Chapter 2: A shallow truss element with Fortran computer program

Myoung-Gyu Lee

TA: Gyu Jang Sim (gyujang95@snu.ac.kr)

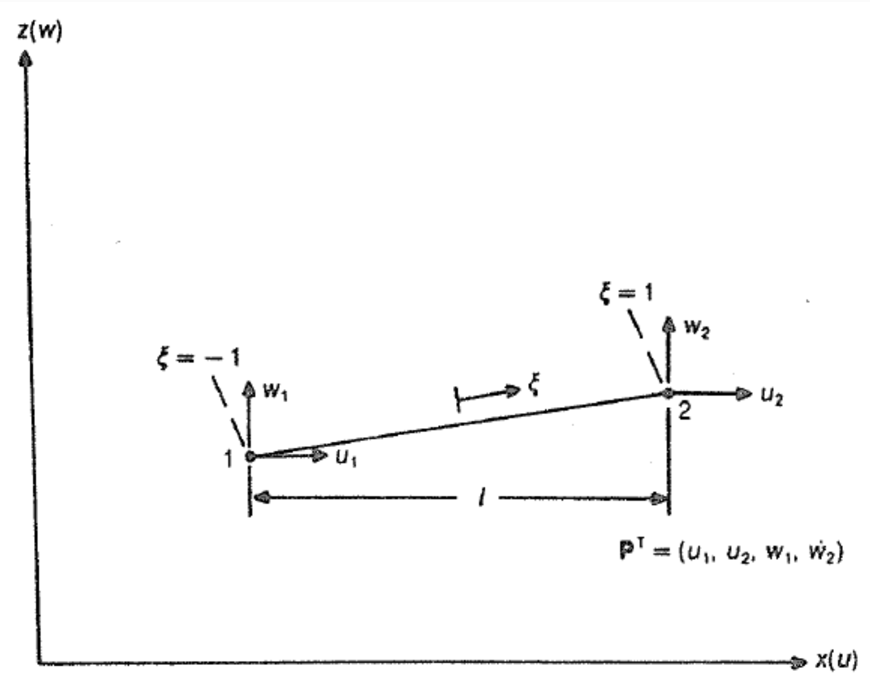
ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY
서 울 대 학 교 공 과 대 학

MAMEL
MATERIALS MECHANICS LABORATORY

In this chapter, we learn

- **computer program for multi-degrees of freedom problem formulated in the form of finite element structure**
- a set of Fortran subroutines
- flowcharts for an ‘incremental formulation’, the ‘Newton-Raphson iterative procedure’, and a combined ‘incremental /iterative technique’.

- **An iso-parametric description** of shallow truss element
  : Displacement and coordinate share the same shape function.



[Fig 2.1 A shallow truss element]

$$x = \frac{1}{2}\begin{bmatrix}1-\xi & 1+\xi\end{bmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix}$$

$$z = \frac{1}{2}\begin{bmatrix}1-\xi & 1+\xi\end{bmatrix}\begin{pmatrix}z_1\\ z_2\end{pmatrix}$$

$$u = \frac{1}{2}\begin{bmatrix}1-\xi & 1+\xi\end{bmatrix}\begin{pmatrix}u_1\\ u_2\end{pmatrix}$$

$$w = \frac{1}{2}\begin{bmatrix}1-\xi & 1+\xi\end{bmatrix}\begin{pmatrix}w_1\\ w_2\end{pmatrix}$$

[eq. 2.1,2.2]

Shape functions
or, interpolation functions

ξ : **parent domain**, x, z, u, w : **spatial domain**

- Strain can be derived in the iso-parametric formulation.

$$\varepsilon = -\frac{u}{l} + \left(\frac{z}{l}\right)\left(\frac{w}{l}\right) + \frac{1}{2}\left(\frac{w}{l}\right)^2$$ [eq. 1.51]

$$\frac{du}{dx} = \frac{du}{d\xi}\frac{d\xi}{dx} = \frac{u_2 - u_1}{l} = \frac{u_{21}}{l}$$ [eq. 2.5]

$$\varepsilon = \frac{du}{dx} + \left(\frac{dz}{dx}\right)\left(\frac{dw}{dx}\right) + \frac{1}{2}\left(\frac{dw}{dx}\right)^2$$ [eq. 2.3]

$$\frac{dw}{dx} = \frac{w_{21}}{l} \quad \frac{dz}{dx} = \frac{z_{21}}{l}$$ [eq. 2.6]

$$\frac{dx}{d\xi} = \frac{x_2 - x_1}{2} = \frac{l}{2}$$ [eq. 2.4]

$$\Rightarrow \boxed{\varepsilon = \frac{u_{21}}{l} + \left(\frac{z_{21}}{l}\right)\left(\frac{w_{21}}{l}\right) + \frac{1}{2}\left(\frac{w_{21}}{l}\right)^2}$$ [eq. 2.7]

- Virtual displacement brings change in strain:

[eq. 2.9]

$$\delta\varepsilon_v = \frac{d\delta u_v}{dx} + \left(\frac{dz}{dx} + \frac{dw}{dx}\right)\frac{d\delta w_v}{dx} + \frac{1}{2}\left(\frac{d\delta w_v}{dx}\right)^2$$ where $\delta\varepsilon_v = \varepsilon(u+\delta u_v, w+\delta w_v) - \varepsilon(u,w)$

- Using previous relations,

and $\left[\dfrac{du}{dx}\right]_{u+\delta u_v} = \dfrac{du}{dx} + \dfrac{d\delta u_v}{dx}$

$$\frac{du}{dx} = \frac{u_{21}}{l}, \quad \frac{dw}{dx} = \frac{w_{21}}{l}, \quad \frac{dz}{dx} = \frac{z_{21}}{l}$$ [eq. 2.5, 2.6]

$$\delta\varepsilon_v = \frac{1}{l}\delta u_{v21} + \frac{1}{l^2}(z_{21} + w_{21})\delta w_{v21} + \frac{1}{2l^2}\delta w_{v21}^2$$ [eq. 2.10]

- (Virtual) strain is inner product of **strain interpolation matrix** $\mathbf{b}$ and (virtual) **nodal displacement** $\delta\mathbf{p}_v$

$$\delta\mathbf{p}_v = \begin{pmatrix} \delta u_{v1} \\ \delta u_{v2} \\ \delta w_{v1} \\ \delta w_{v2} \end{pmatrix}, \quad \delta\varepsilon_v = \frac{1}{l}\delta u_{v21} + \frac{1}{l^2}(z_{21} + w_{21})\delta w_{v21} = \mathbf{b}^T \delta\mathbf{p}_v$$ [eq. 2.11, 2.12]

$\Rightarrow$ $$\mathbf{b} = \frac{1}{l}\begin{pmatrix} -1 \\ 1 \\ -\beta \\ \beta \end{pmatrix}$$ where $$\beta = \frac{z_{21} + w_{21}}{l}$$ [eq. 2.13, 2.14]

- **Discretization** applied to **weak form** derived from **principle of virtual work**

$$V = \int \sigma \delta \varepsilon_v dV - \delta \mathbf{p}_v^T \mathbf{q}_e = 0$$

[eq. 2.15-2.17]

$$= \delta \mathbf{p}_v^T \mathbf{g} = \delta \mathbf{p}_v^T (\mathbf{q}_i - \mathbf{q}_e) = \delta \mathbf{p}_v^T \left(\int \sigma \mathbf{b} dV - \mathbf{q}_e \right)$$

$$\mathbf{q}_i = \int \sigma \mathbf{b} dV = Nl\mathbf{b}$$

Internal force vector

- For equilibrium, V=0 for any virtual displacements or **g**=0
- Tangent stiffness matrix:

$$\mathbf{K}_t = \frac{\partial \mathbf{g}}{\partial \mathbf{p}} = \frac{\partial \mathbf{q}_i}{\partial \mathbf{p}} = l\mathbf{b}\frac{dN}{d\mathbf{p}} + lN\frac{\partial \mathbf{b}}{\partial \mathbf{p}}$$

$$= l\mathbf{b}\frac{dN}{d\varepsilon}\frac{d\varepsilon}{d\mathbf{p}} + lN\frac{\partial \mathbf{b}}{\partial \mathbf{p}}$$

$$= EAl\mathbf{b}\mathbf{b}^T + lN\frac{\partial \mathbf{b}}{\partial \mathbf{p}}$$

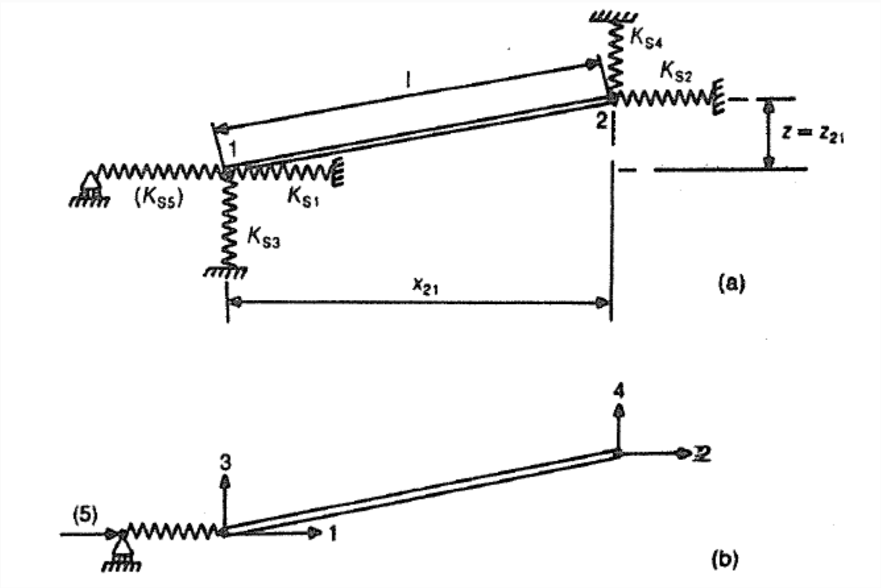
$$= \frac{EA}{l}\begin{bmatrix} 1 & -1 & \beta & -\beta \\ -1 & 1 & -\beta & \beta \\ \beta & -\beta & \beta^2 & -\beta^2 \\ -\beta & \beta & -\beta^2 & \beta^2 \end{bmatrix} + \frac{N}{l}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

$$\mathbf{b} = \frac{1}{l}\begin{pmatrix} -1 \\ 1 \\ -\beta \\ \beta \end{pmatrix}$$

[eq. 2.19-2.23]

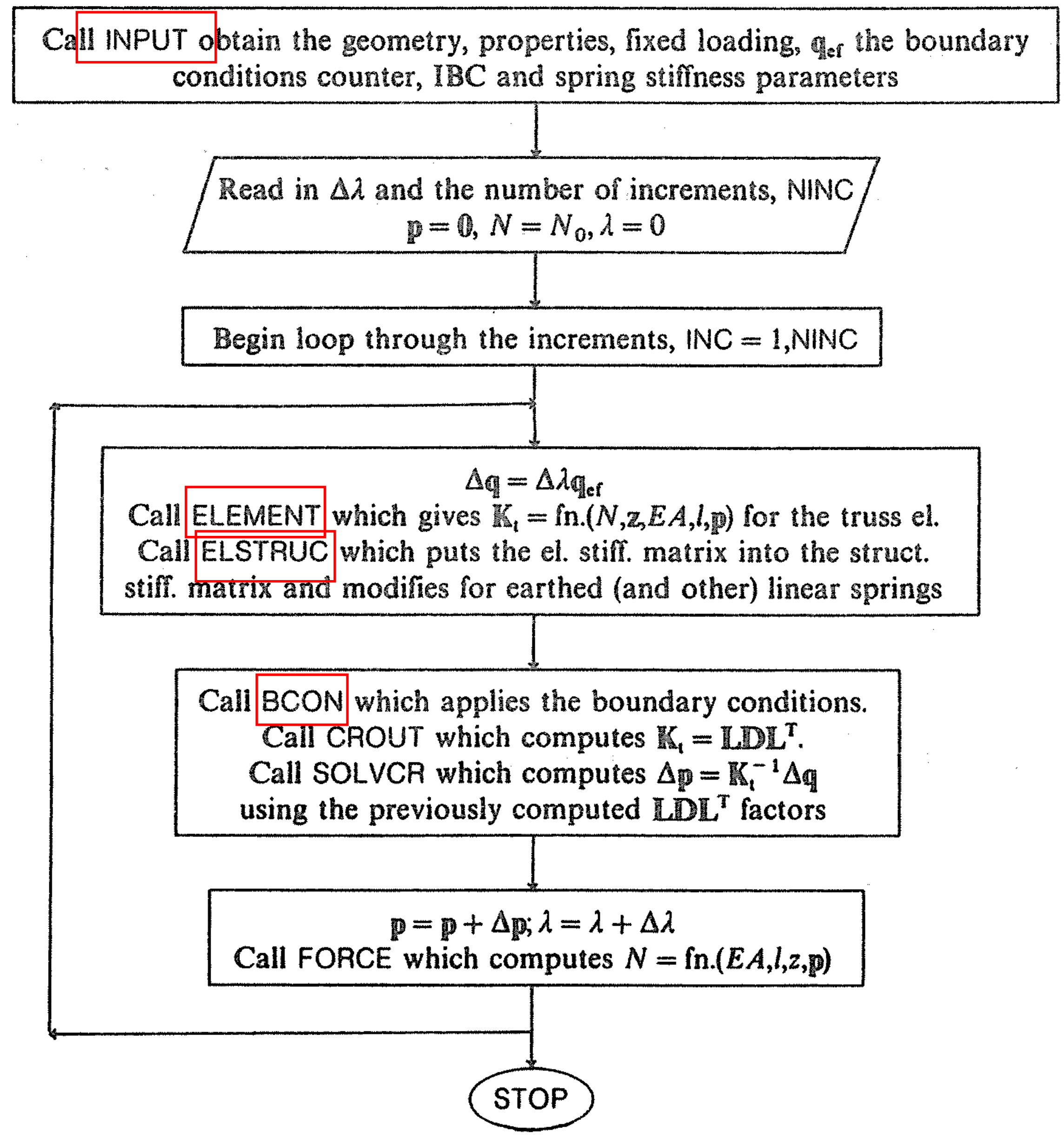
- Fortran subroutines are provided to solve general form of bar-spring system.
  - There are ‘earthed springs’ and a horizontal linear spring.
  - In many cases, the horizontal linear spring K_{s5} can be omitted.

- For general solution procedure (**assembling**, **boundary conditions**, etc) of finite element method, refer to: Daryl L. Logan, “A First Course in the Finite Element Method” – Ch 1.
- Quick introduction to Fortran77 :
  - http://seismic.yonsei.ac.kr/fortran/index.html (kor)
  - https://web.stanford.edu/class/me200c/tutorial_77 (eng)
- There might some typos or errors in the code.



[Fig 2.2 Bar-spring system]
(a) Bar element with springs (b) variables

- **Example of algorithm**



2.2.1 Subroutine ELEMENT

- This subroutine calculates
  - an **internal force vector**
  - **an element tangent stiffness matrix**

$$\mathbf{q}_i = Nl\mathbf{b} = N\begin{pmatrix} -1 \\ 1 \\ -\beta \\ \beta \end{pmatrix} = \texttt{FI(4)}$$

[eq. 2.17] [variable in fortran]

$$\mathbf{K}_t = \frac{EA}{l}\begin{bmatrix} 1 & -1 & \beta & -\beta \\ -1 & 1 & -\beta & \beta \\ \beta & -\beta & \beta^2 & -\beta^2 \\ -\beta & \beta & -\beta^2 & \beta^2 \end{bmatrix} + \frac{N}{l}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} = \texttt{AKT(4,4)}$$

[eq. 2.23] [variable in fortran]

where $\beta = \dfrac{z_{21} + w_{21}}{l}$

2.2.1 Subroutine ELEMENT

```
      SUBROUTINE ELEMENT(FI,AKT,AN,X,Z,P,E,ARA,AL,IWRIT,IWR,IMOD,
     1                   IDUM,ADUM1,ADUM2)
C     ARGUMENTS IN LINE ABOVE AND ARRAY X NOT USED FOR SHALLOW TRUSS
C
C
C     FOR SHALLOW TRUSS ELEMENT
C     IMOD = 1 COMPUTES INT.LD.VECT.FI
C     IMOD = 2 COMPUTES TAN.STIFF.AKT
C     IMOD = 3 COMPUTES BOTH
C
C     AN = INPUT (TOTAL FORCE IN BAR)
C     Z = INPUT (Z COORD VECTOR)
C     P = INPUT (TOTAL DISP.VECTOR)
C     AL = INPUT (LENGTH OF ELEMENT)
C     EA = INPUT (YOUNGS MODULUS)
C     ARA = INPUT (AREA OF ELEMENT)
C
C     IF IWRIT.NE.0(NOT EQUAL TO 0) WRITES OUT FI AND/OR AKT ON CHANNEL IWR
C
      DOUBLE PRECISION AKT(4,4),FI(4),Z(2),P(4),X(2),EA,E,ARA,EAL,AL,
     1                 Z21,W21,BET,AN,ANL,ADUM1,ADUM2
      INTEGER I,J,IDUM,IMOD,IWR,IWRIT

C
      EA = E*ARA
      EAL = EA/AL
      Z21 = Z(2) - Z(1)
      W21 = P(4) - P(3)
      BET = (Z21 + W21)/AL
C
```

AN ⋯ N

Z ⋯ $\begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$

P ⋯ $\begin{pmatrix} u_1 \\ u_2 \\ w_1 \\ w_2 \end{pmatrix}$

AL ⋯ L

EA ⋯ E

ARA ⋯ A

EA = E*ARA ⋯ EA

EAL = EA/AL ⋯ $\dfrac{EA}{L}$

Z21 ⋯ z_{21}

W21 ⋯ w_{21}

BET ⋯ $\beta = \dfrac{z_{21} + w_{21}}{l}$

code typed by Jaehyun You

2.2.1 Subroutine ELEMENT

```
C
      IF (IMOD.NE.2) THEN
C     COMPUTES INT.FORCE.VECT (SEE 2.17)
        FI(1) = -1.D0
        FI(2) = -1.D0
        FI(3) = -BET
        FI(4) = BET
        DO 1 I=1,4
          FI(I) = AN*FI(I)
    1   CONTINUE
        IF (IWRIT.NE.0) THEN
          WRITE (IWR,1000) (FI(I),I=1,4)
 1000     FORMAT(/,1X,'INT.FORCE VECT.FOR TRUSS EL IS',1X,4G13.5,/)
        ENDIF
C
      ENDIF
```

$\cdots\cdots\cdots\cdots\cdots\cdots$ *compute* q_i

$$\mathbf{q}_i = N l \mathbf{b} = N \begin{pmatrix} -1 \\ 1 \\ -\beta \\ \beta \end{pmatrix}$$

code typed by Jaehyun You

2.2.1 Subroutine ELEMENT

```
      IF(IMOD.NE.1) THEN
C     COMPUTES TAN STIFF.MATRIX(UPPER STRIANGLE) (SEE 2.23)
        AKT(1,1) = 1.D0
        AKT(1,2) = -1.D0
        AKT(1,3) = BET
        AKT(1,4) = -BET
        AKT(2,2) = 1.D0
        AKT(2,3) = -BET
        AKT(2,4) = BET
        AKT(3,3) = BET*BET
        AKT(3,4) = -AKT(3,3)
        AKT(4,4) = BET*BET
        DO 12 I=1,4
          DO 13 J=1,4
            AKT(I,J) = EAL*AKT(I,J)
   13     CONTINUE
   12   CONTINUE
C
C           NOW ADD GEOM. OR INIT STRESS MATRIX (SEE 2.23)
C
        ANL = AN/AL
        AKT(3,3) = AKT(3,3) + ANL
        AKT(3,4) = AKT(3,4) - ANL
        AKT(4,4) = AKT(3,3) + ANL
        IF (IWRIT.NE.0) THEN
          WRITE (IWR,1001)
 1001     FORMAT (/, 1X, 'TAN.STIFF.MATRIX FOR TRUSS EL. IS', /)
          DO 14 I = 1,4
            WRITE (IWR,67) (AKT(I,J),J=1,4)
   67       FORMAT (1X,7G13.5)
   14     CONTINUE
        ENDIF
C
      ENDIF
```

compute $\mathbf{K}_t$

$$\mathbf{K}_t = \frac{EA}{l}\begin{bmatrix} 1 & -1 & \beta & -\beta \\ -1 & 1 & -\beta & \beta \\ \beta & -\beta & \beta^2 & -\beta^2 \\ -\beta & \beta & -\beta^2 & \beta^2 \end{bmatrix} + \frac{N}{l}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

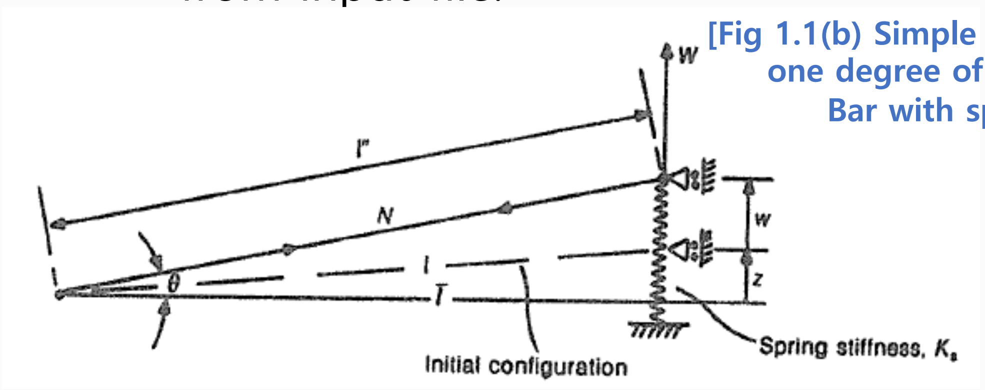
$\frac{N}{L}$

$$\mathbf{K}_t = \frac{EA}{l}\begin{bmatrix} 1 & -1 & \beta & -\beta \\ -1 & 1 & -\beta & \beta \\ \beta & -\beta & \beta^2 & -\beta^2 \\ -\beta & \beta & -\beta^2 & \beta^2 \end{bmatrix} + \frac{N}{l}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

code typed by Jaehyun You

2.2.2 Subroutine INPUT

- This subroutine reads: 1. geometry 2. properties 3. boundary conditions 4. loading from input file.



[Fig 1.1(b) Simple problem with one degree of freedom. Bar with spring]

*

? ? ?

$$\mathbf{q}_e^T = (\boxed{U_1}\ \boxed{U_2}\ \boxed{W_1}\ \boxed{W_2})$$

$$\texttt{IBC} = (1\quad -1\quad 1\quad 0)$$

input (known)

$$\mathbf{p}^T = (\boxed{u_1}\ \boxed{u_2}\ \boxed{w_1}\ \boxed{w_2})$$

0 not 0 0 ?

output (unknown)

$$\texttt{QFI} = (u_1(=0)\quad u_2(\neq 0)\quad w_1(=0)\quad W_2)$$

`4 50000000. 2500. 0.`	`NV, EA, AL, ANIT`	$d.o.f\quad E\quad L\quad N_0$ (initial internal force in bar)
`0. 25.`	`Z`	$(z_1\quad z_2)$
`0. 0. 0. -7.`	`QFI`	$\mathbf{q}_e^T, \mathbf{p}^T$ combined (*)
`1 1 1 0`	`IBC`	boundary condition information (*)
`1`	`NDSP`	number of earthed springs
`4`	`ID14S`	id(s) of earthed springs (Fig. 2.2(a))
`1.35`	`AK14S`	stiffness(es) of earthed springs (Fig. 2.2(a))
	`AK15`(only if NV = 5)	K_{s5} (Fig. 2.2(a))

[input file of Fig 1.1(b)]

[corresponding variables]

2.2.2 Subroutine INPUT

```fortran
      SUBROUTINE INPUT(E,ARA,AL,QFI,X,Z,ANIT,IBC,IRE,IWR,AK14S,ID14S,
     1                NDSP,NV,AK15,
     2                ADUM1,IDUM)
C     ARGUMENTS IN LINE ABOVE AND ARRAY X NOT USED FOR SHALLOW TRUSS
C
C     READS INPUT FOR TRUSS ELEMENT
C
      DOUBLE PRECISION E,ARA,AL,QFI(NV),X(2),Z(2),ANIT,AK14S(4),AK15,
     1                 ADUM1
      INTEGER NV,IDUM,I,NDSP,ID14S(4),IBC(NV)
C
      READ(IRE,*) NV,EA,AL,ANIT
      E = EA
      ARA = 1.D0
      WRITE(IWR,1000) NV,EA,AL,ANIT
 1000 FORMAT(/,1X,'NV=NO. OF VARBLS.=',G13.5,/,1X,
     1      'EA=',G13.5,/,1X,
     2      'AL=EL.LENGTH=',G13.5,1X,
     3      'ANIT=INIT.FORCE=',G13.5,/)
      IF (NV.NE.4.AND.NV.NE.5) STOP 'INPUT 1000'

      READ(IRE,*) Z(1),Z(2)
      WRITE(IWR,1001) Z(1),Z(2)
 1001 FORMAT(/,1X,'Z CO-ORD OF NODE 1=',G13.5,1X,
     1      'Z CO-ORD OF NODE 2=',G13.5,/)

      READ(IRE,*) (QFI(I),I=1,NV)
      WRITE(IWR,1002) (QFI(I),I=1,NV)
 1002 FORMAT(/,1X,'FIXED LOAD OR DISP.VECTOR, QFI=',/,1X,5G13.5,/)
      WRITE(IWR,1008)
 1008 FORMAT(/,1X,'IF IBC(I)-SEE BELOW-=0, VARIABLE=A LOAD',/,1X,
     2       'IF IBC(I)-SEE BELOW-=-1, VARIABLE=A DISP.',/)

      READ(IRE,*) (IBC(I),I=1,NV)
      WRITE(IWR,1003) (IBC(I),I=1,NV)
 1003 FORMAT(/,1X,'BOUND.COND.COUNTER, IBC',/,1X,
     1       '=0, FREE:=1, REST.TO ZERO:=-1 REST.TO NON-ZERO',/,
     2       1X,5G13.5,/)

      READ(IRE,*) NDSP
      IF (NDSP.NE.0) THEN
        READ(IRE,*) (ID14S(I),I=1,NDSP)
        READ(IRE,*) (AK14S(I),I=1,NDSP)
        DO 40 I=1,NDSP
          WRITE(IWR,1004) AK14S(I), ID14S(I)
 1004     FORMAT(/,1X,'LINEAR SPRING OF STIFFNESS',G13.5,/,1X,
     1           'ADDED AT VAR.NO.',G13.5,/)
   40   CONTINUE
      ENDIF
C
      IF (NV.EQ.5) THEN

        READ(IRE,*) AK15
        WRITE(IWR,1005) AK15
 1005   FORMAT(/,1X,'LINEAR SPRING BETWEEN VARBLS. 1 AND 5 OF STIFF ',
     1         G13.5,/)

      ENDIF
C
      RETURN
      END
```

2.2.3 Subroutine FORCE

- This subroutine computes the axial force N in the bar.

$$N = EA\varepsilon = EA\left[\frac{u_{21}}{l} + \left(\frac{z_{21}}{l}\right)\left(\frac{w_{21}}{l}\right) + \frac{1}{2}\left(\frac{w_{21}}{l}\right)^2\right]$$

[eq. 2.7, 2.8]

```fortran
      SUBROUTINE FORCE(AN,ANIT,E,ARA,AL,X,Z,P,IWRIT,IWR,
     1                 ITUM,ADUM1,ADUM2,ADUM3)
C     ARGUMENTS IN LINE ABOVE AND ARRAY X NOT USED FOR SHALLOW TRUSS
C
C     COMPUTES INTERNAL.FORCE IN A SHALLOW TRUSS ELEMENT
C     USING (2.7) AND (2.8)
      DOUBLE PRECISION Z(2),P(4),X(2),AN,ANIT,E,ARA,AL,ADUM1,ADUM2,
     1                 ADUM3,EA,EAL,U21,W21,Z21
      INTEGER IWRIT,IWR

C
      EA = E*ARA
      EAL = EA/AL
      U21 = P(2) - P(1)
      W21 = P(4) - P(3)
      Z21 = Z(2) - Z(1)
      AN = U21 + (Z21*W21/AL) + 0.5D0*(W21*W21*AL)
      AN = EAL*AN + ANIT
      IF (IWRIT.NE.0) WRITE (IWR,1000) AN
 1000 FORMAT(/,1X,'AXIAL FORCE AN= ',G13.5/)
      RETURN
      END
```

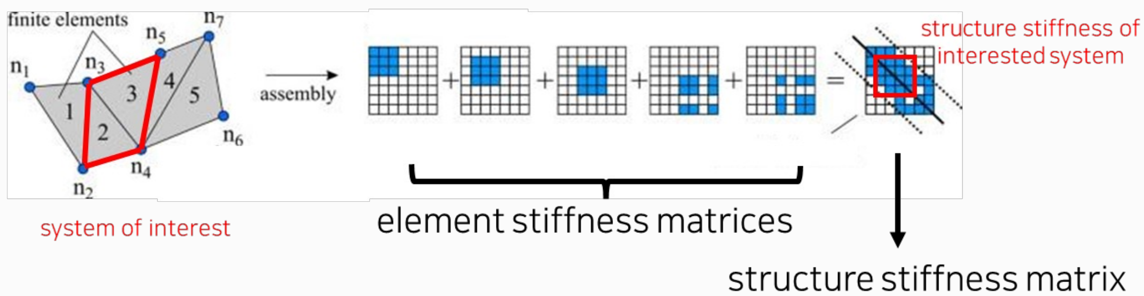
$$N = EA\left[\frac{u_{21}}{l} + \left(\frac{z_{21}}{l}\right)\left(\frac{w_{21}}{l}\right)\right]$$

typo

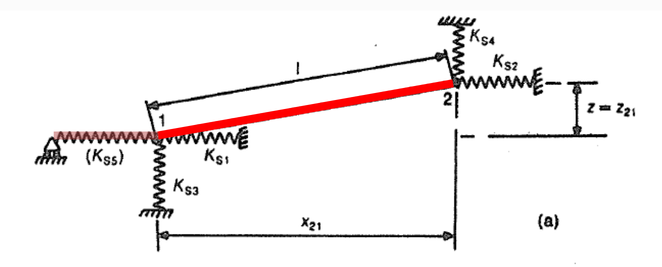
initial internal force N_0

2.2.4 Subroutine ELSTRUC

- This subroutine puts the **element stiffness matrix** `AKTE(4,4)` into **structure stiffness matrix** `AKTS(NV,NV)` (NV=4 or 5)
  - Adds in the 'earthed springs' (if number of spring > 0)
  - Adds in the linear spring between variables 1 and 5 (if NV = 5)



[Element assembly]



system of interest

$$\mathbf{K}_t = \frac{EA}{l}\begin{bmatrix} 1 & -1 & \beta & -\beta \\ -1 & 1 & -\beta & \beta \\ \beta & -\beta & \beta^2 & -\beta^2 \\ -\beta & \beta & -\beta^2 & \beta^2 \end{bmatrix} + \frac{N}{l}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

$$\mathbf{K}_{spring} = \begin{bmatrix} K_s & -K_s \\ -K_s & K_s \end{bmatrix}$$

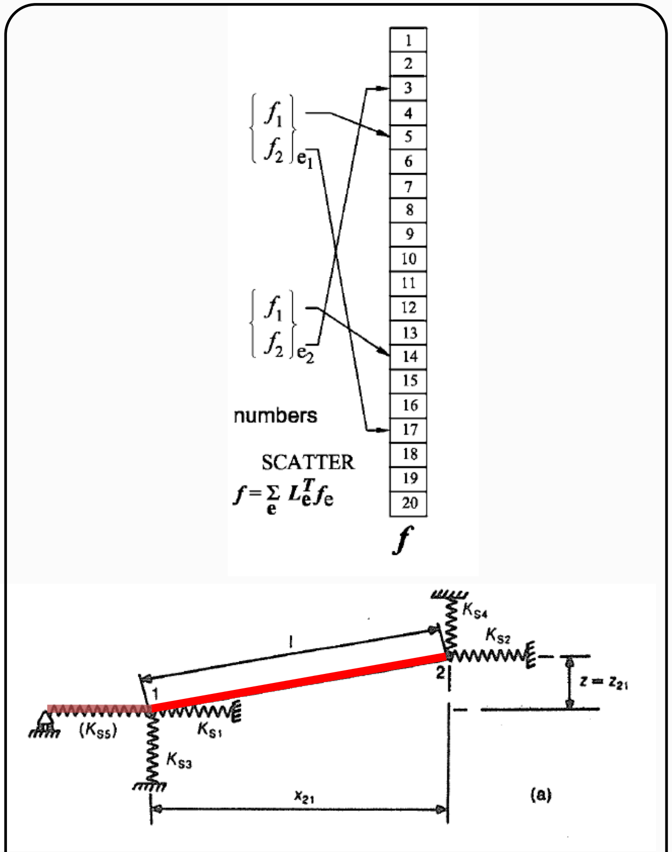
element stiffness matrices

$$\mathbf{K}_{struct} = \mathbf{K}_t + \begin{bmatrix} K_{s1} & 0 & 0 & 0 \\ 0 & K_{s2} & 0 & 0 \\ 0 & 0 & K_{s3} & 0 \\ 0 & 0 & 0 & K_{s4} \end{bmatrix}$$

structured stiffness matrix (NV=4)

2.2.4 Subroutine ELSTRUC

- This subroutine scatters internal force vector
  - adds in the 'earthed springs' (if number of spring > 0)



system of interest

$$\mathbf{q}_i = Nl\mathbf{b} = N\begin{pmatrix} -1 \\ 1 \\ -\beta \\ \beta \end{pmatrix}$$

$$\mathbf{q}_{i,spring} = \begin{bmatrix} K_s & -K_s \\ -K_s & K_s \end{bmatrix}\begin{pmatrix} u_{s1} \\ u_{s2} \end{pmatrix} = K_s u_{21}\begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

element internal force vectors

$$\mathbf{q}_{i,struct} = \mathbf{q}_i + \begin{pmatrix} K_{s1}u_1 \\ K_{s2}w_1 \\ K_{s3}u_2 \\ K_{s4}w_2 \end{pmatrix}$$

structured internal force vector
(NV=4)

2.2.4 Subroutine ELSTRUC

```
      SUBROUTINE ELSTRUC(AKTE,AKTS,NV,AK15,ID14S,AK14S,NDSP,FI,PT,
     1                    IMOD,IWRIT,IWR)
C
C     FOR IMOD=2 OR 3
C     PUTS EL-STIFF MATRIX AKTE(4,4) INTO STRUCT.STIFF AKTS(NV,NV)
C     IF NV = 5, ALSO ADDS IN LINEAR SPRING AK15 BETWEEN VARBLS.1&5
C     ALSO ADDS IN NDSP EARTHED LINEAR SPRINGS FOR VARBLS.1-4
C     USING PROPERTIES IN AK14S(4) AND DEGS.OF F.IN IDSPS(4)
C     THROUGHOUT ONLY WORKS WITH UPPER TRIANGLE
C     FOR IMOD=1 OR 3
C     MODIFIES INTERNAL FORCE VECT., FI TO INCLUDE EFFECTS FROM
C     ARIOUS LINEAR SPRINGS USING TOTAL DISPS., PT.
C
      DOUBLE PRECISION AKTE(4,4),AKTS(NV,NV),FI(NV),PT(NV),AK14S(4),
     1                 AK15
      INTEGER  ID14S(4),NV,NDSP,IMOD,IWRIT,IWR,I,J
C
      IF (IMOD.NE.2) THEN
C     MODIFY FORCES
        IF (INDSP.NE.0) THEN
C     FOR EARTHED SPRINGS
          DO 40 I=1,NDSP
            IDS = ID14S(I)
            FI(IDS) = FI(IDS) + AK14S(I)*PT(IDS)
   40     CONTINUE
        ENDIF
C
        IF (IWRIT.NE.0) WRITE (IWR,1002) FI
 1002   FORMAT(/,1X,'STR.INT.FORCE VECT IS',1X,5G13.5,/)
C
      ENDIF
```

$$\mathbf{q}_{i,struct} = \mathbf{q}_i + \begin{pmatrix} K_{s1}u_1 \\ K_{s2}w_1 \\ K_{s3}u_2 \\ K_{s4}w_2 \end{pmatrix}$$

2.2.4 Subroutine ELSTRUC

```
         IF (IMOD.NE.1) THEN
C        WORK ON STIFFNESS MATRIX; CLEAR STRUCT.STIFFNESS MATRIX
           DO 10 I=1,NV
             DO 11 J=1,NV
               AKTS(I,J) = 0.D0
   11        CONTINUE
   10      CONTINUE
C
C        INSERT EL.STIFFNESS MATRIX
           DO 20 I=1,4
             DO 21 J=1,4
               AKTS(I,J) = AKTE(I,J)
   21        CONTINUE
   20      CONTINUE
C
C        SPRING BETWEEN VARBLS.1&5
           IF (NV.EQ.5) THEN
             AKTS(1,1) = AKTS(1,1) + AK15
             AKTS(1,5) = AKTS(1,5) - AK15
             AKTS(5,5) = AKTS(5,5) + AK15
           ENDIF
C
C          EARTHED SPRINGS FOR VARBLS.1-4
           IF (NDSP.NE.0) THEN
             DO 30 I=1,NDSP
               IDS = ID14S(I)
               AKTS(IDS,IDS) = AKTS(IDS,IDS) + AK14S(I)
   30        CONTINUE
           ENDIF
C
```

$$\mathbf{K}_{struct} = \mathbf{K}_t + \begin{bmatrix} K_{s5} & 0 & 0 & 0 & -K_{s5} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ K_{s5} & 0 & 0 & 0 & -K_{s5} \end{bmatrix} \quad \cdots\cdots(\text{d.o.f} = 5)$$

$$\mathbf{K}_{struct} = \mathbf{K}_t + \begin{bmatrix} K_{s1} & 0 & 0 & 0 \\ 0 & K_{s2} & 0 & 0 \\ 0 & 0 & K_{s3} & 0 \\ 0 & 0 & 0 & K_{s4} \end{bmatrix} \quad \cdots\cdots(\text{d.o.f} = 4)$$

Boundary value problem – from finite-difference method

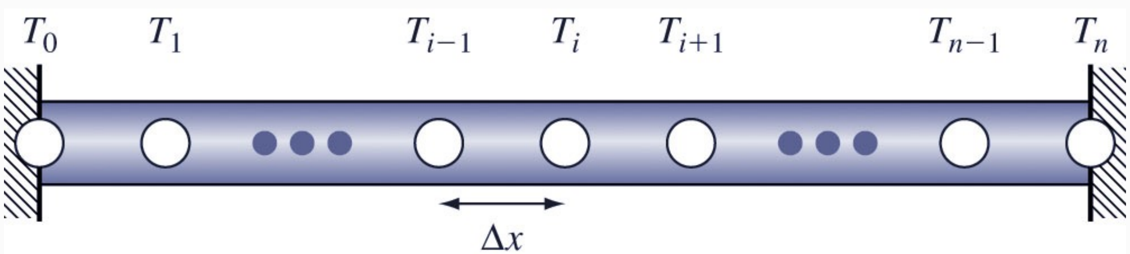
$$\frac{d^2T}{dx^2} + h'(T_\infty - T) = 0$$

$$\frac{d^2T}{dx^2} = \frac{T_{i-1} - 2T_i + T_{i+1}}{\Delta x^2}$$

$$\frac{T_{i-1} - 2T_i + T_{i+1}}{\Delta x^2} + h'\left(T_\infty - T_i\right) = 0$$

$$-T_{i-1} + \left(2 + h'\Delta x^2\right)T_i - T_{i+1} = h'\Delta x^2 T_\infty$$

Example)



Finite-Difference Example (cont)

- Since T_0 and T_n are known, they will be on the right-hand-side of the linear algebra system (in this case, in the first and last entries, respectively):

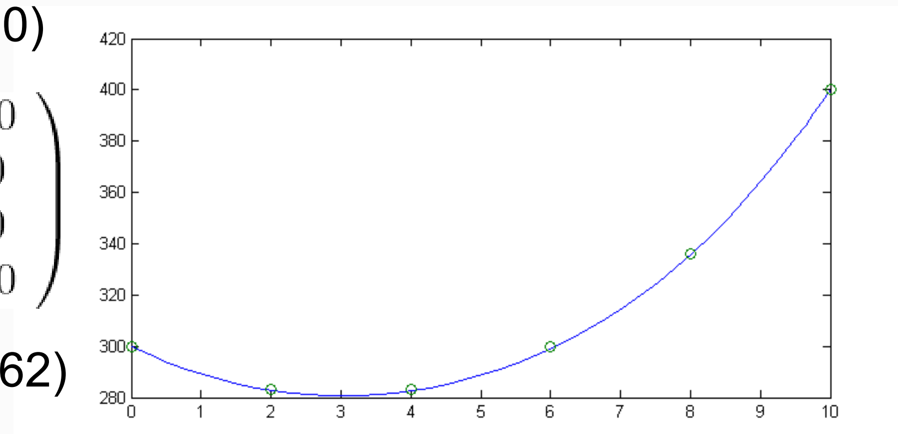
$$\begin{bmatrix} 2+h'\Delta x^2 & -1 & & \\ -1 & 2+h'\Delta x^2 & -1 & \\ & \ddots & \ddots & \ddots \\ & & -1 & 2+h'\Delta x^2 \end{bmatrix} \begin{Bmatrix} T_1 \\ T_2 \\ \vdots \\ T_{n-1} \end{Bmatrix} = \begin{Bmatrix} h'\Delta x^2 T_\infty + T_0 \\ h'\Delta x^2 T_\infty \\ \vdots \\ h'\Delta x^2 T_\infty + T_n \end{Bmatrix}$$

Tridiagonal matrix

Ex) $\Delta x = 2\text{m}$ $T_0(=300), T_1, T_2, T_3, T_4, T_5(=400)$

$$\begin{pmatrix} 2.2 & -1 & 0 & 0 \\ -1 & 2.2 & -1 & 0 \\ 0 & -1 & 2.2 & -1 \\ 0 & 0 & -1 & 2.2 \end{pmatrix} \begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{pmatrix} = \begin{pmatrix} 340 \\ 40 \\ 40 \\ 440 \end{pmatrix}$$

T=(283.2660, 283.1853, 299.7415, 336.2462)



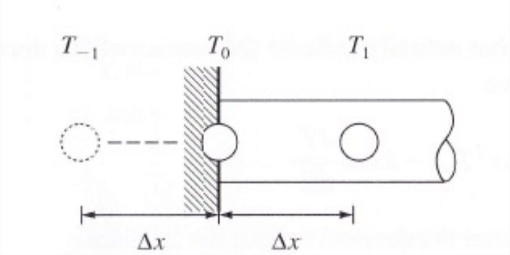
✓ Two ways to improve the numerical solution.

Derivative Boundary Conditions

- Neumann boundary conditions are resolved by solving the centered difference equation at the point and rewriting the system equation accordingly.

- For example, if there is a Neumann condition at the T_0 point,

$$\left.\frac{dT}{dx}\right|_0 = \frac{T_1 - T_{-1}}{2\Delta x} \Rightarrow T_{-1} = T_1 - 2\Delta x\left(\left.\frac{dT}{dx}\right|_0\right)$$



$$-T_{-1} + \left(2 + h'\Delta x^2\right)T_0 - T_1 = h'\Delta x^2 T_\infty$$

$$-\left[T_1 - 2\Delta x\left.\frac{dT}{dx}\right|_0\right] + \left(2 + h'\Delta x^2\right)T_0 - T_1 = h'\Delta x^2 T_\infty$$

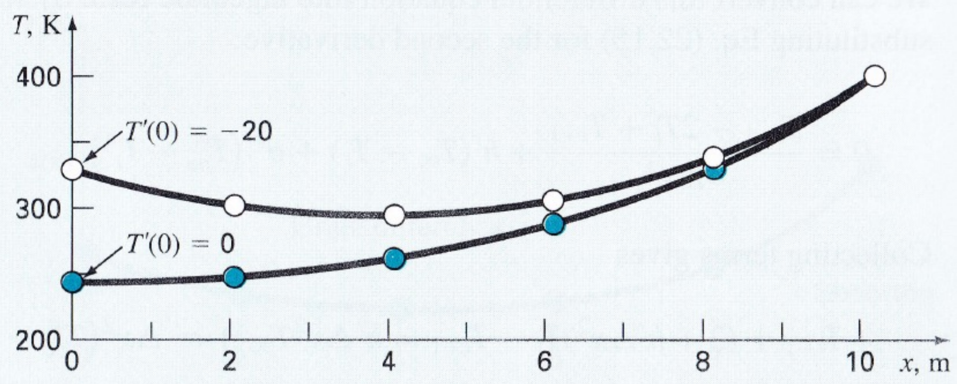
$$\left(2 + h'\Delta x^2\right)T_0 - 2T_1 = h'\Delta x^2 T_\infty - 2\Delta x\left(\left.\frac{dT}{dx}\right|_0\right)$$

Example of derivative boundary condition

$T_a' = 0$ & $T_b = 400$ K, T_inf = 200 K

$2.2T_0 - 2T_1 = 40$, $-T_0 + 2.2T_1 - T_2 = 40$, Eqs for other nodes are the same.

$$\begin{bmatrix} 2.2 & -2 & & & \\ -1 & 2.2 & -1 & & \\ & -1 & 2.2 & -1 & \\ & & -1 & 2.2 & -1 \\ & & & -1 & 2.2 \end{bmatrix} \begin{Bmatrix} T_0 \\ T_1 \\ T_2 \\ T_3 \\ T_4 \end{Bmatrix} = \begin{Bmatrix} 40 \\ 40 \\ 40 \\ 40 \\ 440 \end{Bmatrix} \longrightarrow \begin{aligned} T_0 &= 243.0278 \\ T_1 &= 247.3306 \\ T_2 &= 261.0994 \\ T_3 &= 287.0882 \\ T_4 &= 330.4946 \end{aligned}$$



2.2.5 Subroutine BCON

- This subroutine converts **constrained displacement** into **external force**. (load control)

[eq. 2.25]

$$\mathbf{q} = \begin{pmatrix}\mathbf{q}_f\\ \mathbf{q}_p\end{pmatrix} = \mathbf{Kp} = \begin{bmatrix}\mathbf{K}_{ff} & \mathbf{K}_{fp}\\ \mathbf{K}_{pf} & \mathbf{K}_{pp}\end{bmatrix}\begin{pmatrix}\mathbf{p}_f\\ \mathbf{p}_p\end{pmatrix}$$

'f' free
'p' prescribed

Ordering can be changed in the matrix

(Case 1) For the prescribed displacement
$\mathbf{p}_p$=0 (IBC(i)=1)

$$\mathbf{q} = \begin{pmatrix}\mathbf{q}_f\\ \cancelto{0}{\mathbf{q}_p}\end{pmatrix} = \mathbf{Kp} = \begin{bmatrix}\mathbf{K}_{ff} & \cancelto{0}{\mathbf{K}_{fp}}\\ \cancelto{0}{\mathbf{K}_{pf}} & \cancelto{\mathbf{I}}{\mathbf{K}_{pp}}\end{bmatrix}\begin{pmatrix}\mathbf{p}_f\\ \mathbf{p}_p\end{pmatrix}$$

Dummy equation

(Case 2) Constrained displacement $\mathbf{p}_p$ NE. 0

$$\mathbf{q}_f = \mathbf{K}_{ff}\mathbf{p}_f + \mathbf{K}_{fp}\mathbf{p}_p \;\Rightarrow\; \mathbf{q}_f - \mathbf{K}_{fp}\mathbf{p}_p = \mathbf{K}_{ff}\mathbf{p}_f$$

$$\Rightarrow \begin{pmatrix}\mathbf{q}_f - \mathbf{K}_{fp}\mathbf{p}_p\\ \mathbf{p}_p\end{pmatrix} = \begin{bmatrix}\mathbf{K}_{ff} & 0\\ 0 & \mathbf{I}\end{bmatrix}\begin{pmatrix}\mathbf{p}_f\\ \mathbf{p}_p\end{pmatrix}$$

[eq. 2.26]

2.2.5 Subroutine BCON

```
      SUBROUTINE BCON(AK,IBC,N,F,IWRIT,IWR)
C     APPLIES BOUNDARY CONDITIONS TO MATRIX AK AS WELL AS
C     ALTERING 'LOAD VECTOR', F FOR PRESCRIBED DISPLACEMENTS.
C     BY SETTING DIAG = 1. AND ROW AND COL TO ZERO IN REST.
C     USES COUNTER IBC WHICH IS 0 IF FREE, 1 IF REST. TO ZERO,
C     -1 IF REST. TO NON-ZERO VALUE
C     ON ENTRY F HAS LOADS FOR FREE ARIABLES AND DISPLACEMENTS FOR
C     REST. (POSSIBLY ZERO) VARIABLES
C     ON EXIT THE LATTER ARE UNCHANGED BUT LOADS ARE ALTERED
C
      DOUBLE PRECISION AK(N,N),F(N)
      INTEGER N,IBC(N),I,J,IPRS,IWRIT,IWR
C
      IPRS = 0
      DO 10 I=1,N
      II = IBC(I)
      IF (II.LT.0) IPRS = 1
      IF (II.NE.0) AK(I,I) = 1.D0
      IF (I.EQ.N) GO TO 10
        DO 20 J=1+1,N
        JJ = IBC(J)
        IF (II.EQ.0.AND.JJ.EQ.0) GO TO 20
C     ABOVE BOTH FREE, BELOW BOTH REST
        IF (II.NE.0.AND.JJ.EQ.0) GO TO 25
C     BELOW I REST OR PRESC
        IF (II.NE.0) THEN
          F(J) = F(J) - AK(I,J)*F(I)
C     BELOW J REST OR PRESC
        ELSE
          F(I) = F(I) - AK(I,J)*F(J)
        ENDIF
   25   AK(I,J) = 0.d0
   20   CONTINUE
   10 CONTINUE
```

$$\begin{pmatrix} \mathbf{q}_f \\ \mathbf{q}_p \end{pmatrix} = \begin{bmatrix} \mathbf{K}_{ff} & \mathbf{K}_{fp} \\ \mathbf{K}_{pf} & \mathbf{K}_{pp} \end{bmatrix} \begin{pmatrix} \mathbf{p}_f \\ \mathbf{p}_p \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} \mathbf{q}_f - \mathbf{K}_{fp}\mathbf{p}_p \\ \mathbf{p}_p \end{pmatrix} = \begin{bmatrix} \mathbf{K}_{ff} & 0 \\ 0 & \mathbf{I} \end{bmatrix} \begin{pmatrix} \mathbf{p}_f \\ \mathbf{p}_p \end{pmatrix}$$

Solution principle - inverse matrix

$$[A]\{x\} = \{b\} \leftrightarrow \{x\} = [A]^{-1}\{b\}$$

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$$

$$\mathbf{A}^{-1} = \frac{1}{\det \mathbf{A}}\left[C_{jk}\right]^T = \frac{1}{\det \mathbf{A}}\begin{bmatrix} C_{11} & C_{21} & \cdots & C_{n1} \\ C_{12} & C_{22} & \cdots & C_{n2} \\ \vdots & \vdots & \cdots & \vdots \\ C_{1n} & C_{2n} & \cdots & C_{nn} \end{bmatrix}$$

C_{jk} : cofactor of a_{jk}

Elimination of unknowns

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 &= b_1 \\ a_{21}x_1 + a_{22}x_2 &= b_2 \end{aligned} \longrightarrow \begin{aligned} a_{21}a_{11}x_1 + a_{21}a_{12}x_2 &= a_{21}b_1 \\ a_{11}a_{21}x_1 + a_{11}a_{22}x_2 &= a_{11}b_2 \end{aligned}$$

$$\longrightarrow \quad (a_{21}a_{12} - a_{11}a_{22})x_2 = a_{21}b_1 - a_{11}b_2$$

$$\to x_2 = \frac{a_{11}b_2 - a_{21}b_1}{a_{11}a_{22} - a_{21}a_{12}}$$

$$\longrightarrow \quad x_1 = \frac{a_{22}b_1 - a_{12}b_2}{a_{11}a_{22} - a_{21}a_{12}}$$

Of course, this is in agreement with results with Cramer's rule

Gauss elimination

The elimination of unknowns can be generalized into the Gauss elimination method.

$$\begin{pmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 4 & -1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \\ -3 \\ 4 \end{pmatrix}$$

$$\longrightarrow \begin{pmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -1 & -5 \\ 0 & -4 & -1 & -7 \\ 0 & 3 & 4 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 4 \\ -7 \\ -15 \\ 8 \end{pmatrix}$$

$$\longrightarrow \begin{pmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -1 & -5 \\ 0 & 0 & 3 & 13 \\ 0 & 0 & 1 & -13 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 4 \\ -7 \\ 13 \\ -13 \end{pmatrix}$$

$$\longrightarrow \begin{pmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -1 & -5 \\ 0 & 0 & 3 & 13 \\ 0 & 0 & 0 & 52 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 4 \\ -7 \\ 13 \\ 52 \end{pmatrix}$$

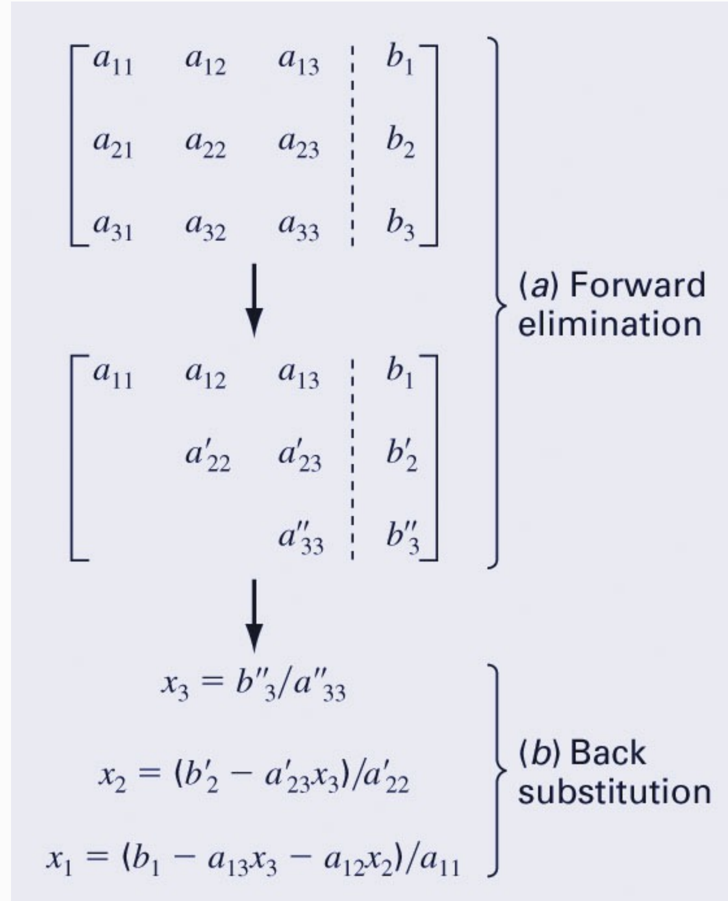
$\boxed{\text{Forward eliminations}}$

$$\begin{aligned} x_4 &= 1 \\ x_3 &= (13 - 13x_4)/3 = 0 \\ x_2 &= -(-7 + x_3 + 5x_4) = 2 \\ x_1 &= 4 - x_2 - 3x_4 = -1 \end{aligned}$$

$\boxed{\text{Backward substitutions}}$

Gauss Elimination (cont)

- Forward elimination
  - Starting with the first row, add or subtract multiples of that row to eliminate the first coefficient from the second row and beyond.
  - Continue this process with the second row to remove the second coefficient from the third row and beyond.
  - Stop when an upper triangular matrix remains.
- Back substitution
  - Starting with the *last* row, solve for the unknown, then substitute that value into the next highest row.
  - Because of the upper-triangular nature of the matrix, each row will contain only one more unknown.



•Pivot equation/ Pivot element/ Normalization

Naïve Gauss Elimination Program – Matlab example

```
function x = GaussNaive(A,b)
% GaussNaive(A,b) :
%     Gauss elimination without pivoting.
% input:
%   A = coefficient matrix
%   b = right hand side vector
% output:
%   x = solution vector

[m,n] = size(A);
if m ~= n, error('Matrix A must be square'); end
nb = n+1;
Aug = [A b];
% forward elimination
for k = 1:n-1  % index for pivot equation
    for i = k+1:n  % i
        factor = Aug(i,k)/Aug(k,k);
        Aug(i,k:nb) = Aug(i,k:nb)-factor*Aug(k,k:nb);
    end
end
% back substitution
x = zeros(n,1);
x(n) = Aug(n,nb)/Aug(n,n);
for i = n-1:-1:1
    x(i) = (Aug(i,nb)-Aug(i,i+1:n)*x(i+1:n))/Aug(i,i);
end
```

should be a column vector

nested loop

(row vector) x (column vector)

Partial Pivoting Program - example

```
function x = gausspivot(A,b)
% GAUSSPIVOT: x = gausspivot(A,b):
%   Gauss elimination with pivoting.
% input:
%   A = coefficient matrix
%   b = right hand side vector
% output:
%   x = solution vector
[m,n]=size(A);
if m~=n,   error('Matrix A must be square'); end
nb=n+1;
Aug=[A b];
% Forward elimination
for k = 1:n-1
    % partial pivoting
    [big,i] = max(abs(Aug(k:n,k)));
    ipr=i+k-1;
    if ipr~=k
       Aug([k,ipr],:)=Aug([ipr,k],:);
    end
   for i = k+1:n
     factor=Aug(i,k)/Aug(k,k);
     Aug(i,k:nb)=Aug(i,k:nb)-factor*Aug(k,k:nb);
   end
end
% Back substitution
x=zeros(n,1);
x(n)=Aug(n,nb)/Aug(n,n);
for i = n-1:-1:1
  x(i)=(Aug(i,nb)-Aug(i,i+1:n)*x(i+1:n))/Aug(i,i);
end
```

LU decomposition

Suppose that we have to change {b} in [A]{x} = {b} frequently for the same [A]. If we apply the Gauss elimination method for every {b}, the forward elimination step is repeated unnecessarily. Therefore, it would be efficient if the forward elimination and back substitution can be separated. This can be achieved through LU (lower\upper) decomposition (or factorization). Let's take the example of a 3x3 matrix. Suppose that we can find L and U matrices such that [L][U] = A and in the form of

$$[L] = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \qquad [U] = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

$$[L]\{[U]\{x\} - \{d\}\} = [A]\{x\} - \{b\}$$

$$[L]\{d\} = \{b\}$$

Since [U] is already upper triangular, [U]{x} = {d} can be obtained by back substitutio

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix}$$

On the other hand, [L]{d} = {b} can be obtained by forward substitution.

In fact, Gauss elimination corresponds to LU factorization.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix}$$

Save memory!

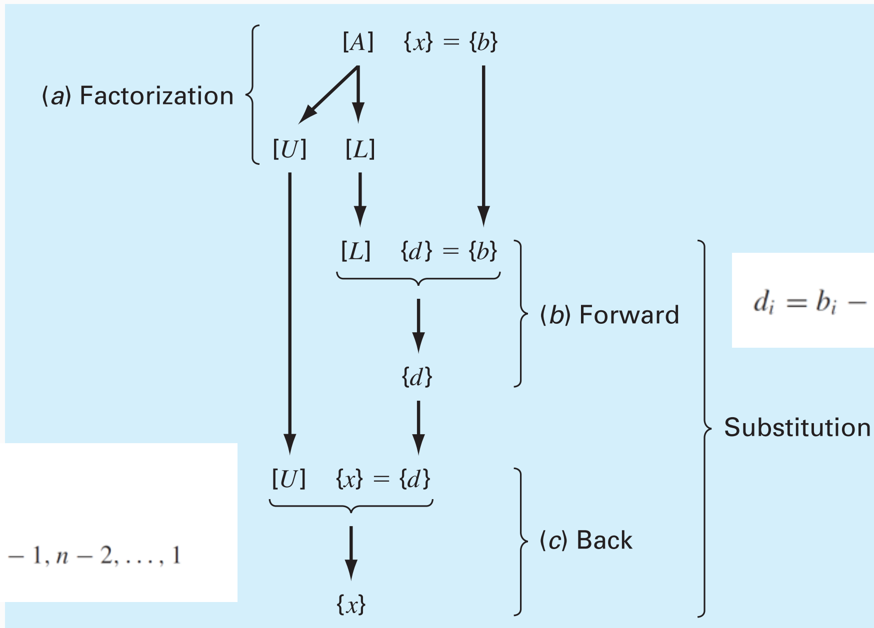
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ f_{21} & a'_{22} & a'_{23} \\ f_{31} & f_{32} & a''_{33} \end{bmatrix}$$

First elimination, $f_{21} = \frac{a_{21}}{a_{11}}$ $\quad f_{31} = \frac{a_{31}}{a_{11}}$

Second elimination, $f_{32} = \frac{a'_{32}}{a'_{22}}$

$$[U] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & 0 & a''_{33} \end{bmatrix} \qquad [L] = \begin{bmatrix} 1 & 0 & 0 \\ f_{21} & 1 & 0 \\ f_{31} & f_{32} & 1 \end{bmatrix}$$

$$[L] = \begin{bmatrix} 1 & 0 & 0 \\ f_{21} & 1 & 0 \\ f_{31} & f_{32} & 1 \end{bmatrix} \qquad [A] = [L][U]$$



$$d_i = b_i - \sum_{j=1}^{i-1} l_{ij} d_j \qquad \text{for } i = 1, 2, \ldots, n$$

$$x_n = d_n / u_{nn}$$

$$x_i = \frac{d_i - \sum_{j=i+1}^{n} u_{ij} x_j}{u_{ii}} \qquad \text{for } i = n-1, n-2, \ldots, 1$$

```
[L,U] = lu(X)
```

What is the use of LU decomposition? For special form of matrices such as sparse, banded, and symmetric ones, there are special algorithms to carry out LU factorizations that are much more efficient than original Gauss elimination. The determinant and inverse matrix can also be obtained by LU decomposition. Brute-force calculation would cost NxN! in comparison with N3 scaling in LU

LU Factorization with Gauss Elimination

Problem Statement. Derive an *LU* factorization based on the Gauss elimination performed previously in Example 9.3.

Solution. In Example 9.3, we used Gauss elimination to solve a set of linear algebraic equations that had the following coefficient matrix:

$$[A] = \begin{bmatrix} 3 & -0.1 & -0.2 \\ 0.1 & 7 & -0.3 \\ 0.3 & -0.2 & 10 \end{bmatrix}$$

After forward elimination, the following upper triangular matrix was obtained:

$$[U] = \begin{bmatrix} 3 & -0.1 & -0.2 \\ 0 & 7.00333 & -0.293333 \\ 0 & 0 & 10.0120 \end{bmatrix}$$

The factors employed to obtain the upper triangular matrix can be assembled into a lower triangular matrix. The elements a_{21} and a_{31} were eliminated by using the factors

$$f_{21} = \frac{0.1}{3} = 0.0333333 \qquad f_{31} = \frac{0.3}{3} = 0.1000000$$

and the element a_{32} was eliminated by using the factor

$$f_{32} = \frac{-0.19}{7.00333} = -0.0271300$$

Thus, the lower triangular matrix is

$$[L] = \begin{bmatrix} 1 & 0 & 0 \\ 0.0333333 & 1 & 0 \\ 0.100000 & -0.0271300 & 1 \end{bmatrix}$$

Consequently, the *LU* factorization is

$$[A] = [L][U] = \begin{bmatrix} 1 & 0 & 0 \\ 0.0333333 & 1 & 0 \\ 0.100000 & -0.0271300 & 1 \end{bmatrix} \begin{bmatrix} 3 & -0.1 & -0.2 \\ 0 & 7.00333 & -0.293333 \\ 0 & 0 & 10.0120 \end{bmatrix}$$

This result can be verified by performing the multiplication of $[L][U]$ to give

$$[L][U] = \begin{bmatrix} 3 & -0.1 & -0.2 \\ 0.0999999 & 7 & -0.3 \\ 0.3 & -0.2 & 9.99996 \end{bmatrix}$$

where the minor discrepancies are due to roundoff.

The Substitution Steps

Problem Statement. Complete the problem initiated in Example 10.1 by generating the final solution with forward and back substitution.

Solution. As just stated, the intent of forward substitution is to impose the elimination manipulations that we had formerly applied to $[A]$ on the right-hand-side vector $\{b\}$. Recall that the system being solved is

$$\begin{bmatrix} 3 & -0.1 & -0.2 \\ 0.1 & 7 & -0.3 \\ 0.3 & -0.2 & 10 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 7.85 \\ -19.3 \\ 71.4 \end{Bmatrix}$$

and that the forward-elimination phase of conventional Gauss elimination resulted in

$$\begin{bmatrix} 3 & -0.1 & -0.2 \\ 0 & 7.00333 & -0.293333 \\ 0 & 0 & 10.0120 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 7.85 \\ -19.5617 \\ 70.0843 \end{Bmatrix}$$

The forward-substitution phase is implemented by applying Eq. (10.8):

$$\begin{bmatrix} 1 & 0 & 0 \\ 0.0333333 & 1 & 0 \\ 0.100000 & -0.0271300 & 1 \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} 7.85 \\ -19.3 \\ 71.4 \end{Bmatrix}$$

or multiplying out the left-hand side:

$$\begin{aligned} d_1 \qquad\qquad\qquad\qquad &= 7.85 \\ 0.0333333d_1 + \qquad d_2 \qquad &= -19.3 \\ 0.100000d_1 - 0.0271300d_2 + d_3 &= 71.4 \end{aligned}$$

We can solve the first equation for $d_1 = 7.85$, which can be substituted into the second equation to solve for

$$d_2 = -19.3 - 0.0333333(7.85) = -19.5617$$

Both d_1 and d_2 can be substituted into the third equation to give

$$d_3 = 71.4 - 0.1(7.85) + 0.02713(-19.5617) = 70.0843$$

Thus,

$$\{d\} = \begin{Bmatrix} 7.85 \\ -19.5617 \\ 70.0843 \end{Bmatrix}$$

This result can then be substituted into Eq. (10.3), $[U]\{x\} = \{d\}$:

$$\begin{bmatrix} 3 & -0.1 & -0.2 \\ 0 & 7.00333 & -0.293333 \\ 0 & 0 & 10.0120 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 7.85 \\ -19.5617 \\ 70.0843 \end{Bmatrix}$$

which can be solved by back substitution (see Example 9.3 for details) for the final solution:

$$\{x\} = \begin{Bmatrix} 3 \\ -2.5 \\ 7.00003 \end{Bmatrix}$$

2.2.6 Subroutine CROUT

William H. Press, Saul a. Teukolsky, William T. Vetterling, Brian P. Flannery, "Numerical Recipes in Fortran 77: the Art of Scientific Computing. Second Edition", vol. 1, 1996.

- This subroutine applies the **Crout factorization** to the **tangent stiffness matrix**, to conduct **LDL^T decomposition**.

Suppose we are able to write the matrix **A** as a product of two matrices,

$$\mathbf{K} = \mathbf{LU} = \mathbf{LDL}^T \qquad \mathbf{L} \cdot \mathbf{U} = \mathbf{A} \tag{2.3.1}$$

where **L** is *lower triangular* (has elements only on the diagonal and below) and **U** is *upper triangular* (has elements only on the diagonal and above). For the case of a 4×4 matrix **A**, for example, equation (2.3.1) would look like this:

$$\begin{bmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix} \cdot \begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \tag{2.3.2}$$

Performing the LU Decomposition

How then can we solve for **L** and **U**, given **A**? First, we write out the i, jth component of equation (2.3.1) or (2.3.2). That component always is a sum beginning with

$$\alpha_{i1}\beta_{1j} + \cdots = a_{ij}$$

The number of terms in the sum depends, however, on whether i or j is the smaller number. We have, in fact, the three cases,

$$i < j: \qquad \alpha_{i1}\beta_{1j} + \alpha_{i2}\beta_{2j} + \cdots + \alpha_{ii}\beta_{ij} = a_{ij} \tag{2.3.8}$$

$$i = j: \qquad \alpha_{i1}\beta_{1j} + \alpha_{i2}\beta_{2j} + \cdots + \alpha_{ii}\beta_{jj} = a_{ij} \tag{2.3.9}$$

$$i > j: \qquad \alpha_{i1}\beta_{1j} + \alpha_{i2}\beta_{2j} + \cdots + \alpha_{ij}\beta_{jj} = a_{ij} \tag{2.3.10}$$

Equations (2.3.8)–(2.3.10) total N^2 equations for the $N^2 + N$ unknown α's and β's (the diagonal being represented twice). Since the number of unknowns is greater than the number of equations, we are invited to specify N of the unknowns arbitrarily and then try to solve for the others. In fact, as we shall see, it is always possible to take

$$\alpha_{ii} \equiv 1 \qquad i = 1, \ldots, N \tag{2.3.11}$$

A surprising procedure, now, is *Crout's algorithm*, which quite trivially solves the set of $N^2 + N$ equations (2.3.8)–(2.3.11) for all the α's and β's by just arranging the equations in a certain order! That order is as follows:

- Set $\alpha_{ii} = 1$, $i = 1, \ldots, N$ (equation 2.3.11).
- For each $j = 1, 2, 3, \ldots, N$ do these two procedures: First, for $i = 1, 2, \ldots, j$, use (2.3.8), (2.3.9), and (2.3.11) to solve for β_{ij}, namely

$$\beta_{ij} = a_{ij} - \sum_{k=1}^{i-1} \alpha_{ik}\beta_{kj}. \tag{2.3.12}$$

(When $i = 1$ in 2.3.12 the summation term is taken to mean zero.) Second, for $i = j+1, j+2, \ldots, N$ use (2.3.10) to solve for α_{ij}, namely

$$\alpha_{ij} = \frac{1}{\beta_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} \alpha_{ik}\beta_{kj} \right). \tag{2.3.13}$$

Be sure to do both procedures before going on to the next j.

If you work through a few iterations of the above procedure, you will see that the α's and β's that occur on the right-hand side of equations (2.3.12) and (2.3.13) are already determined by the time they are needed. You will also see that every a_{ij} is used only once and never again. This means that the corresponding α_{ij} or β_{ij} can be stored in the location that the a used to occupy: the decomposition is "in place." [The diagonal unity elements α_{ii} (equation 2.3.11) are not stored at all.] In brief, Crout's method fills in the combined matrix of α's and β's,

$$\begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ \alpha_{21} & \beta_{22} & \beta_{23} & \beta_{24} \\ \alpha_{31} & \alpha_{32} & \beta_{33} & \beta_{34} \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \beta_{44} \end{bmatrix} \tag{2.3.14}$$

by columns from left to right, and within each column from top to bottom (see Figure 2.3.1).

2.2.6 Subroutine CROUT

$$\mathbf{K} = \mathbf{LU} = \mathbf{LDL}^T$$

```
      SUBROUTINE CROUT(AK,D,N,IWRIT,IWR)
C
C     INPUTS AK(N,N); OUTPUTS UPPER TRIANGLE IN AK AND DIAG
C     PIVOTS IN D(N)
C
      DOUBLE PRECISION AK(N,N),D(N),A
      INTEGER N,I,J,IWR,IWRIT
C
      D(1) = AK(1,1)
      DO 1 J=2,N
        DO 2 I=1,J-1
          A = AK(I,J)
          IF (I.EQ.1) GO TO 2
          DO 3 L=1,I-1
            A=A-AK(L,J)*AK(L,I)
    3     CONTINUE
          AK(I,J) = A
    2   CONTINUE
        DO 4 I=1,J-1
          AK(I,J) = AK(I,J)/AK(I,I)
    4   CONTINUE
        DO 5 L=1,J-1
          AK(J,J) = AK(J,J) - AK(L,J)*AK(L,J)*AK(L,J)
    5   CONTINUE
        D(J) = AK(J,J)
    1   CONTINUE
C
```

2.2.6 Subroutine SOLVCR

- This subroutine solves the problem by Crout forward-backward method.

$$\begin{pmatrix} \mathbf{q}_f - \mathbf{K}_{fp}\mathbf{p}_p \\ \mathbf{p}_p \end{pmatrix} = \begin{bmatrix} \mathbf{K}_{ff} & 0 \\ 0 & \mathbf{I} \end{bmatrix} \begin{pmatrix} \mathbf{p}_f \\ \mathbf{p}_p \end{pmatrix} = \left(\mathbf{LDL}^T\right)\mathbf{p}$$

```fortran
      SUBROUTINE SOLVCR(AK,D,Q,N,IWRIT,IWR)
C
C     APPLIES FORWARD AND BACK CROUT SUBS ON Q
C
      DOUBLE PRECISION AK(N,N),D(N),Q(N)
      INTEGER N,I,J,L,IWRIT,IWR
C
C     FORWARD SUBS
      DO 1 J=2,N
        DO 2 L=1,J-1
          Q(J) = Q(J) - AK(L,J)*Q(L)
    2   CONTINUE
    1 CONTINUE
      IF (IWRIT.NE.0) THEN
        WRITE (IWR,1000) (Q(I), I=1,N)
 1000   FORMAT(/,1X,'DISP.INCS AFTER FORWARD SUBS.ARE',1X,7G12.5,/)
      ENDIF
C
C     BACK SUBS.
      DO 3 I=1,N
        Q(I) = Q(I)/D(I)
    3 CONTINUE
C
      DO 4 JJ=2,N
        J = N + 2 - JJ
        DO 5 L=1,J-1
          Q(L) = Q(L) - AK(L,J)*Q(J)
    5   CONTINUE
    4 CONTINUE
```

Thank you!