4;
          }else       {Index[i]=2;}}
        Index[1]= 2;
        Index[N]= 2;
        for(i=0;i<N;i++){
                Index[i+1]= Index[i+1] + Index[i];}

        NPLU= Index[N];
        for(i=0;i<N;i++){
                int jS = Index[i];
                if(i == 0){
                        Item[jS  ] = i+1;
                        Item[jS+1] = i+2;
                }else if(i == N-1){
                        Item[jS  ] = i-2;
                        Item[jS+1] = i-1;
                }else{
                  if (i%2==1){
                        Item[jS  ] = i-1;
                        Item[jS+1] = i+1;
                  } else       {
                        Item[jS  ] = i-2;
                        Item[jS+1] = i-1;
                        Item[jS+2] = i+1;
                        Item[jS+3] = i+2;}}}
```

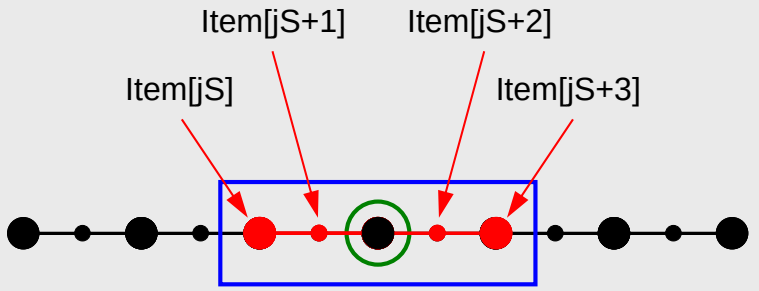


Program: 1d2.c (4/7)

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

```
/*
// +--------------+
// | CONNECTIVITY |
// +--------------+
*/
        Index[0]= 0;
        for(i=1;i<N;i++) {
          if (i%2==1) {Index[i]=4;
          }else       {Index[i]=2;}}
        Index[1]= 2;
        Index[N]= 2;
        for(i=0;i<N;i++){
                Index[i+1]= Index[i+1] + Index[i];}

        NPLU= Index[N];
        for(i=0;i<N;i++){
                int jS = Index[i];
                if(i == 0){
                        Item[jS  ] = i+1;
                        Item[jS+1] = i+2;
                }else if(i == N-1){
                        Item[jS  ] = i-2;
                        Item[jS+1] = i-1;
                }else{
                  if (i%2==1){
                        Item[jS  ] = i-1;
                        Item[jS+1] = i+1;
                  } else       {
                        Item[jS  ] = i-2;
                        Item[jS+1] = i-1;
                        Item[jS+2] = i+1;
                        Item[jS+3] = i+2;}}}
```

jS= index[i]

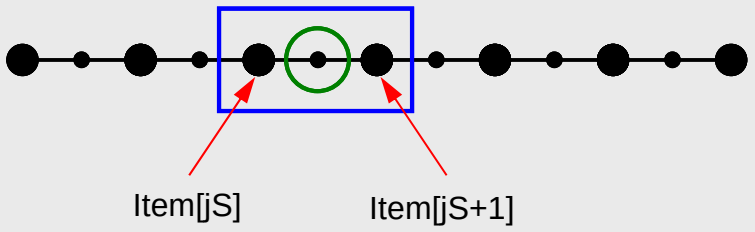


Program: 1d2.c (4/7)

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

```
/*
// +--------------+
// | CONNECTIVITY |
// +--------------+
*/
        Index[0]= 0;
        for(i=1;i<N;i++) {
          if (i%2==1) {Index[i]=4;
          }else       {Index[i]=2;}}
        Index[1]= 2;
        Index[N]= 2;
        for(i=0;i<N;i++){
                Index[i+1]= Index[i+1] + Index[i];}

        NPLU= Index[N];
        for(i=0;i<N;i++){
                int jS = Index[i];
                if(i == 0){
                        Item[jS  ] = i+1;
                        Item[jS+1] = i+2;
                }else if(i == N-1){
                        Item[jS  ] = i-2;
                        Item[jS+1] = i-1;
                }else{
                  if (i%2==1){
                        Item[jS  ] = i-1;
                        Item[jS+1] = i+1;
                  } else       {
                        Item[jS  ] = i-2;
                        Item[jS+1] = i-1;
                        Item[jS+2] = i+1;
                        Item[jS+3] = i+2;}}}
```

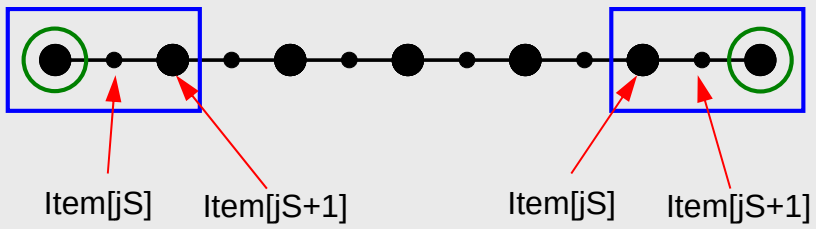
jS= index[i]



Program: 1d2.c (4/7)

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

```
/*
// +--------------+
// | CONNECTIVITY |
// +--------------+
*/
        Index[0]= 0;
        for(i=1;i<N;i++) {
          if (i%2==1) {Index[i]=4;
          }else       {Index[i]=2;}}
        Index[1]= 2;
        Index[N]= 2;
        for(i=0;i<N;i++){
                Index[i+1]= Index[i+1] + Index[i];}

        NPLU= Index[N];
        for(i=0;i<N;i++){
                int jS = Index[i];
                if(i == 0){
                        Item[jS  ] = i+1;
                        Item[jS+1] = i+2;
                }else if(i == N-1){
                        Item[jS  ] = i-2;
                        Item[jS+1] = i-1;
                }else{
                  if (i%2==1){
                        Item[jS  ] = i-1;
                        Item[jS+1] = i+1;
                  } else       {
                        Item[jS  ] = i-2;
                        Item[jS+1] = i-1;
                        Item[jS+2] = i+1;
                        Item[jS+3] = i+2;}}}
```

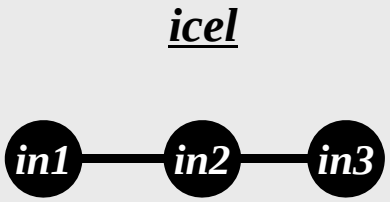


jS= index[i]

Program: 1d2.c (5/7)

Element Matrix ~ Global Matrix

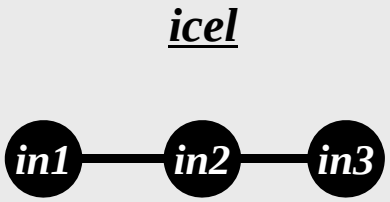
```
/*
// +-----------------+
// | MATRIX assemble |
// +-----------------+
*/
 for(icel=0;icel<NE;icel++){
          in1= Icelnod[3*icel];
          in2= Icelnod[3*icel+1];
          in3= Icelnod[3*icel+2];
          X1 = X[in1];
          X2 = X[in2];
          X3 = X[in3];
          DL = fabs(X3-X1);

          Ck= Area*Young/DL;
          Emat[0][0]= Ck*Kmat[0][0];
          Emat[0][1]= Ck*Kmat[0][1];
          Emat[0][2]= Ck*Kmat[0][2];
          Emat[1][0]= Ck*Kmat[1][0];
          Emat[1][1]= Ck*Kmat[1][1];
          Emat[1][2]= Ck*Kmat[1][2];
          Emat[2][0]= Ck*Kmat[2][0];
          Emat[2][1]= Ck*Kmat[2][1];
          Emat[2][2]= Ck*Kmat[2][2];

          Diag[in1]= Diag[in1] + Emat[0][0];
          Diag[in2]= Diag[in2] + Emat[1][1];
          Diag[in3]= Diag[in3] + Emat[2][2];
```



Program: 1d2.c (5/7)

Element Matrix ~ Global Matrix

```
/*
// +-----------------+
// | MATRIX assemble |
// +-----------------+
*/
 for(icel=0;icel<NE;icel++){
          in1= Icelnod[3*icel];
          in2= Icelnod[3*icel+1];
          in3= Icelnod[3*icel+2];
          X1 = X[in1];
          X2 = X[in2];
          X3 = X[in3];
          DL = fabs(X3-X1);

          Ck= Area*Young/DL;
          Emat[0][0]= Ck*Kmat[0][0];
          Emat[0][1]= Ck*Kmat[0][1];
          Emat[0][2]= Ck*Kmat[0][2];
          Emat[1][0]= Ck*Kmat[1][0];
          Emat[1][1]= Ck*Kmat[1][1];
          Emat[1][2]= Ck*Kmat[1][2];
          Emat[2][0]= Ck*Kmat[2][0];
          Emat[2][1]= Ck*Kmat[2][1];
          Emat[2][2]= Ck*Kmat[2][2];

          Diag[in1]= Diag[in1] + Emat[0][0];
          Diag[in2]= Diag[in2] + Emat[1][1];
          Diag[in3]= Diag[in3] + Emat[2][2];
```

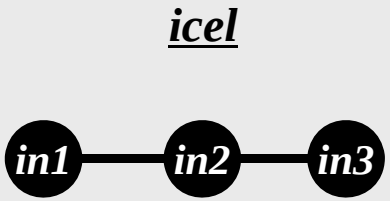


$$[Emat] = \frac{EA}{L}\frac{1}{6}\begin{bmatrix} +14 & -16 & +2 \\ -16 & +32 & -16 \\ +2 & -16 & +14 \end{bmatrix} = \frac{EA}{L}[Kmat]$$

Program: 1d2.c (5/7)

Element Matrix ~ Global Matrix

```
/*
// +-----------------+
// | MATRIX assemble |
// +-----------------+
*/
 for(icel=0;icel<NE;icel++){
          in1= Icelnod[3*icel];
          in2= Icelnod[3*icel+1];
          in3= Icelnod[3*icel+2];
          X1 = X[in1];
          X2 = X[in2];
          X3 = X[in3];
          DL = fabs(X3-X1);

          Ck= Area*Young/DL;
          Emat[0][0]= Ck*Kmat[0][0];
          Emat[0][1]= Ck*Kmat[0][1];
          Emat[0][2]= Ck*Kmat[0][2];
          Emat[1][0]= Ck*Kmat[1][0];
          Emat[1][1]= Ck*Kmat[1][1];
          Emat[1][2]= Ck*Kmat[1][2];
          Emat[2][0]= Ck*Kmat[2][0];
          Emat[2][1]= Ck*Kmat[2][1];
          Emat[2][2]= Ck*Kmat[2][2];

          Diag[in1]= Diag[in1] + Emat[0][0];
          Diag[in2]= Diag[in2] + Emat[1][1];
          Diag[in3]= Diag[in3] + Emat[2][2];
```



$$[Emat] = \frac{EA}{6L}\begin{bmatrix} +14 & -16 & +2 \\ -16 & +32 & -16 \\ +2 & -16 & +14 \end{bmatrix}$$

Program: 1d2.c (6/7)

Global Matrix

```
if (icel==0){k12=Index[in1];
             k13=Index[in1]+1;
      }else {k12=Index[in1]+2;
             k13=Index[in1]+3;}

       k21=Index[in2];
       k23=Index[in2]+1;

       k31=Index[in3];
       k32=Index[in3]+1;

       AMat[k12]= AMat[k12] + Emat[0][1];
       AMat[k13]= AMat[k13] + Emat[0][2];
       AMat[k21]= AMat[k21] + Emat[1][0];
       AMat[k23]= AMat[k23] + Emat[1][2];
       AMat[k31]= AMat[k31] + Emat[2][0];
       AMat[k32]= AMat[k32] + Emat[2][1];
 }
```

$$[Emat] = \frac{EA}{6L}\begin{bmatrix} +14 & -16 & +2 \\ -16 & +32 & -16 \\ +2 & -16 & +14 \end{bmatrix}$$

1st row: corresponds to *in1* $\begin{bmatrix} - & k12 & k13 \end{bmatrix}$

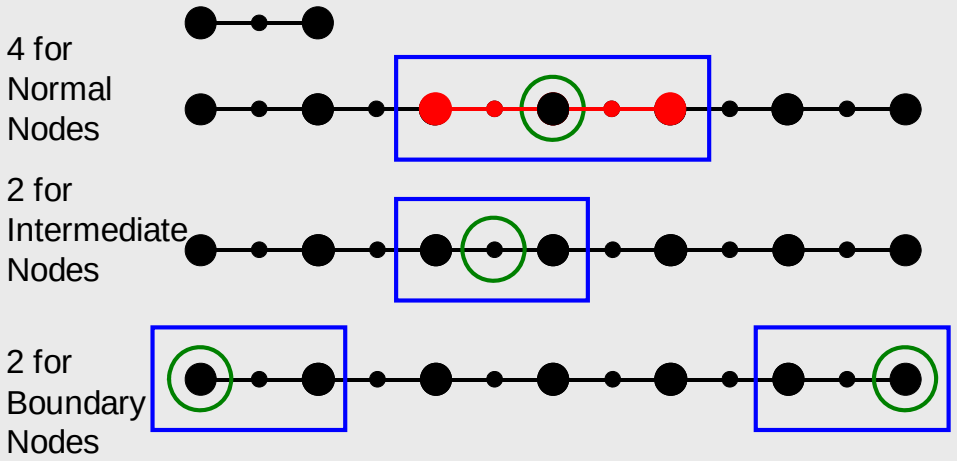
2nd row: corresponds to *in2* $\begin{bmatrix} k21 & - & k23 \end{bmatrix}$

3rd row: corresponds to *in3* $\begin{bmatrix} k31 & k32 & - \end{bmatrix}$

Supplementary Materials

Non-Zero Off-Diagonals – Column Numbers

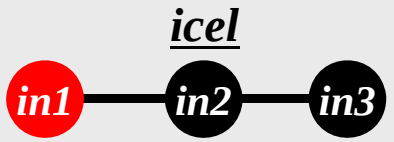
```
/*
// +--------------+
// | CONNECTIVITY |
// +--------------+
*/
        Index[0]= 0;
        for(i=1;i<N;i++) {
          if (i%2==1) {Index[i]=4;
          }else       {Index[i]=2;}}
        Index[1]= 2;
        Index[N]= 2;
        for(i=0;i<N;i++){
                Index[i+1]= Index[i+1] + Index[i];}

        NPLU= Index[N];
        for(i=0;i<N;i++){
                int jS = Index[i];
                if(i == 0){
                        Item[jS  ] = i+1;
                        Item[jS+1] = i+2;
                }else if(i == N-1){
                        Item[jS  ] = i-2;
                        Item[jS+1] = i-1;
                }else{
                  if (i%2==1){
                        Item[jS  ] = i-1;
                        Item[jS+1] = i+1;
                  } else       {
                        Item[jS  ] = i-2;
                        Item[jS+1] = i-1;
                        Item[jS+2] = i+1;
                        Item[jS+3] = i+2;}}}
```



Supplementary Materials

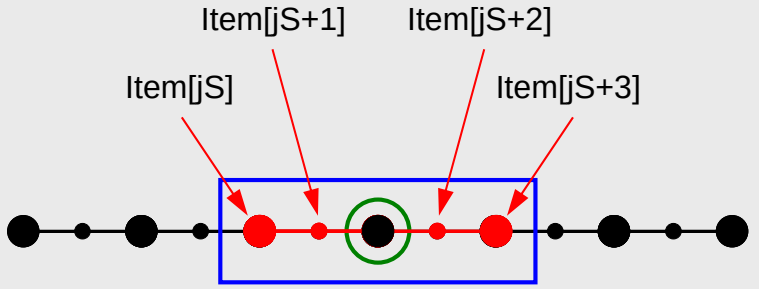
Non-Zero Off-Diagonals – Column Numbers

```
/*
// +--------------+
// | CONNECTIVITY |
// +--------------+
*/
      Index[0]= 0;
      for(i=1;i<N;i++) {
        if (i%2==1) {Index[i]=4;
        }else       {Index[i]=2;}}
      Index[1]= 2;
      Index[N]= 2;
      for(i=0;i<N;i++){
              Index[i+1]= Index[i+1] + Index[i];}

      NPLU= Index[N];
      for(i=0;i<N;i++){
              int jS = Index[i];
              if(i == 0){
                      Item[jS  ] = i+1;
                      Item[jS+1] = i+2;
              }else if(i == N-1){
                      Item[jS  ] = i-2;
                      Item[jS+1] = i-1;
              }else{
                if (i%2==1){
                      Item[jS  ] = i-1;
                      Item[jS+1] = i+1;
                } else      {
                      Item[jS  ] = i-2;
                      Item[jS+1] = i-1;
                      Item[jS+2] = i+1;
                      Item[jS+3] = i+2;}}}
```




```
if (icel==0){k12=Index[in1];
             k13=Index[in1]+1;
      }else {k12=Index[in1]+2;
             k13=Index[in1]+3;}

       k21=Index[in2];
       k23=Index[in2]+1;

       k31=Index[in3];
       k32=Index[in3]+1;}
```

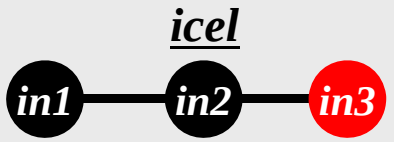
jS= index[i]



Supplementary Materials

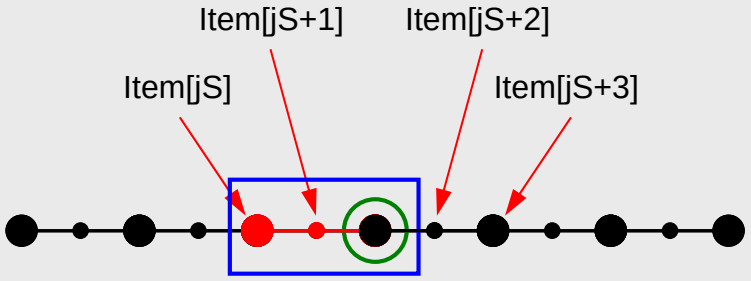
Non-Zero Off-Diagonals – Column Numbers

```
/*
// +-------------+
// | CONNECTIVITY |
// +-------------+
*/
      Index[0]= 0;
      for(i=1;i<N;i++) {
        if (i%2==1) {Index[i]=4;
        }else       {Index[i]=2;}}
      Index[1]= 2;
      Index[N]= 2;
      for(i=0;i<N;i++){
              Index[i+1]= Index[i+1] + Index[i];}

      NPLU= Index[N];
      for(i=0;i<N;i++){
              int jS = Index[i];
              if(i == 0){
                      Item[jS  ] = i+1;
                      Item[jS+1] = i+2;
              }else if(i == N-1){
                      Item[jS  ] = i-2;
                      Item[jS+1] = i-1;
              }else{
                if (i%2==1){
                      Item[jS  ] = i-1;
                      Item[jS+1] = i+1;
                } else      {
                      Item[jS  ] = i-2;
                      Item[jS+1] = i-1;
                      Item[jS+2] = i+1;
                      Item[jS+3] = i+2;}}}
```




```
if (icel==0){k12=Index[in1];
             k13=Index[in1]+1;
      }else {k12=Index[in1]+2;
             k13=Index[in1]+3;}

       k21=Index[in2];
       k23=Index[in2]+1;

       k31=Index[in3];
       k32=Index[in3]+1;}
```

jS= index[i]



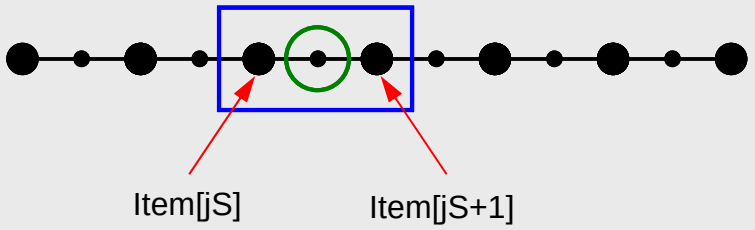
Supplementary Materials

Non-Zero Off-Diagonals – Column Numbers

```
/*
// +-------------+
// | CONNECTIVITY |
// +-------------+
*/
      Index[0]= 0;
      for(i=1;i<N;i++) {
        if (i%2==1) {Index[i]=4;
        }else       {Index[i]=2;}}
      Index[1]= 2;
      Index[N]= 2;
      for(i=0;i<N;i++){
              Index[i+1]= Index[i+1] + Index[i];}

      NPLU= Index[N];
      for(i=0;i<N;i++){
              int jS = Index[i];
              if(i == 0){
                      Item[jS  ] = i+1;
                      Item[jS+1] = i+2;
              }else if(i == N-1){
                      Item[jS  ] = i-2;
                      Item[jS+1] = i-1;
              }else{
                if (i%2==1){
                      Item[jS  ] = i-1;
                      Item[jS+1] = i+1;
                } else       {
                      Item[jS  ] = i-2;
                      Item[jS+1] = i-1;
                      Item[jS+2] = i+1;
                      Item[jS+3] = i+2;}}}
```


```
if (icel==0){k12=Index[in1];
             k13=Index[in1]+1;
      }else {k12=Index[in1]+2;
             k13=Index[in1]+3;}

       k21=Index[in2];
       k23=Index[in2]+1;

       k31=Index[in3];
       k32=Index[in3]+1;}
```

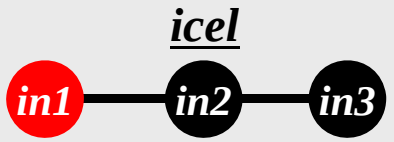
jS= index[i]



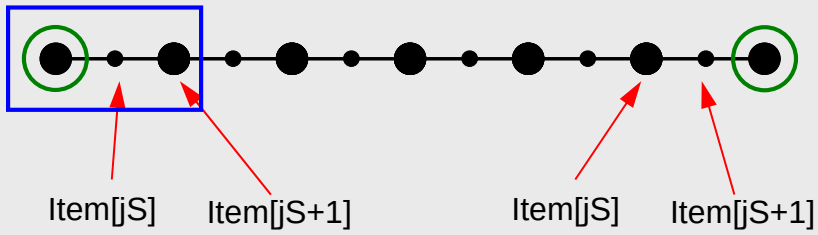
Supplementary Materials

Non-Zero Off-Diagonals – Column Numbers

```
/*
// +--------------+
// | CONNECTIVITY |
// +--------------+
*/
      Index[0]= 0;
      for(i=1;i<N;i++) {
        if (i%2==1) {Index[i]=4;
        }else       {Index[i]=2;}}
      Index[1]= 2;
      Index[N]= 2;
      for(i=0;i<N;i++){
              Index[i+1]= Index[i+1] + Index[i];}

      NPLU= Index[N];
      for(i=0;i<N;i++){
              int jS = Index[i];
              if(i == 0){
                      Item[jS  ] = i+1;
                      Item[jS+1] = i+2;
              }else if(i == N-1){
                      Item[jS  ] = i-2;
                      Item[jS+1] = i-1;
              }else{
                if (i%2==1){
                      Item[jS  ] = i-1;
                      Item[jS+1] = i+1;
                } else       {
                      Item[jS  ] = i-2;
                      Item[jS+1] = i-1;
                      Item[jS+2] = i+1;
                      Item[jS+3] = i+2;}}}
```




```
if (icel==0){k12=Index[in1];
             k13=Index[in1]+1;
      }else {k12=Index[in1]+2;
             k13=Index[in1]+3;}

       k21=Index[in2];
       k23=Index[in2]+1;

       k31=Index[in3];
       k32=Index[in3]+1;}
```



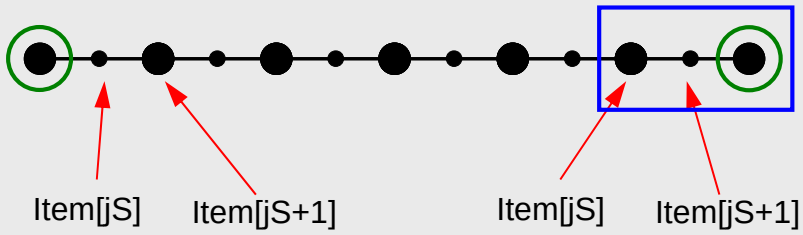
jS= index[i]

Supplementary Materials

Non-Zero Off-Diagonals – Column Numbers

```
/*
// +--------------+
// | CONNECTIVITY |
// +--------------+
*/
      Index[0]= 0;
      for(i=1;i<N;i++) {
        if (i%2==1) {Index[i]=4;
        }else       {Index[i]=2;}}
      Index[1]= 2;
      Index[N]= 2;
      for(i=0;i<N;i++){
              Index[i+1]= Index[i+1] + Index[i];}

      NPLU= Index[N];
      for(i=0;i<N;i++){
              int jS = Index[i];
              if(i == 0){
                      Item[jS  ] = i+1;
                      Item[jS+1] = i+2;
              }else if(i == N-1){
                      Item[jS  ] = i-2;
                      Item[jS+1] = i-1;
              }else{
                if (i%2==1){
                      Item[jS  ] = i-1;
                      Item[jS+1] = i+1;
                } else      {
                      Item[jS  ] = i-2;
                      Item[jS+1] = i-1;
                      Item[jS+2] = i+1;
                      Item[jS+3] = i+2;}}}
```

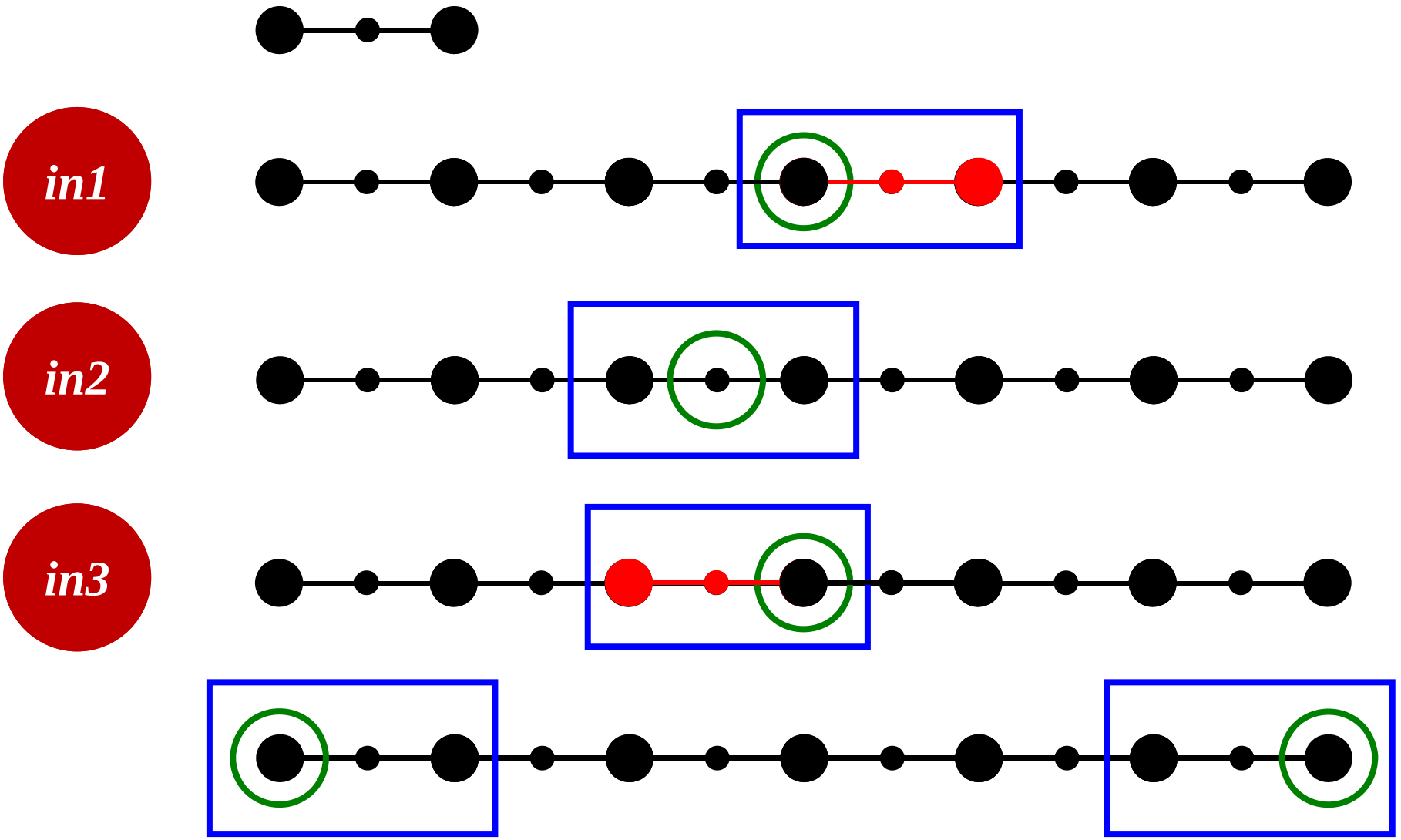

```
if (icel==0){k12=Index[in1];
             k13=Index[in1]+1;
      }else {k12=Index[in1]+2;
             k13=Index[in1]+3;}

       k21=Index[in2];
       k23=Index[in2]+1;

       k31=Index[in3];
       k32=Index[in3]+1;}
```



jS= index[i]

Number of Non-Zero Off-Diagonals

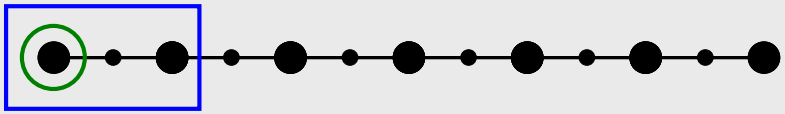
Different number of connections according to location of each node



Program: 1d2.c (7/7)

Boundary Conditions

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

/* X=Xmin */
        i=0;
        jS= Index[i];
        AMat[jS]  = 0.0;
        AMat[jS+1]= 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;
```



Just solve derived linear equations by CG method

If A (sectional area) is function of x …

$$\int_V E\left(\frac{d[N]^T}{dx}\frac{d[N]}{dx}\right)dV = \int_0^L \begin{bmatrix} dN_i/dx \\ dN_j/dx \\ dN_k/dx \end{bmatrix} E\left[\frac{dN_i}{dx}, \frac{dN_j}{dx}, \frac{dN_k}{dx}\right] A(x)\,dx$$

$$= E\int_0^L \begin{bmatrix} A(x)\frac{dN_i}{dx}\frac{dN_i}{dx} & A(x)\frac{dN_i}{dx}\frac{dN_j}{dx} & A(x)\frac{dN_i}{dx}\frac{dN_k}{dx} \\ A(x)\frac{dN_j}{dx}\frac{dN_i}{dx} & A(x)\frac{dN_j}{dx}\frac{dN_j}{dx} & A(x)\frac{dN_j}{dx}\frac{dN_k}{dx} \\ A(x)\frac{dN_k}{dx}\frac{dN_i}{dx} & A(x)\frac{dN_k}{dx}\frac{dN_j}{dx} & A(x)\frac{dN_k}{dx}\frac{dN_k}{dx} \end{bmatrix} dx$$

$$N_i = \left(1-\frac{2x}{L}\right)\left(1-\frac{x}{L}\right) \qquad \frac{dN_i}{dx} = \left(\frac{4x}{L^2}-\frac{3}{L}\right)$$

$$N_j = \left(\frac{4x}{L}\right)\left(1-\frac{x}{L}\right) \qquad \frac{dN_j}{dx} = \left(\frac{4}{L}-\frac{8x}{L^2}\right)$$

$$N_k = \left(-\frac{x}{L}\right)\left(1-\frac{2x}{L}\right) \qquad \frac{dN_k}{dx} = \left(\frac{4x}{L^2}-\frac{1}{L}\right)$$

Integration is possible, but very complicated.
It's convenient if this is done numerically.

- 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

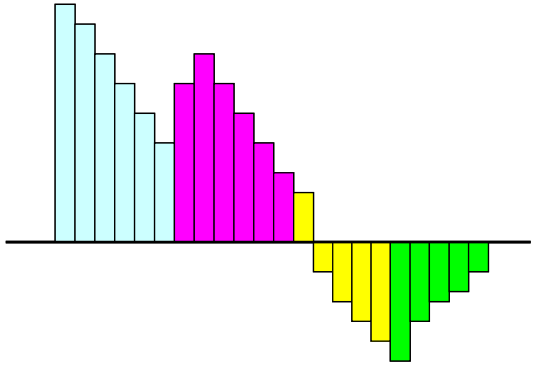
- Gaussian Quadrature
- Isoparametric Elements
- Implementation
- Report #1

Methods for Numerical Integration

- Trapezoidal Rule
- Simpson's Rule
- Gaussian Quadrature (or Gauss-Legendre)
  - accurate

- Values of functions at finite numbers of sample points are utilized:

$$\int_{X_1}^{X_2} f(x)dx \Rightarrow \sum_{k=1}^{m} [w_k \cdot f(x_k)]$$



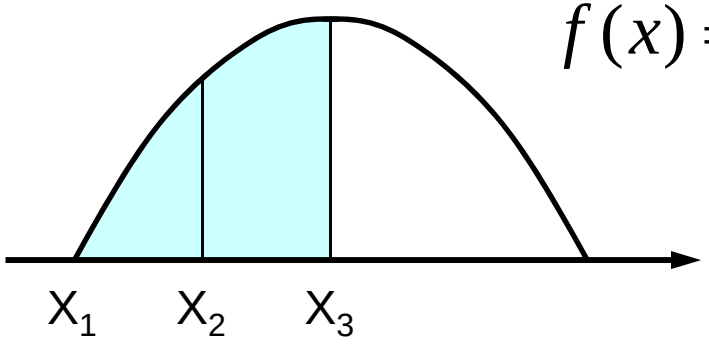
Gaussian Quadrature in 1D

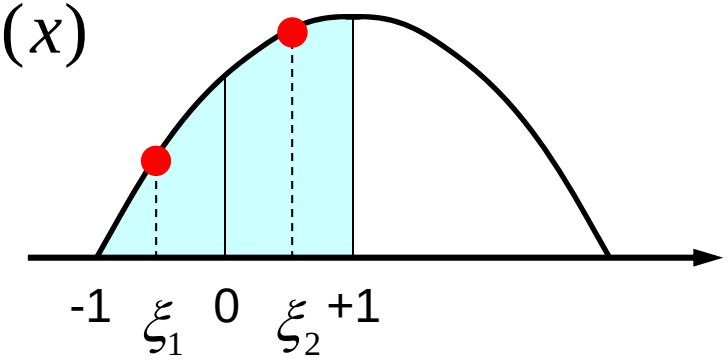
more accurate than Simpson's rule

Simpson's

Gauss

$$f(x) = \sin(x)$$





$$X_1 = 0, \quad X_2 = \frac{\pi}{4}, \quad X_3 = \frac{\pi}{2}$$

$$\xi_1, \xi_2 = \pm 0.5773502692$$

$$h = X_2 - X_1 = X_3 - X_1 = \frac{\pi}{4}$$

$$S = \int_0^{\pi/2} f(x)dx = \int_{-1}^{+1} f(\xi)\, h\, d\xi$$

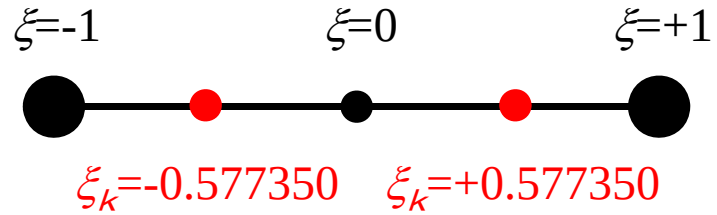
$$S = \frac{h}{3}\left[f(X_1) + 4f(X_2) + f(X_3)\right] = 1.0023$$

$$\cong h\sum_{k=1}^{2} W_k \cdot f(\xi_k) = 0.99847$$

Gaussian Quadrature
ガウスの積分公式

- On normalized “natural (or local)” coordinate system [-1,+1]（自然座標系，局所座標系）
- Can approximate up to (*2m-1*)-th order of functions by *m* quadrature points (*m*=2 is enough for quadratic shape functions).

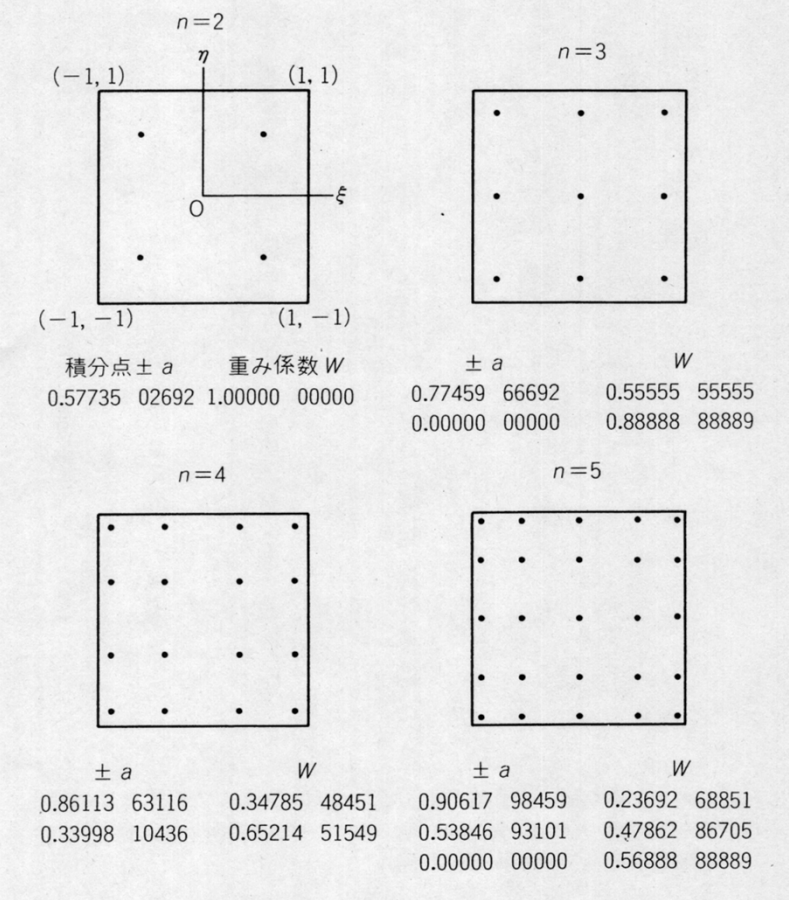
$$\int_{-1}^{+1} f(\xi)d\xi = \sum_{k=1}^{m}[w_k \cdot f(\xi_k)]$$



$$m=1 \quad \xi_k = 0.00,\ w_k = 2.00$$
$$m=2 \quad \xi_k = \pm 0.577350,\ w_k = 1.00$$
$$m=3 \quad \xi_k = 0.00,\ w_k = 8/9$$
$$\xi_k = \pm 0.774597,\ w_k = 5/9$$

Gaussian Quadrature

can be easily extended to 2D & 3D



$$I = \int_{-1}^{+1} \int_{-1}^{+1} f(\xi,\eta)\, d\xi d\eta$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{n} \left[W_i \cdot W_j \cdot f(\xi_i,\eta_j) \right]$$

m,n: number of quadrature points in ξ,η-direction

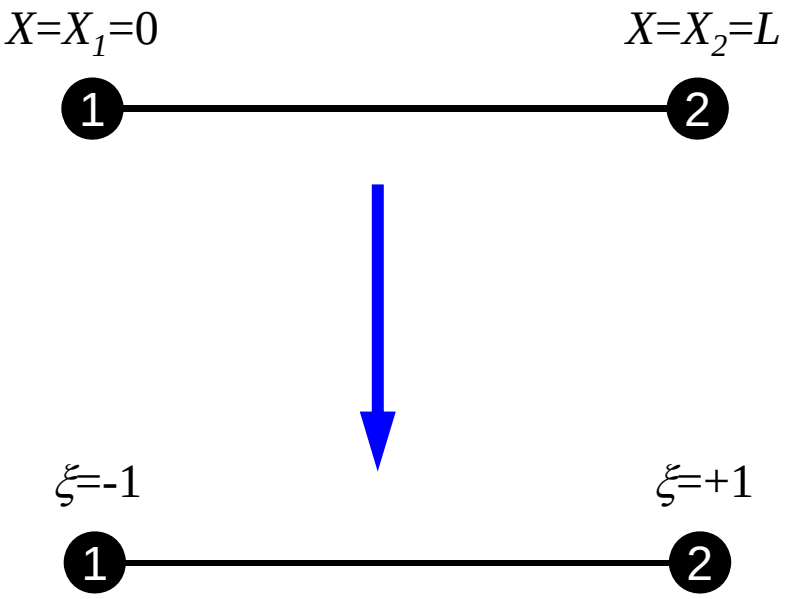
(ξ_i,η_j): Coordinates of Quad's

W_i, W_j: Weighting Factor

- Gaussian Quadrature
- Isoparametric Elements
- Implementation
- Report #1

How to use Gaussian Quadrature

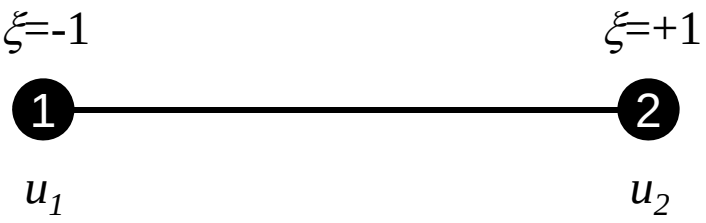
- Coordinate transformation from $[0,L]$ (or $[X_1,X_2]$) to [-1,+1] is needed.
- Shape/Interpolation functions must be handled on natural/local coordinate system.



Local node ID's (1,2,3…) are defined as subscripts of X, u, N and etc.

Piecewise Linear Element

$$u = \alpha_1 + \alpha_2 \xi$$
$$u_1 = \alpha_1 + \alpha_2 (-1)$$
$$u_2 = \alpha_1 + \alpha_2 (+1)$$



$$\alpha_1 = \frac{u_1 + u_2}{2}, \alpha_2 = \frac{-u_1 + u_2}{2}$$

$$u = \alpha_1 + \alpha_2 \xi = \frac{u_1 + u_2}{2} + \frac{-u_1 + u_2}{2} \xi = \underbrace{\frac{1}{2}(1-\xi)}_{N_1(\xi)} u_1 + \underbrace{\frac{1}{2}(1+\xi)}_{N_2(\xi)} u_2$$

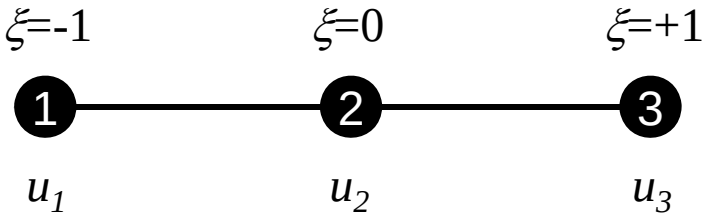
$$N_1(\xi) = \frac{1}{2}(1-\xi), \quad N_2(\xi) = \frac{1}{2}(1+\xi)$$

2nd-order/Quadratic Element

$$u = \alpha_1 + \alpha_2 \xi + \alpha_3 \xi^2$$

$$u_1 = \alpha_1 - \alpha_2 + \alpha_3, \quad u_2 = \alpha_1$$

$$u_3 = \alpha_1 + \alpha_2 + \alpha_3$$



$$\alpha_1 = u_2, \alpha_2 = \frac{-u_1 + u_3}{2}, \alpha_3 = \frac{u_1 - 2u_2 + u_3}{2}$$

$$u = \alpha_1 + \alpha_2 \xi + \alpha_3 \xi^2 = u_2 + \frac{-u_1 + u_3}{2}\xi + \frac{u_1 - 2u_2 + u_3}{2}\xi^2$$

$$= \frac{-\xi + \xi^2}{2} u_1 + \left(1 - \xi^2\right) u_2 + \frac{\xi + \xi^2}{2} u_3$$

$$= \frac{1}{2}\xi(-1+\xi) u_1 + (1+\xi)(1-\xi) u_2 + \frac{1}{2}\xi(1+\xi) u_3$$

$N_1(\xi)$ $N_2(\xi)$ $N_3(\xi)$

$$N_1(\xi) = \frac{1}{2}\xi(-1+\xi), \; N_2(\xi) = (1+\xi)(1-\xi), \; N_3(\xi) = \frac{1}{2}\xi(1+\xi)$$

Isoparametric Element
アイソパラメトリック要素

- Definitions of “Isoparametric” Elements
  - Each element is defined on natural/local coordinate for [-1,+1]
  - And the components of global coordinate system of each node (e.g. x,y,z) for certain kinds of elements are defined by shape functions $[N]$ on natural/local coordinate system, where shape functions $[N]$ are also used for interpolation of dependent variables.

$$u = \sum_{i=1}^{n_N} N_i(\xi) \cdot u_i, \quad x = \sum_{i=1}^{n_N} N_i(\xi) \cdot x_i$$

$$n_N : 2,3...$$

Sub-Parametric
Super-Parametric

$$u = \frac{1}{2}(1-\xi)u_1 + \frac{1}{2}(1+\xi)u_2$$

$$x = \frac{1}{2}(1-\xi)x_1 + \frac{1}{2}(1+\xi)x_2$$

$$\left(= \frac{1}{2}(x_2 - x_1)\xi + \frac{1}{2}(x_1 + x_2) \right)$$

- Gaussian Quadrature
- Isoparametric Elements
- Implementation
- Report #1

Integration over Each Element: $[k]$

$$\int_V E\left(\frac{\partial[N]^T}{\partial x}\frac{\partial[N]}{\partial x}\right)dV = \int_V E\left(\frac{\partial[N]^T}{\partial x}\frac{\partial[N]}{\partial x}\right)Adx$$

Partial Differentiation on Natural/Local Coordinate System (1/2)

- According to formulae:

$$\frac{\partial N_i(\xi)}{\partial \xi} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial \xi}$$

$\left[\frac{\partial N_i}{\partial \xi}\right]$ can be easily derived according to definitions.

$\left[\frac{\partial N_i}{\partial x}\right]$ are required for computations.

$\left[\frac{\partial x}{\partial \xi}\right]$ is Jacobian (=J)

$$\frac{\partial x}{\partial \xi} = \frac{\partial}{\partial \xi}\left(\sum_{i=1}^{n_N} N_i x_i\right) = \sum_{i=1}^{n_N} \frac{\partial N_i}{\partial \xi} x_i = J$$

Partial Differentiation on Natural/Local Coordinate System (2/2)

- Target terms are derived as follows:

$$\frac{\partial N_i}{\partial x} = \frac{\dfrac{\partial N_i(\xi)}{\partial \xi}}{\dfrac{\partial x}{\partial \xi}} = \frac{\partial N_i(\xi)}{\partial \xi} \cdot \frac{1}{J}$$

- Integration on natural/local coordinate system:

$$\int_0^L f(x)dx = \int_{-1}^{+1} f(\xi)|J|d\xi \quad \because dx = |J|d\xi = \left|\frac{\partial x}{\partial \xi}\right| d\xi$$

Integration over Each Element: $[k]$ (1/2)

$$\int_V E\left(\frac{\partial[N]^T}{\partial x}\frac{\partial[N]}{\partial x}\right)dV = E\int_V\left(\frac{\partial[N]^T}{\partial x}\frac{\partial[N]}{\partial x}\right)A(x)dx$$

$$= E\int_{-1}^{+1}\left(\frac{\partial[N]^T}{\partial x}\frac{\partial[N]}{\partial x}\right)A(\xi)|J|d\xi = E\int_{-1}^{+1}\left(\left[\frac{\partial[N]^T}{\partial \xi}\frac{1}{J}\right]\left[\frac{\partial[N]}{\partial \xi}\frac{1}{J}\right]\right)A(\xi)|J|d\xi$$

$$= E\int_{-1}^{+1}\left(\frac{1}{|J|}\frac{\partial[N]^T}{\partial \xi}\frac{\partial[N]}{\partial \xi}A(\xi)\right)d\xi$$

$$= E\sum_{k=1}^{m}\left[w_k \cdot \frac{1}{|J|}\bigg|_{\xi=\xi_k}\frac{\partial[N]^T}{\partial \xi}\bigg|_{\xi=\xi_k}\frac{\partial[N]}{\partial \xi}\bigg|_{\xi=\xi_k}A(\xi_k)\right]$$

Integration over Each Element: $[k]$ (2/2)

$$E\sum_{k=1}^{m}\left[w_k \cdot \frac{1}{|J|}\bigg|_{\xi=\xi_k} \frac{\partial[N]^T}{\partial\xi}\bigg|_{\xi=\xi_k} \frac{\partial[N]}{\partial\xi}\bigg|_{\xi=\xi_k} A(\xi_k) \right]$$

$$= E\sum_{k=1}^{m}\left[w_k \cdot \frac{1}{|J|}\bigg|_{\xi=\xi_k} \begin{bmatrix} \partial N_1/\partial\xi \\ \partial N_2/\partial\xi \end{bmatrix} \begin{bmatrix} \frac{\partial N_1}{\partial\xi}, \frac{\partial N_2}{\partial\xi} \end{bmatrix}\bigg|_{\xi=\xi_k} A(\xi_k) \right]$$

$$= E\sum_{k=1}^{m}\left[w_k \cdot \frac{1}{|J|}\bigg|_{\xi=\xi_k} \begin{bmatrix} \frac{\partial N_1}{\partial\xi}\frac{\partial N_1}{\partial\xi} & \frac{\partial N_1}{\partial\xi}\frac{\partial N_2}{\partial\xi} \\ \frac{\partial N_2}{\partial\xi}\frac{\partial N_1}{\partial\xi} & \frac{\partial N_2}{\partial\xi}\frac{\partial N_2}{\partial\xi} \end{bmatrix}\bigg|_{\xi=\xi_k} A(\xi_k) \right]$$

Files

“b1.c”: “a1.c” on natural/local coordinate
1D Piecewise Linear Elements

1D Code for Static Linear-Elastic Problems with Variable Sectional Area

```
>$ cd <$fem1>
>$ cp /home03/skengon/Documents/class/fem1/1d3.tar .
>$ tar xvf 1d3.tar
>$ cd 1darea
```

Compile & Go !

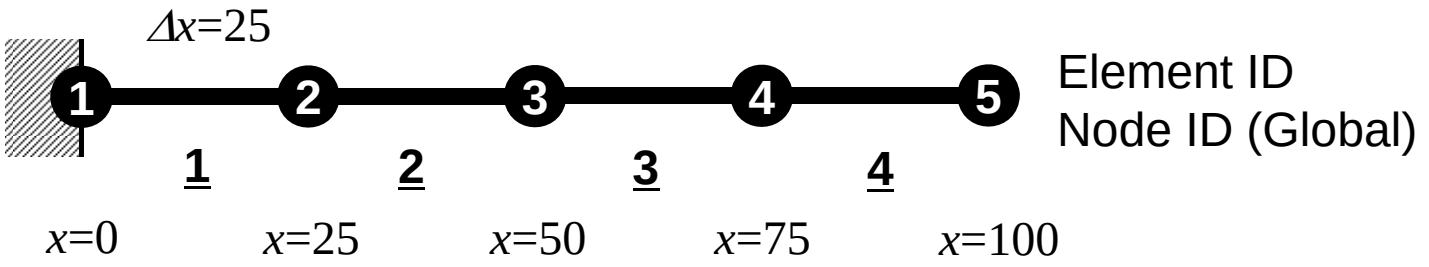
```
>$ cd <$fem1>/1darea
>$ cc -O b1.c      (or g95 -O b1.f)
>$ ./a.out
```

input.dat

```
4
25.0   5.e4   -0.105 12   5.e6
100
1.e-8
```

NE (Number of Elements)
Δx, **F**, **A_1**, **A_2**, **E**
Number of MAX. Iterations for CG Solver
Convergence Criteria for CG Solver

$$x = 0 \quad A_1 x + A_2 = 12.$$
$$x = 100 \quad A_1 x + A_2 = 1.5$$



Program: b1.c (1/6)
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, A1, A2, F, Young, Jacobi;
        double X1, X2, U1, U2, DL, Strain, Sigma, Ck, XX, X0, Disp;
        double *U, *Rhs, *X;
        double *Diag, *AMat;
        double **W;

        int *Index, *Item, *Icelnod;
        double POI[2], WEI[2], dNdQ[2], Emat[2][2];

        int i, j, in1, in2, k, icel, k1, k2, jS, ip;
        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, A1, A2	R		I	Area= A1*x + A2
F	R		I	Axial Force F at X=Xmax
Young	R		I	Young's Modulus
Jacobi	R		O	Jacobian at Gaussian Quadrature Point
X0	R		O	Location of Mid-Point of Elem. (Global)
XX	R		O	Location of Gauusian Quad. Point (Global)
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
Disp	R		O	Displacement at Node
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
POI, WEI	R	[2]	O	Gaussian Quad. Point, Weighting Factor
dNdQ	R	[2]	O	$dN/d\xi$
Emat	R	[2][2]	O	Element Matrix

Program: b1.c (2/6)
Initialization, Allocation of Arrays

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

Program: b1.c (3/6)

Initialization, Allocation of Arrays (cont.)

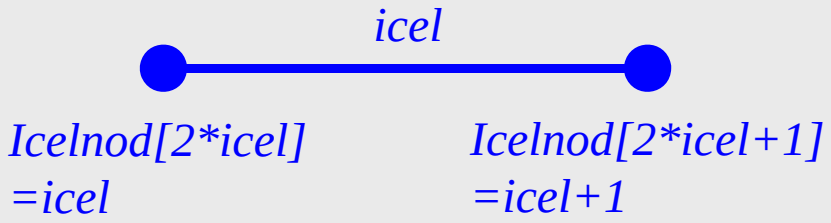
```
W = (double **)malloc(sizeof(double *)*4);
if(W == NULL) {
        fprintf(stderr, "Error: %s¥n", strerror(errno));
        return -1;
}
for(i=0; i<4; i++) {
        W[i] = (double *)malloc(sizeof(double)*N);
        if(W[i] == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));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

for(i=0;i<N;i++)    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;
}

WEI[0]= +1.0;
WEI[1]= +1.0;
POI[0]= -0.577350;
POI[1]= +0.577350;
```

X: X-coordinate component of each node

Program: b1.c (3/6)

Initialization, Allocation of Arrays (cont.)

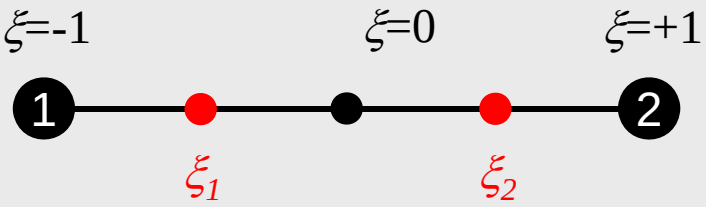
```
W = (double **)malloc(sizeof(double *)*4);
if(W == NULL) {
        fprintf(stderr, "Error: %s¥n", strerror(errno));
        return -1;
}
for(i=0; i<4; i++) {
        W[i] = (double *)malloc(sizeof(double)*N);
        if(W[i] == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));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

for(i=0;i<N;i++)    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;
}

WEI[0]= +1.0;
WEI[1]= +1.0;
POI[0]= -0.577350;
POI[1]= +0.577350;
```



Program: b1.c (3/6)

Initialization, Allocation of Arrays (cont.)

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

for(i=0;i<N;i++)    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;
}

WEI[0]= +1.0;
WEI[1]= +1.0;
POI[0]= -0.577350;
POI[1]= +0.577350;
```



Program: b1.c (4/6)

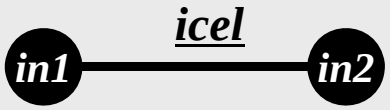
Global Matrix: Column ID for Non-Zero Off-Diag's

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

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

        NPLU= Index[N];

        for(i=0;i<N;i++){
                int 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;
                }
        }
```

Program: b1.c (5/6)

Element Matrix ~ Global Matrix

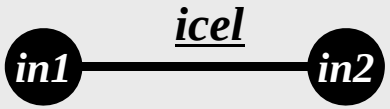
```
/*
// +-----------------+
// | MATRIX assemble |
// +-----------------+
*/
      for(icel=0;icel<NE;icel++){
          in1= Icelnod[2*icel];
          in2= Icelnod[2*icel+1];
          X1 = X[in1];
          X2 = X[in2];
          DL = fabs(X2-X1);
          X0 = 0.5 * (X1+X2);

          Emat[0][0]= 0.0;
          Emat[0][1]= 0.0;
          Emat[1][0]= 0.0;
          Emat[1][1]= 0.0;

          for(ip=0;ip<2;ip++){
            dNdQ[0]= -0.5;
            dNdQ[1]= +0.5;
            XX= X0 + POI[ip]*0.50*DL;
            Area= A1*XX + A2;
            if(Area<= 0.) {
                 fprintf(stderr, "ERROR: Area<0: ¥n");
                 return -1;}
            Jacobi= fabs(dNdQ[0]*X1 + dNdQ[1]*X2);
            Ck= Area*Young/Jacobi;
            Emat[0][0]= Emat[0][0] + Ck * WEI[ip] * dNdQ[0] * dNdQ[0];
            Emat[0][1]= Emat[0][1] + Ck * WEI[ip] * dNdQ[0] * dNdQ[1];
            Emat[1][0]= Emat[1][0] + Ck * WEI[ip] * dNdQ[1] * dNdQ[0];
            Emat[1][1]= Emat[1][1] + Ck * WEI[ip] * dNdQ[1] * dNdQ[1];}
```



Program: b1.c (5/6)

Element Matrix ~ Global Matrix

```
/*
// +-----------------+
// | MATRIX assemble |
// +-----------------+
*/
      for(icel=0;icel<NE;icel++){
          in1= Icelnod[2*icel];
          in2= Icelnod[2*icel+1];
          X1 = X[in1];
          X2 = X[in2];
          DL = fabs(X2-X1);
          X0 = 0.5 * (X1+X2);

          Emat[0][0]= 0.0;
          Emat[0][1]= 0.0;
          Emat[1][0]= 0.0;
          Emat[1][1]= 0.0;

          for(ip=0;ip<2;ip++){
            dNdQ[0]= -0.5;
            dNdQ[1]= +0.5;
            XX= X0 + POI[ip]*0.50*DL;
            Area= A1*XX + A2;
            if(Area<= 0.) {
                 fprintf(stderr, "ERROR: Area<0: ¥n");
                 return -1;}
            Jacobi= fabs(dNdQ[0]*X1 + dNdQ[1]*X2);
            Ck= Area*Young/Jacobi;
            Emat[0][0]= Emat[0][0] + Ck * WEI[ip] * dNdQ[0] * dNdQ[0];
            Emat[0][1]= Emat[0][1] + Ck * WEI[ip] * dNdQ[0] * dNdQ[1];
            Emat[1][0]= Emat[1][0] + Ck * WEI[ip] * dNdQ[1] * dNdQ[0];
            Emat[1][1]= Emat[1][1] + Ck * WEI[ip] * dNdQ[1] * dNdQ[1];}
```



Derivatives of Shape Functions $dN/d\xi$

Piecewise Linear Element

Derivatives of Shape Functions: Constants

$$N_1(\xi)=\frac{1}{2}(1-\xi),\quad N_2(\xi)=\frac{1}{2}(1+\xi)$$

$$u = N_1(\xi)\cdot u_1 + N_2(\xi)\cdot u_2$$

$$\frac{dN_1}{d\xi}=\frac{d}{d\xi}\left[\frac{1}{2}(1-\xi)\right]=-\frac{1}{2}$$

$$\frac{dN_2}{d\xi}=\frac{d}{d\xi}\left[\frac{1}{2}(1+\xi)\right]=+\frac{1}{2}$$

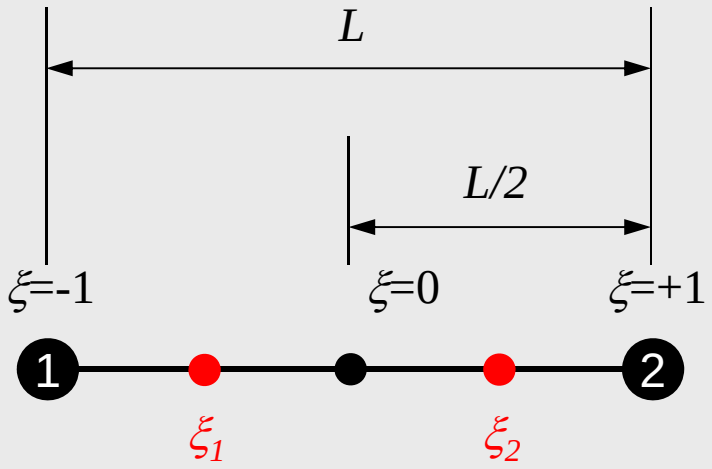
dNdQ[0]
(dNdQ(1) for FOTRAN)

dNdQ[1]
(dNdQ(2) for FOTRAN)

Program: b1.c (5/6)

Element Matrix ~ Global Matrix

```
/*
// +-----------------+
// | MATRIX assemble |
// +-----------------+
*/
      for(icel=0;icel<NE;icel++){
          in1= Icelnod[2*icel];
          in2= Icelnod[2*icel+1];
          X1 = X[in1];
          X2 = X[in2];
          DL = fabs(X2-X1);
          X0 = 0.5 * (X1+X2);

          Emat[0][0]= 0.0;
          Emat[0][1]= 0.0;
          Emat[1][0]= 0.0;
          Emat[1][1]= 0.0;

          for(ip=0;ip<2;ip++){
            dNdQ[0]= -0.5;
            dNdQ[1]= +0.5;
            XX= X0 + POI[ip]*0.50*DL;
            Area= A1*XX + A2;
            if(Area<= 0.) {
                 fprintf(stderr, "ERROR: Area<0: ¥n");
                 return -1;}
            Jacobi= fabs(dNdQ[0]*X1 + dNdQ[1]*X2);
            Ck= Area*Young/Jacobi;
            Emat[0][0]= Emat[0][0] + Ck * WEI[ip] * dNdQ[0] * dNdQ[0];
            Emat[0][1]= Emat[0][1] + Ck * WEI[ip] * dNdQ[0] * dNdQ[1];
            Emat[1][0]= Emat[1][0] + Ck * WEI[ip] * dNdQ[1] * dNdQ[0];
            Emat[1][1]= Emat[1][1] + Ck * WEI[ip] * dNdQ[1] * dNdQ[1];}
```



XX: Global Coordinate for Gaussian Quad. Point

X-Coord. or Gaussian Quad Points

Piecewise Linear Element

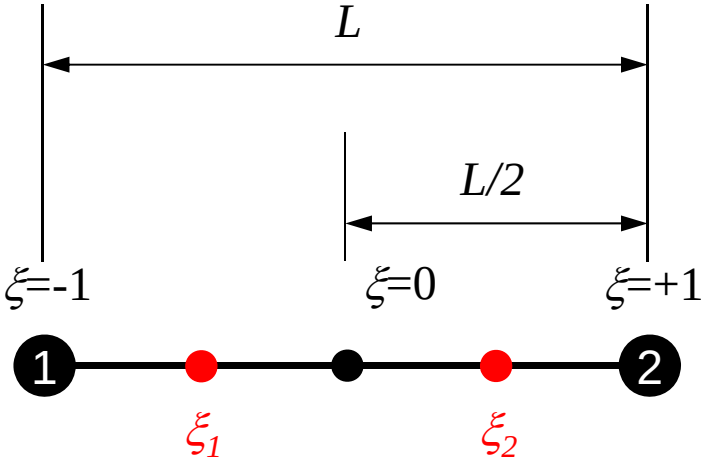
- According to definitions of "isoparametric" elements:

$$X(\xi)=\sum_{k=1}^{2} N_k(\xi)X_k$$

$$= N_1(\xi)X_1 + N_2(\xi)X_2$$

$$= \frac{1}{2}(1-\xi)X_1 + \frac{1}{2}(1+\xi)X_2$$

$$= \frac{X_1+X_2}{2} + \frac{X_2-X_1}{2}\xi$$



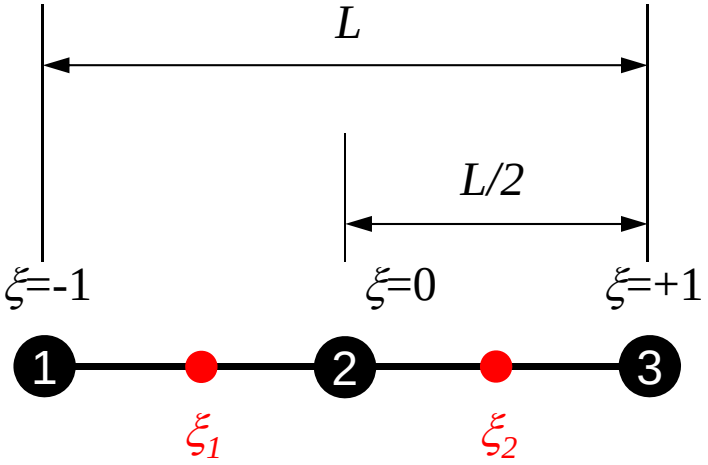
X0 L/2 (=DL/2 in the program)

∴ XX= X0 + POI[ip]*0.50*DL;

X-Coord. or Gaussian Quad Points

2nd-Order/Quadratic Element

- According to definitions of "isoparametric" elements:



$$X(\xi)=\sum_{k=1}^{3} N_k(\xi)X_k$$

$$= N_1(\xi)X_1 + N_2(\xi)X_2 + N_3(\xi)X_3$$

$$= \frac{1}{2}\xi(-1+\xi)X_1 + (1-\xi^2)X_2 + \frac{1}{2}\xi(1+\xi)X_3$$

$$= X_2 + \frac{X_3 - X_1}{2}\xi + \frac{X_1 - 2X_2 + X_3}{2}\xi^2$$

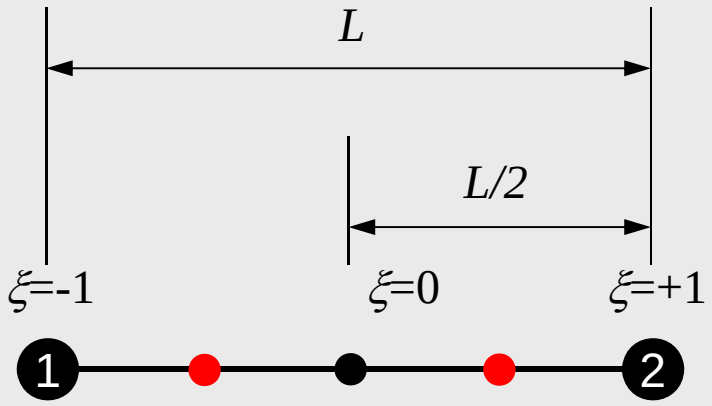
$$= \frac{X_1 + X_3}{2} + \frac{X_3 - X_1}{2}\xi \quad \because X_2 = \frac{X_1 + X_3}{2}$$

∴ XX= XO + POI[ip]*0.50*DL;

Program: b1.c (5/6)

Element Matrix ~ Global Matrix

```
/*
// +-----------------+
// | MATRIX assemble |
// +-----------------+
*/
      for(icel=0;icel<NE;icel++){
          in1= Icelnod[2*icel];
          in2= Icelnod[2*icel+1];
          X1 = X[in1];
          X2 = X[in2];
          DL = fabs(X2-X1);
          X0 = 0.5 * (X1+X2);

          Emat[0][0]= 0.0;
          Emat[0][1]= 0.0;
          Emat[1][0]= 0.0;
          Emat[1][1]= 0.0;

          for(ip=0;ip<2;ip++){
            dNdQ[0]= -0.5;
            dNdQ[1]= +0.5;
            XX= X0 + POI[ip]*0.50*DL;
            Area= A1*XX + A2;
            if(Area<= 0.) {
                 fprintf(stderr, "ERROR: Area<0: ¥n");
                 return -1;}
            Jacobi= fabs(dNdQ[0]*X1 + dNdQ[1]*X2);
            Ck= Area*Young/Jacobi;
            Emat[0][0]= Emat[0][0] + Ck * WEI[ip] * dNdQ[0] * dNdQ[0];
            Emat[0][1]= Emat[0][1] + Ck * WEI[ip] * dNdQ[0] * dNdQ[1];
            Emat[1][0]= Emat[1][0] + Ck * WEI[ip] * dNdQ[1] * dNdQ[0];
            Emat[1][1]= Emat[1][1] + Ck * WEI[ip] * dNdQ[1] * dNdQ[1];}
```

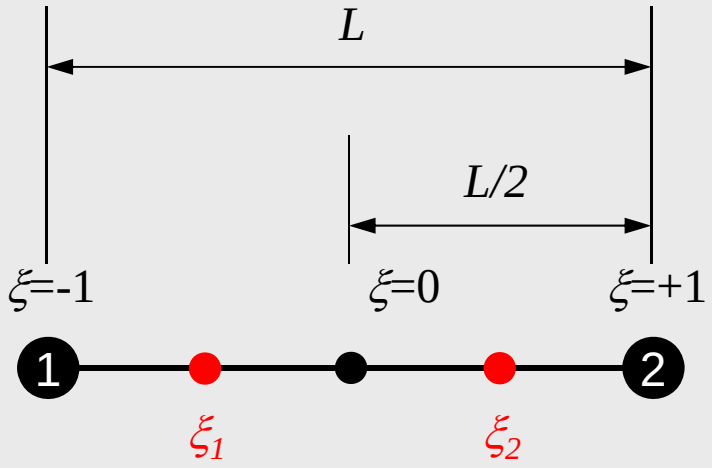


XX: Global Coordinate for Gaussian Quad. Point
Area: Sectional Area at Gaussian Quad. Point

Program: b1.c (5/6)

Element Matrix ~ Global Matrix

```
/*
// +-----------------+
// | MATRIX assemble |
// +-----------------+
*/
      for(icel=0;icel<NE;icel++){
          in1= Icelnod[2*icel];
          in2= Icelnod[2*icel+1];
          X1 = X[in1];
          X2 = X[in2];
          DL = fabs(X2-X1);
          X0 = 0.5 * (X1+X2);

          Emat[0][0]= 0.0;
          Emat[0][1]= 0.0;
          Emat[1][0]= 0.0;
          Emat[1][1]= 0.0;

          for(ip=0;ip<2;ip++){
            dNdQ[0]= -0.5;
            dNdQ[1]= +0.5;
            XX= X0 + POI[ip]*0.50*DL;
            Area= A1*XX + A2;
            if(Area<= 0.) {
                 fprintf(stderr, "ERROR: Area<0: ¥n");
                 return -1;}
            Jacobi= fabs(dNdQ[0]*X1 + dNdQ[1]*X2);
            Ck= Area*Young/Jacobi;
            Emat[0][0]= Emat[0][0] + Ck * WEI[ip] * dNdQ[0] * dNdQ[0];
            Emat[0][1]= Emat[0][1] + Ck * WEI[ip] * dNdQ[0] * dNdQ[1];
            Emat[1][0]= Emat[1][0] + Ck * WEI[ip] * dNdQ[1] * dNdQ[0];
            Emat[1][1]= Emat[1][1] + Ck * WEI[ip] * dNdQ[1] * dNdQ[1];}
```



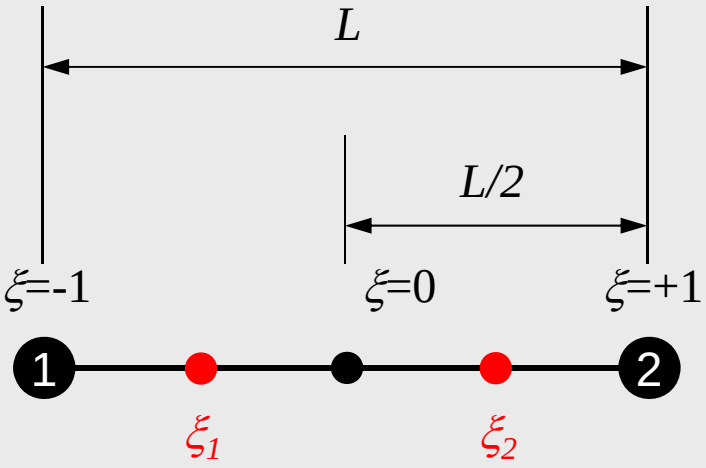
$$|J|_{\xi=\xi_k} = \left|\frac{\partial x}{\partial \xi}\right|_{\xi=\xi_k} = \left|\frac{\partial}{\partial \xi}\left(N_1 x_1 + N_2 x_2\right)\right|_{\xi=\xi_k}$$

$$= \left|\frac{\partial N_1}{\partial \xi} x_1 + \frac{\partial N_2}{\partial \xi} x_2\right|_{\xi=\xi_k}$$

Program: b1.c (5/6)

Element Matrix ~ Global Matrix

```
/*
// +-----------------+
// | MATRIX assemble |
// +-----------------+
*/
      for(icel=0;icel<NE;icel++){
           in1= Icelnod[2*icel];
           in2= Icelnod[2*icel+1];
           X1 = X[in1];
           X2 = X[in2];
           DL = fabs(X2-X1);
           X0 = 0.5 * (X1+X2);

           Emat[0][0]= 0.0;
           Emat[0][1]= 0.0;
           Emat[1][0]= 0.0;
           Emat[1][1]= 0.0;

           for(ip=0;ip<2;ip++){
             dNdQ[0]= -0.5;
             dNdQ[1]= +0.5;
             XX= X0 + POI[ip]*0.50*DL;
             Area= A1*XX + A2;
             if(Area<= 0.) {
                  fprintf(stderr, "ERROR: Area<0: ¥n");
                  return -1;}
             Jacobi= fabs(dNdQ[0]*X1 + dNdQ[1]*X2);
             Ck= Area*Young/Jacobi;
             Emat[0][0]= Emat[0][0] + Ck * WEI[ip] * dNdQ[0] * dNdQ[0];
             Emat[0][1]= Emat[0][1] + Ck * WEI[ip] * dNdQ[0] * dNdQ[1];
             Emat[1][0]= Emat[1][0] + Ck * WEI[ip] * dNdQ[1] * dNdQ[0];
             Emat[1][1]= Emat[1][1] + Ck * WEI[ip] * dNdQ[1] * dNdQ[1];}
```



$$[Emat] = E\sum_{k=1}^{m}\left[w_k \cdot \frac{1}{|J|}\Big|_{\xi=\xi_k} \begin{bmatrix} \frac{\partial N_1}{\partial \xi}\frac{\partial N_1}{\partial \xi} & \frac{\partial N_1}{\partial \xi}\frac{\partial N_2}{\partial \xi} \\ \frac{\partial N_2}{\partial \xi}\frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \xi}\frac{\partial N_2}{\partial \xi} \end{bmatrix}\Big|_{\xi=\xi_k} A(\xi_k) \right]$$

Program: b1.c (6/6)

Boundary Conditions

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

- Gaussian Quadrature
- Isoparametric Elements
- Implementation
- Report #1

Report #1

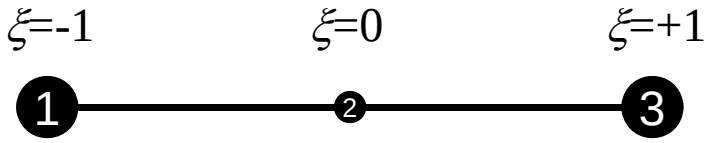
- Implement quadratic interpolation （二次形状関数） on “b1.c/b1.f”. Name of the developed code is “b2.c/b2.f”
- Evaluate accuracy of “b1” and “b2” according to effect of number of meshes.
- Confirm the following equation is correct, if sectional area is constant:

$$\int_V E\left(\frac{d[N]^T}{dx}\frac{d[N]}{dx}\right)dV = \frac{EA}{6L}\begin{bmatrix} +14 & -16 & +2 \\ -16 & +32 & -16 \\ +2 & -16 & +14 \end{bmatrix}$$

- Due on August 18th (M), 2014 at 17:00
- Documents
  - Report (Outline, Results, Discussions) (less than 5 pages)
  - List of Source Code

Report #1: Tips (1)

- Derivative of Shape Functions



$$N_1(\xi)=\frac{1}{2}\xi(-1+\xi)$$
$$N_2(\xi)=(1+\xi)(1-\xi)$$
$$N_3(\xi)=\frac{1}{2}\xi(1+\xi)$$

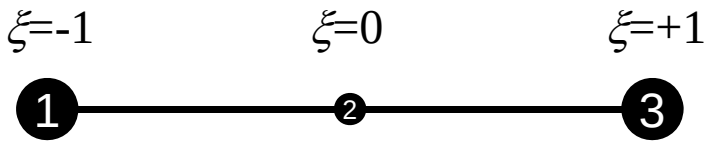
$$\frac{dN_1}{d\xi}=-\frac{1}{2}+\xi$$
$$\frac{dN_2}{d\xi}=-2\xi$$
$$\frac{dN_3}{d\xi}=\frac{1}{2}+\xi$$

$$[Emat]=E\sum_{k=1}^{m}\left[w_k\cdot\frac{1}{|J|}\Big|_{\xi=\xi_k}\begin{bmatrix}\frac{\partial N_1}{\partial\xi}\frac{\partial N_1}{\partial\xi} & \frac{\partial N_1}{\partial\xi}\frac{\partial N_2}{\partial\xi} & \frac{\partial N_1}{\partial\xi}\frac{\partial N_3}{\partial\xi}\\ \frac{\partial N_2}{\partial\xi}\frac{\partial N_1}{\partial\xi} & \frac{\partial N_2}{\partial\xi}\frac{\partial N_2}{\partial\xi} & \frac{\partial N_2}{\partial\xi}\frac{\partial N_3}{\partial\xi}\\ \frac{\partial N_3}{\partial\xi}\frac{\partial N_1}{\partial\xi} & \frac{\partial N_3}{\partial\xi}\frac{\partial N_2}{\partial\xi} & \frac{\partial N_3}{\partial\xi}\frac{\partial N_3}{\partial\xi}\end{bmatrix}\Big|_{\xi=\xi_k}A(\xi_k)\right]$$

Values at Gaussian Quad Points (ξ_k)

Report #1: Tips (2)

- Jacobian



$$\frac{\partial x}{\partial \xi} = \frac{\partial}{\partial \xi}\sum_{i=1}^{3}\left(N_i x_i\right) = \sum_{i=1}^{3}\left(\frac{\partial N_i}{\partial \xi} x_i\right) = \frac{\partial N_1}{\partial \xi} x_1 + \frac{\partial N_2}{\partial \xi} x_2 + \frac{\partial N_3}{\partial \xi} x_3$$

- Jacobian at Gaussian Quad Points (ξ_k)

$$\left.\frac{\partial x}{\partial \xi}\right|_{\xi=\xi_k} = \left.\frac{\partial N_1}{\partial \xi}\right|_{\xi=\xi_k} x_1 + \left.\frac{\partial N_2}{\partial \xi}\right|_{\xi=\xi_k} x_2 + \left.\frac{\partial N_3}{\partial \xi}\right|_{\xi=\xi_k} x_3$$