Chapter 2: A shallow truss element with Fortran computer program

Myoung-Gyu Lee

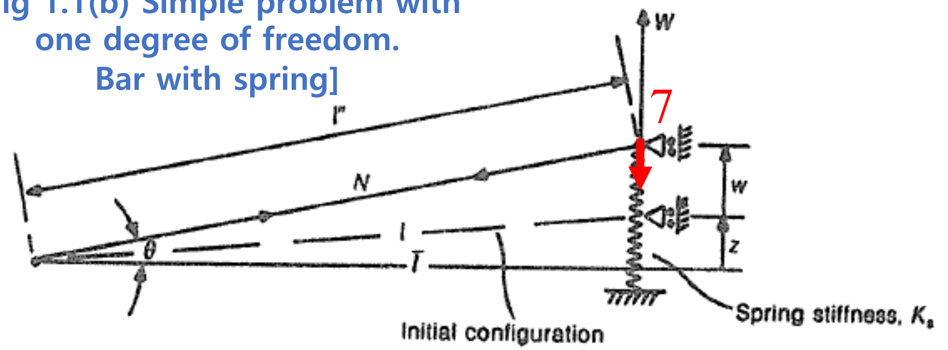
TA: Gyu Jang Sim (gyujang95@snu.ac.kr)

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

MAMEL
MATERIALS MECHANICS LABORATORY

- **Incremental (Euler) solution is implemented by a load-level factor λ .**

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




```
4    50000000.   2500. 0.
0.   25.
0.   0.    0.    -7.
1    1     1     0
1
4
1.35
1.   12    1
```

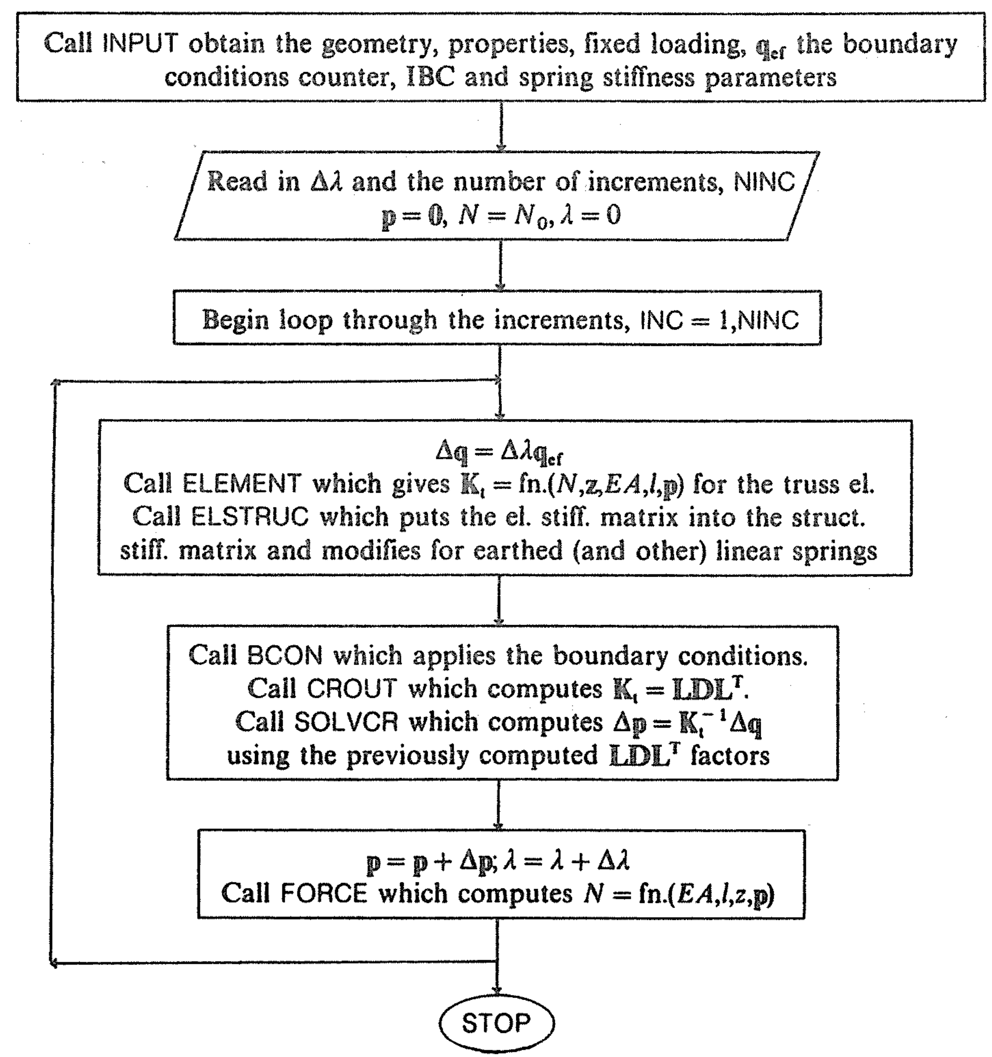
$\mathbf{q}_{ef}$

$\lambda = 1.0, 2.0, 3.0, \cdots 12.0$

, *write mode on*

[input file of Fig 1.1(b)]

$$\mathbf{q}_e = \lambda \mathbf{q}_{ef}$$



[Fig. 2.3 Flowchart for an incremental solution (program NONLTA)]

2.3.1 Program NONLTA

```
      PROGRAM NONLTA
C
C     PERFORMS NON-LINEAR INCREMENTAL SOLUTION FOR TRUSS
C     N = NUMBER OF ARIABLES(4 OR 5)
C     QFI = FIXED LOAD VECTOR
C     IBC = BOUNDARY CONDITION COUNTER(0=FREE, 1=FIXED)
C     Z = C COORDS OF NODES
C     QINC = INCREMENTAL LOAD VECTOR
C     PT = TOTAL DISPLACEMENT VECTOR
C     AKTS = STRUCT. TANGENT STIFFNESS MATRIX
C     AKTE = ELEMENT TANGENT STIFFNESS MATRIX
C     FI(NOT USED HERE) = INTERNAL FORCES
C     D = DIAGONAL PIVOTS FROM LDL(TRAN) FACTORISATION
C     ID14S = VAR. NOS. (1-4) AT WHICH LIN EARTHED SPRINGS
C     AK14S = EQUIVALENT LINEAR SPRING STIFFNESSES
C     AK15 = LINEAR SPRING STIFFNESS BETWEEN VARBLS. 1 AND 5 (IF NV=5)
C
      DOUBLE PRECISION QFI(5),Z(2),QINC(5),PT(5),AKTE(4,4),FI(5),D(5),
     1                 AK14S(4),AKTS(25),X(2),POISS,E,ARA,AL,ANIT,AK15,
     2                 AN,ALN,ARN
      INTEGER IBC(5),ID14S(4),IRE,IWR,I,NV,NDSP,ITYEL,N

C     ARRAY X ABOVE NOT USED FOR SHALLOW TRUSS
C
      IRE = 5
      IWR = 6
      OPEN(UNIT=5, FILE=' ')
      OPEN(UNIT=6, FILE=' ')
```

2.3.1 Program NONLTA

```
      CALL INPUT(E,ARA,AL,QFI,X,Z,ANIT,IBC,IRE,IWR,AK14S,ID14S,NDSP,
     1          NV,AK15,
     2          POISS,ITYEL)
C     ARGUMENTS IN LINE ABOVE NOT USED FOR SHALLOW TRUSS
C     BELOW RELEVANT TO DEEP TRUSS BUT LEAVE FOR SHALLOW TRUSS
      ALN = AL
      ARN = ARA
C
      READ(IRE,*) FACI,NINC,IWRIT
      WRITE(IWR,1000) FACI,NINC,IWRIT
 1000 FORMAT(/,1X,'INCREMENTAL LOAD FACTOR =',G13.5,/,1X,
     1      'NO. OF INC. (NINC)=',G13.5,/,1X,
     2      'WRITE CONTROL (IWRIT)',G13.5,/,3X
     3      '0=LIMITED ; 1=FULL',/)
C
      AN = ANIT
      FACT = 0.D0
      DO 5 I=1,NV
        PT(I) = 0.D0
    5 CONTINUE
C
C
      DO 100 INC=1,NINC
        FACT = FACT + FACI
        WRITE(IWR,1001) INC,FACT
 1001   FORMAT(/,1X,'INC = ',G13.5,'LD.FACTOR = ',G13.5,/)
        DO 10 I=1,NV
          QINC(I) = FACI*QFI(I)
   10   CONTINUE
```

2.3.1 Program NONLTA

```
C     BELOW FORMS ELEMENT TANGENT STIFFNESS MATRIX AKT
        CALL ELEMENT(FI,AKTE,AN,X,Z,PT,E,ARA,AL,IWRIT,IWR,2,
     1               ITYEL,ALN,ARN)
C     ARGUMENTS IN LINE ABOVE NOT USED FOR SHALLOW TRUSS
C
        CALL ELSTRUC(AKTE,AKTS,NV,AK15,ID14S,AK14S,NDSP,FI,PT,
     1              2,IWRIT,IWR)
C     ABOVE PUTS EL.STIFF AKTE IN STRUC STIFF AKTS AND
C     ADDS EFFECTS OF VARIOUS LINEAR SPRINGS
C
        CALL BCON(AKTS,IBC,N,QINC,IWRIT,IWR)
C     ABOVE APPLIES B.CONDITIONS
        CALL CROUT(AKTS,D,NV,IWRIT,IWR)
C     ABOVE FORMS LDL(TRAN) FACTORISATION INTO AKT AND D
        CALL SOLVCR(AKTS,D,QINC,NV,IWRIT,IWR)
C     ABOVE SOLVES EQNS. AND GETS INC. DISPS IN QIN
C
        DO 20 I=1,NV
          PT(I) = PT(I) + QINC(I)
   20   CONTINUE
C     ABOVE UPDATES TOTAL DISPS.
C
      WRITE (6,1002) (PT(I), I=1,NV)
 1002 FORMAT(/,1X,'TOTAL DISPS. ARE',1X,5G13.5,/)
C
C     BELOW FORMS TOTAL FORCE IN BAR
      CALL FORCE(AN,ANIT,E,ARA,AL,X,Z,PT,IWRIT,IWR,
     1           ITYEL,ARN,ALN,POISS)
C     ABOVE ARGUMENTS NOT USED FOR SHALLOW TRUSS
  100 CONTINUE
C
      STOP 'NONLTA'
```

Iterative method for solving linear systems

Unlike direct method such as Gauss elimination, a solution is assumed in the iterative method and it is iteratively updated until the convergence is achieved.

Gauss-Seidel Method

- The *Gauss-Seidel method* is the most commonly used iterative method for solving linear algebraic equations [*A*]{*x*}={*b*}.
- The method solves each equation in a system for a particular variable, and then uses that value in later equations to solve later variables. For a 3x3 system with nonzero elements along the diagonal, for example, the *j*th iteration values are found from the *j*-1th iteration using:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned} \quad \leftrightarrow \quad \begin{aligned} x_1 &= \frac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}} \\ x_2 &= \frac{b_2 - a_{21}x_1 - a_{23}x_3}{a_{22}} \\ x_3 &= \frac{b_3 - a_{31}x_1 - a_{32}x_2}{a_{33}} \end{aligned} \quad \longrightarrow \quad \begin{aligned} x_1^j &= \frac{b_1 - a_{12}x_2^{j-1} - a_{13}x_3^{j-1}}{a_{11}} \\ x_2^j &= \frac{b_2 - a_{21}x_1^j - a_{23}x_3^{j-1}}{a_{22}} \\ x_3^j &= \frac{b_3 - a_{31}x_1^j - a_{32}x_2^j}{a_{33}} \end{aligned}$$

If $\{x\}^j$ and $\{x\}^{j-1}$ are equal, the equations become self-consistent and $\{x\}^j$ is the solution set.

Jacobi Iteration

- The *Jacobi iteration* is similar to the Gauss-Seidel method, except the j-1th information is used to update all variables in the jth iteration:

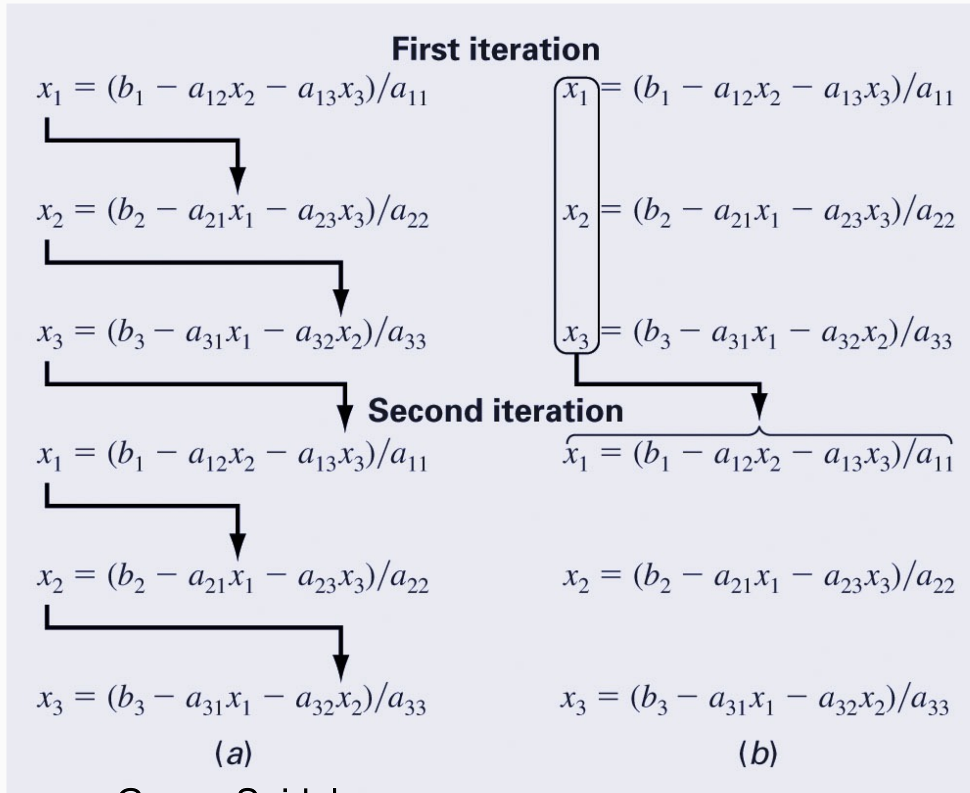
  a) Gauss-Seidel

  b) Jacobi

$$x_1^j = \frac{b_1 - a_{12}x_2^{j-1} - a_{13}x_3^{j-1}}{a_{11}}$$

$$x_2^j = \frac{b_2 - a_{21}x_1^{j-1} - a_{23}x_3^{j-1}}{a_{22}}$$

$$x_3^j = \frac{b_3 - a_{31}x_1^{j-1} - a_{32}x_2^{j-1}}{a_{33}}$$



Gauss-Seidel Jacobi

Convergence

- The convergence of an iterative method can be calculated by determining the relative percent change of each element in {*x*}. For example, for the *i*th element in the *j*th iteration,

$$\varepsilon_{a,i} = \left| \frac{x_i^j - x_i^{j-1}}{x_i^j} \right| \times 100\%$$

- The method is ended when all elements have converged to a set tolerance.

Ex)

$$3x_1 - 0.1x_2 - 0.2x_3 = 7.85$$
$$0.1x_1 + 7x_2 - 0.3x_3 = -19.3$$
$$0.3x_1 - 0.2x_2 + 10x_3 = 71.4$$

$\rightarrow$

$$x_1 = \frac{7.85 + 0.1x_2 + 0.2x_3}{3}$$
$$x_2 = \frac{-19.3 - 0.1x_1 + 0.3x_3}{7}$$
$$x_3 = \frac{71.4 - 0.3x_1 + 0.2x_2}{10}$$

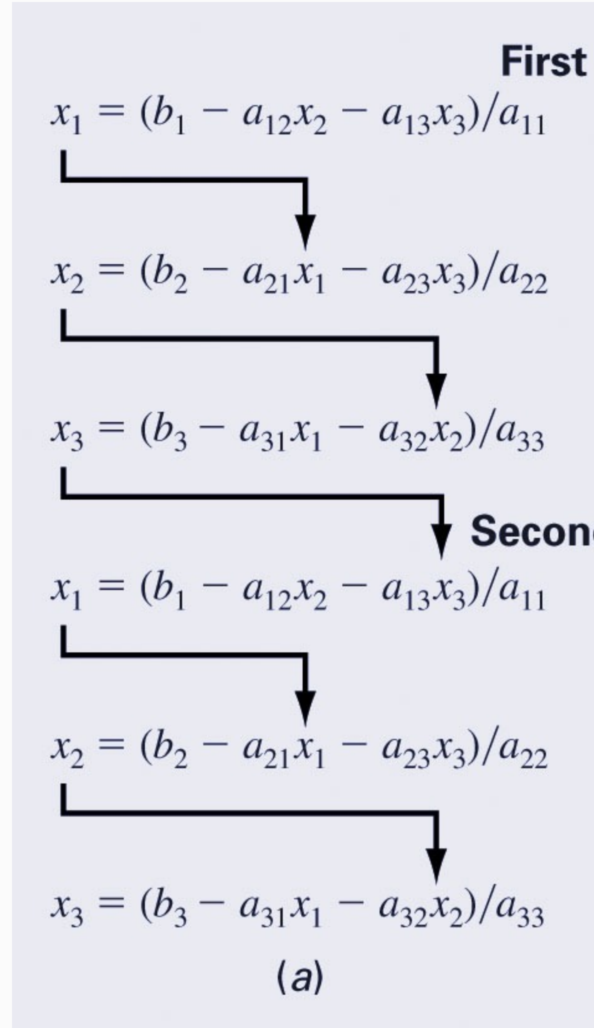
i) Gauss-Seidel (start with $x_2 = x_3 = 0$)

Iter.	x_1	x_2	x_3	x_1 err(%)	x_2 err(%)	x_3 err(%)
1	2.61667	-2.79452	7.00561	-Infinity	Infinity	-Infinity
2	2.99056	-2.49962	7.00029	-14.28878	10.55275	0.07592
3	3.00003	-2.49999	7.00000	-0.31684	-0.01453	0.00416
4	3.00000	-2.50000	7.00000	0.00105	-0.00048	-0.00001
5	3.00000	-2.50000	7.00000	0.00001	0.00000	0.00000

ii) Jacobi (x_1=x_2=x_3=0)

Iter.	x_1	x_2	x_3	x_1 err(%)	x_2 err(%)	x_3 err(%)
1	2.61667	-2.75714	7.14000	-Infinity	Infinity	-Infinity
2	3.00076	-2.48852	7.00636	-14.67880	9.74266	1.87175
3	3.00081	-2.49974	7.00021	-0.00148	-0.45065	0.08778
4	3.00002	-2.50000	6.99998	0.02612	-0.01057	0.00322
5	3.00000	-2.50000	7.00000	0.00079	0.00006	-0.00026

```
function x = GaussSeidel(A,b,es,maxit)
% x = GaussSeidel(A,b):
%   Gauss Seidel method.
% input:
%   A = coefficient matrix
%   b = right hand side vector
%   es = (optional) stop criterion (%) (default = 0.00001)
%   maxit = (optional) max iterations (default = 50)
% output:
%   x = solution vector
if nargin<4, maxit=50; end
if nargin<3, es=0.00001; end
[m,n] = size(A);
if m~=n, error('Matrix A must be square'); end
C = A;
for i = 1:n
    C(i,i) = 0;
    x(i) = 0;
end
x = x';
for i = 1:n
    C(i,1:n) = C(i,1:n)/A(i,i);
end
for i = 1:n
    d(i) = b(i)/A(i,i);
end
iter = 0;
while (1)
  xold = x;
  for i = 1:n
    x(i) = d(i)-C(i,:)*x;
    if x(i) ~= 0
      ea(i) = abs((x(i) - xold(i))/x(i)) * 100;
    end
  end
  iter = iter+1;
  if max(ea)<=es | iter >= maxit,
     break,
  end
end
```



Convergence condition

The Gauss-Seidel method converge if the system is i) *strictly diagonally dominant* or ii) *symmetric positive-definite*.

Strict diagonal dominance:

$$\left|a_{ii}\right| > \sum_{\substack{j=1 \\ j \neq i}}^{n} \left|a_{ij}\right|$$

Cf. (Weak) diagonal dominance:

$$\left|a_{ii}\right| \geq \sum_{\substack{j=1 \\ j \neq i}}^{n} \left|a_{ij}\right|$$

Ex. Determine diagonal dominance:

$$\mathbf{A} = \begin{bmatrix} 3 & -2 & 1 \\ 1 & -3 & 2 \\ -1 & 2 & 4 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -2 & 2 & 1 \\ 1 & 3 & 2 \\ 1 & -2 & 0 \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} -4 & 2 & 1 \\ 1 & 6 & 2 \\ 1 & -2 & 5 \end{bmatrix}$$

Symmetric positive-definite

A symmetric real matrix M is said to be positive definite if $z^T M z$ is positive for all nonzero real column vector z

Ex.

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad z^T I z = \begin{pmatrix} a & b \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = a^2 + b^2$$

$$M = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix} \quad z^T M z = \begin{pmatrix} a & b & c \end{pmatrix} \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = a^2 + (a-b)^2 + (b-c)^2 + c^2$$

$$N = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = -2$$

Equivalent condition for positive-definitedness is that all eigenvalues of M are positive.

Examples for Gauss-Seidel

$$A=\begin{pmatrix} -4 & 2 & 1 \\ 1 & 6 & 2 \\ 1 & -2 & 5 \end{pmatrix} \quad b=\begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix}$$

: diagonally dominant. (not symmetric) Changing b into (300 2 1)' still results in good convergence. The results are checked with A\b.

$$A=\begin{pmatrix} 2 & -4 & 1 \\ 1 & 6 & 2 \\ 1 & -2 & 5 \end{pmatrix} \quad b=\begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix}$$

:diagonally not dominant but converges.

$$A=\begin{pmatrix} 2 & -4 & 1 \\ 6 & 1 & 2 \\ 1 & -2 & 5 \end{pmatrix} \quad b=\begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix}$$

: diverge

$$A=\begin{pmatrix} 5 & 2 & 4 \\ 2 & 4 & 3 \\ 4 & 3 & 5 \end{pmatrix} \quad b=\begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix}$$

Not diagonally dominant but positive-definite and converges

Relaxation

- To enhance convergence, an iterative program can introduce *relaxation* where the value at a particular iteration is made up of a combination of the old value and the newly calculated value:

$$x_i^{\text{new}} = \lambda x_i^{\text{new}} + (1 - \lambda) x_i^{\text{old}}$$

where λ is a weighting factor that is assigned a value between 0 and 2.

  - $0 < \lambda < 1$: underrelaxation
  - $\lambda = 1$: no relaxation
  - $1 < \lambda \leq 2$: overrelaxation

Example with Jacobi method

Newton-Raphson

- Nonlinear systems may also be solved using the Newton-Raphson method for multiple variables.
- For a two-variable system, the Taylor series approximation and resulting Newton-Raphson equations are:

$$f_1(x_1, x_2) = 0$$
$$f_2(x_1, x_2) = 0$$

Suppose that at the ith step, $x_1 = x_{1,i}$ and $x_2 = x_{2,i}$ with $f_{1,i}(x_{1,i}, x_{2,i})$ and $f_{2,i}(x_{1,i}, x_{2,i})$

$$f_1(x_1, x_2) \simeq f_1(x_{1,i}, x_{2,i}) + (x_1 - x_{1,i}) \left.\frac{\partial f_1}{\partial x_1}\right|_{(x_{1,i}, x_{2,i})} + (x_2 - x_{2,i}) \left.\frac{\partial f_1}{\partial x_2}\right|_{(x_{1,i}, x_{2,i})}$$
$$f_2(x_1, x_2) \simeq f_2(x_{1,i}, x_{2,i}) + (x_1 - x_{1,i}) \left.\frac{\partial f_2}{\partial x_1}\right|_{(x_{1,i}, x_{2,i})} + (x_2 - x_{2,i}) \left.\frac{\partial f_2}{\partial x_2}\right|_{(x_{1,i}, x_{2,i})}$$

Choose $x_{1,i+1}$ and $x_{2,i+1}$ such that $f_1(x_{1,i+1}, x_{2,i+1}) = f_2(x_{1,i+1}, x_{2,i+1}) = 0$

$$\longrightarrow \quad f_{1,i} + \left(x_{1,i+1} - x_{1,i}\right)\frac{\partial f_{1,i}}{\partial x_1} + \left(x_{2,i+1} - x_{2,i}\right)\frac{\partial f_{1,i}}{\partial x_2} = 0$$

$$f_{2,i} + \left(x_{1,i+1} - x_{1,i}\right)\frac{\partial f_{2,i}}{\partial x_1} + \left(x_{2,i+1} - x_{2,i}\right)\frac{\partial f_{2,i}}{\partial x_2} = 0$$

Using Cramer's rule

$$x_{1,i+1} = x_{1,i} - \frac{f_{1,i}\dfrac{\partial f_{2,i}}{\partial x_2} - f_{2,i}\dfrac{\partial f_{1,i}}{\partial x_2}}{\dfrac{\partial f_{1,i}}{\partial x_1}\dfrac{\partial f_{2,i}}{\partial x_2} - \dfrac{\partial f_{1,i}}{\partial x_2}\dfrac{\partial f_{2,i}}{\partial x_1}}$$

$$x_{2,i+1} = x_{2,i} - \frac{f_{2,i}\dfrac{\partial f_{1,i}}{\partial x_1} - f_{1,i}\dfrac{\partial f_{2,i}}{\partial x_1}}{\dfrac{\partial f_{1,i}}{\partial x_1}\dfrac{\partial f_{2,i}}{\partial x_2} - \dfrac{\partial f_{1,i}}{\partial x_2}\dfrac{\partial f_{2,i}}{\partial x_1}}$$

In general, for a set of n equations with n variables,

$$\{x_{i+1}\} = \{x_i\} - J^{-1}\{f\}$$

$$J = \begin{pmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_n}{\partial x_1} & \cdots & \dfrac{\partial f_n}{\partial x_n} \end{pmatrix}_{\{x_i\}}$$

cf. N-R method for single variable: $x_{i+1} = x_i - \dfrac{f(x_i)}{f'(x_i)}$

MATLAB Program

```
function [x,f,ea,iter]=newtmult(func,x0,es,maxit,varargin)
% newtmult: Newton-Raphson root zeroes nonlinear systems
%  [x,f,ea,iter]=newtmult(func,x0,es,maxit,p1,p2,...):
%    uses the Newton-Raphson method to find the roots of
%    a system of nonlinear equations
% input:
%   func = name of function that returns f and J
%   x0 = initial guess
%   es = desired percent relative error (default = 0.0001%)
%   maxit = maximum allowable iterations (default = 50)
%   p1,p2,... = additional parameters used by function
% output:
%   x = vector of roots
%   f = vector of functions evaluated at roots
%   ea = approximate percent relative error (%)
%   iter = number of iterations

if nargin<2,error('at least 2 input arguments required'),end
if nargin<3|isempty(es),es=0.0001;end
if nargin<4|isempty(maxit),maxit=50;end
iter = 0;
x=x0;
while (1)
  [J,f]=func(x,varargin{:});
  dx=J\f;
  x=x-dx;
  iter = iter + 1;
  ea=100*max(abs(dx./x));
  if iter>=maxit|ea<=es, break, end
end
```

Ex)

```
function [J,f] = jf(x)
J = [2*x(1)+x(2) x(1); 3*x(2)^2 1+6*x(1)*x(2)];
f = [x(1)^2+x(1)*x(2)-10;x(2)+3*x(1)*x(2)^2-57];
```

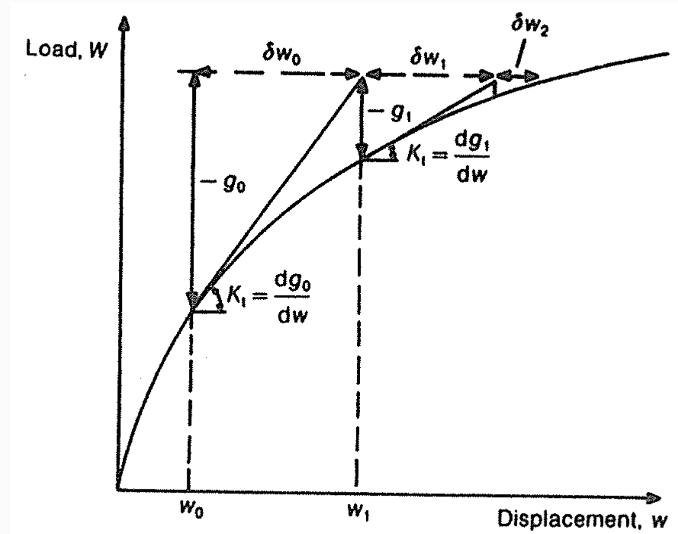
- This program uses an iterative solution by extending the concepts of ch 1.2.2 and 1.3 .

$$g = \frac{EA}{l^3}\left(z^2 w + \frac{3}{2} z w^2 + \frac{1}{2} w^3\right) - W = \frac{N(z+w)}{l} - W = 0$$

$$\delta w = -\left(\frac{dg}{dw}\right)^{-1} g$$

[eq.1.23,1.28]

$$w_{n+1} = w_n + \delta w_n$$



[Fig 1.5 The Newton-Raphson method]

- Newton-Raphson method requires convergence criteria, since practically, out-of-balance force will never be zero.

$$\|\mathbf{g}\| < \beta \|\mathbf{q}_e\| \quad \cdots\cdots \quad \textit{force control}$$

$$\|\mathbf{g}\| < \beta \|\mathbf{r}\| \quad \cdots\cdots \quad \textit{displacement control}$$

[eq. 2.30]

$$\mathbf{r} : \textit{reaction vector} \qquad \beta = 0.001 \sim 0.01$$

- Newton-Raphson method also requires initial guess of $\mathbf{p}$.

```
0.0 0.0 0.0 0.0
0.001 1
```

$\mathbf{p}^T$

β, *write mode on*

[Last part of input file for NONLTB]

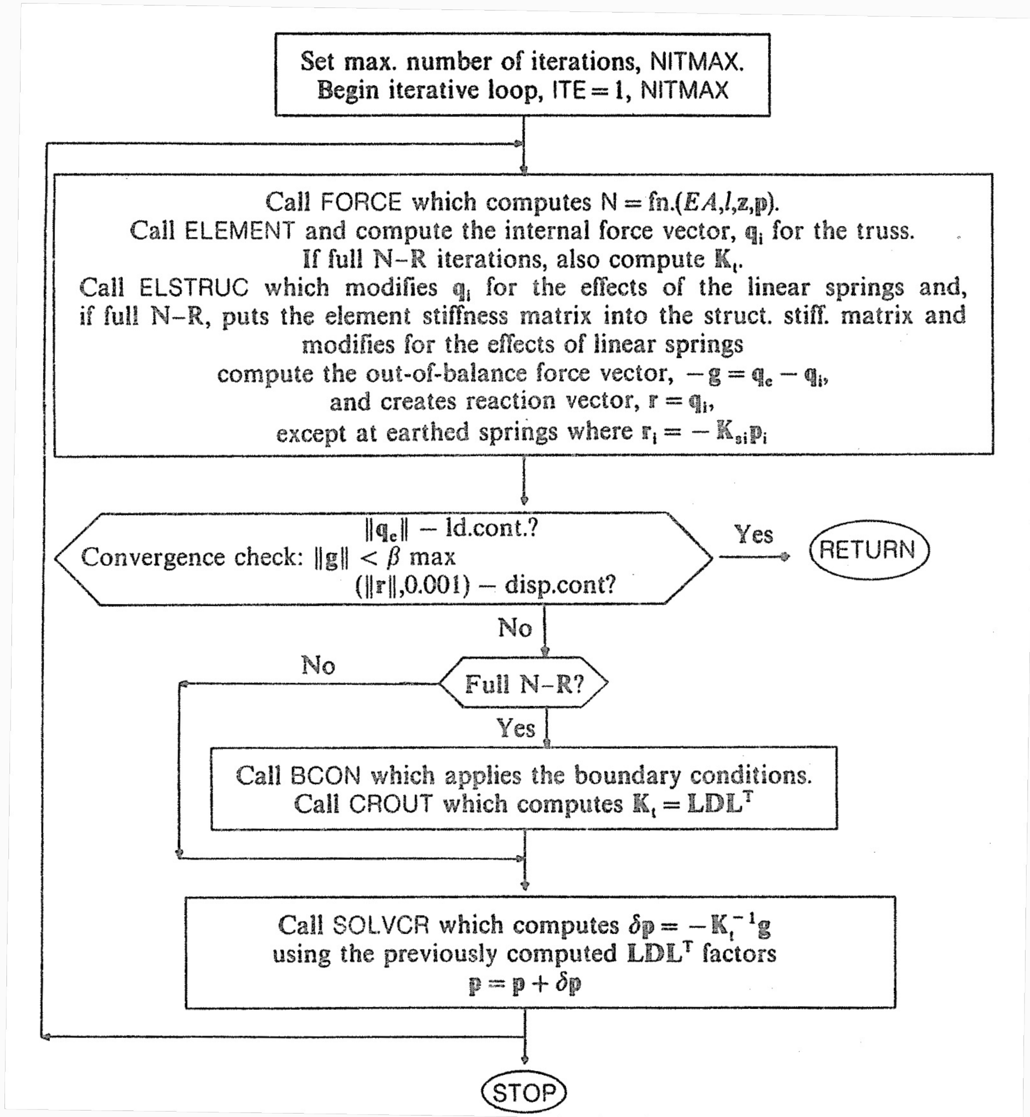
2.4.1 Program NONLTB

```
      PROGRAM NONLTB
C
C     PREFORMS NEWTON-RAPHSON ITERATION FROM STARTING PREDICTOR, PT
C     NV=NO. OF VARIABLES (4 OR 5)
C     IBC = B. COND. COUNTER (0=FREE, 1=FIXED)
C     Z=Z COORDS OF NODES
C     PT=TOTAL DISP. VECTOR
C     ID14S=VAR. NOS. (1-4) AT WHICH LINEAR EARTHED SPRINGS
C     AK12S=EQUIV. LINEAR SPRING STIFFNESSES
C     QFI=TOTAL LOAD VECTOR
C     AKTS=STRUC. STIFF. MATRIX
C     AK15=LIN SPRING STIFF. BETWEEN VARBLS. 1 AND 5 (IF NV=5)
C     FI=INTERNAL FORCE VECTOR
C     GM=OUT-OF-BALANCE FORCE VECTOR
C     REAC=REACTIONS
C     X=X COORDS
C     ARGUMENTS IN COMMON/DAT2/ AND ARRAY X NOT USED FOR SHALLOW TRUSS
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON /DAT/ X(2),Z(2),E,ARA,AL,ID14S(4),AK14S(4),NDSP,ANIT,AK15
      COMMON /DAT2/ ARN,POISS,ALN,ITYEL
      DIMENSION QFI(5),IBC(5),PT(5),AKTS(25),D(5),GM(5),FI(5)
      DIMENSION REAC(5)
C
      IRE=5
      IWR=6
      OPEN (UNIT=5,FILE=' ')
      OPEN (UNIT=6,FILE=' ')
```

2.4.1 Program NONLTB

```
      CALL INPUT(E,ARA,AL,QFI,X,Z,ANIT,IBC,IRE,IWR,AK14S,ID14S,NDSP,
     1        NV,AK15
     2        POISS,ITYEL)
C     ARGUMENTS IN LINE ABOVE NOT USED FOR SHALLOW TRUSS
C     BELOW RELEVANT TO DEEP TRUSS BUT LEAVE FOR SHALLOW TRUSS
      ALN=AL
      ARN=ARA
C
      READ (IRE*) (PT(I),I=1,NV)
      WRITE (IWR,2000) (PT(I),I=1,NV)
 2000 FORMAT(/1X,'STRATING PREDICTOR DISPS ARE',/,1X,6G12.5/)
      READ (IRE*) BETOK,IWRIT
 2001 FORMAT(/,1X,'CONV. TOL FACTOR, BETOK= ',G12.5,/,1X,
     2    'DIAGNOSTIC WRITE CONTROL(IWRIT)= ',15,/,3X,
     3    '0=NO, 1=YES'/)
C     SET TO NEWTON-RAPHSON ITERATIONS
      ITERY=1
C
      CALL ITER(PT,AN,BETOK,QFI,IBC,IWRIT,IWR,AKTS,D,ITERTY,NV,
     1        GM,FI,REAC)
C
      WRITE (IWR,1004) (PT(I),I=1,NV)
 1004 FORMAT(/,1X,'FINAL TOTAL DISPLACEMENTS ARE',/,1X,5G12.5/)
      WRITE (IWR,1006) (REAC(I),I=1,NV)
 1006 FORMAT(/,1X,'FINAL REACTIONS ARE',/,5G12.5/)
      WRITE (IWR,1005) AN
 1005 FORMAT(/,1X,'AXIAL FORCE IN BAR IS ',G12.5/)
      STOP 'NONLTB'
      END
```

2.4.2 Flowchart and computer listing for subroutine ITER



2.4.2 Flowchart and computer listing for subroutine ITER

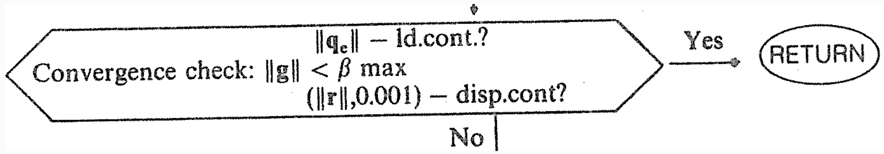
```
      SUBROUTINE ITER(PT,AN,BETOK,QEX,IBC,IWRIT,IWR,AKTS,D,ITERY,NV,
     1        GM,FI,REAC)
C     INPUTS PREDICTOR DISPS. PT(NV) AND EX. FORCE VECTOR QEX(NV)
C     ALSO BETOK=CONV. TOL, IBC=B. CON COUNTER
C     ITERATES TO EQUILIBRIUM: OUTPUTS NEW PT AND FORCE IN BAR, AN
C     IF ITERTY (INPUT)=1 USES FULL N-R, =2 USES MOD N-R
C     IN LATTER CASE, AKTS AND D INPUT AS CROUT FACTORS (D=PIVOTS)
C     LOCAL ARRAY IS AKTE=EL. STIF. MATRIX
C     ARGUMENTS IN COMMON/DAT2/ AND ARRAY X NOT USED FOR SHALLOW TRUSS
C
      IMPLICIT DOUBLE PRECISON (A-H,O-Z)
      COMMON /DAT/ X(2),Z(2),E,ARA,AL,ID14S(4),NDSP,ANIT,AK15
      COMMON /DAT2/ ARN,POISS,ALN,ITYEL
      DIMENSION PT(NV),QEX(NV),IBC(NV),REAC(NV)
      DIMENSION FI(NV),GM(NV),AKTS(NV,NV),D(NV),AKTE(4,4)
C
      SMALL=0.1D-2
      NITMAX=16
      IMOD=1
      IF (ITERY.EQ.1) IMOD=3
C
```

2.4.2 Flowchart and computer listing for subroutine ITER

```
      DO 100 ITE=1,NITMAX
C
        IF (IWRIT.EQ.1) WRITE (IWR,1005) ITERY
 1005   FORMAT(/,IX,'ITERATIVE LOOP WITH ITERY=',I5/)
C       BELOW CALCS FORCE IN BAR (AN)
        CALL FORCE(AN,ANIT,E,ARA,AL,X,Z,PT,IWRIT,IWR,
     1     ITYEL,ARN,ALN,POISS)
C       ABOVE ARGUMENTS NOT USED FOR SHALLOW-TRUSS
C
C       ABOVE CALCS FORCE IN BAR, AN: BELOW TAN STIFF AKT
C       (IF NR) AND INT. FORCE VECT. FI
        CALL ELEMENT(FI,AKTE,AN,X,Z,PT,E,ARA,AL,IWRIT,IWR,IMOD,
     1     ITYEL,ALN,ARN)
C       ABOVE ARGUMENTS NOT USED FOR SHALLOW TRUSS
C
C       BELOW PUTS EL. STIFF. MAT., AKTE, IN STR. STIFF., AKTS AND
C       ADDS IN EFFECTS OF VARIOUS LINEAR SPRINGS (IF NR)
C       ALSO MODIFIES INT. FORCE VECT. FI FOR SPRING EFFECTS
        CALL ELSTRUC(AKTE,AKTS,NV,AK15,DI14S,AK14S,NDSP,FI,PT,
     1       IMOD,IWRIT,IWR)
C
C       BELOW FORMS GM=OUT-OF-BALANCE FORCE VECTOR
C       AND REACTION VECTOR
        DO 10 I=1,NV
          GM(I)=0.D0
          REAC(I)=FI(I)
          IF (IBC(I).EQ.0) THEN
            GM(I)=QEX(I)-FI(I)
          ENDIF
   10   CONTINUE
   67   FORMAT(6G13.5)
   47   FORMAT(5I5)
C
C       OVERWRITE SPRING REACTIONS TERMS
        IF (NDSP.NE.0) THEN
          DO 50 I=1,NDSP
            REAC(ID14S(I))=-AK14S(I)*PT(ID14S(I))
   50     CONTINUE
        ENDIF
```

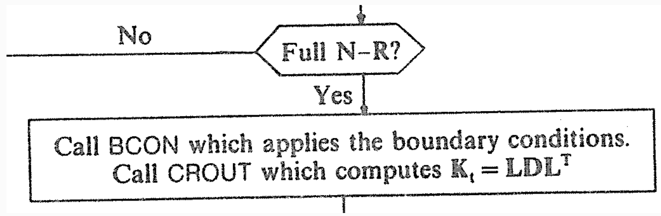
Call FORCE which computes $N = \text{fn.}(EA, l, z, p)$.
Call ELEMENT and compute the internal force vector, q_i for the truss.
If full N–R iterations, also compute K_t.
Call ELSTRUC which modifies q_i for the effects of the linear springs and, if full N–R, puts the element stiffness matrix into the struct. stiff. matrix and modifies for the effects of linear springs
compute the out-of-balance force vector, $-g = q_e - q_i$,
and creates reaction vector, $r = q_i$,
except at earthed springs where $r_i = -K_{si} p_i$

2.4.2 Flowchart and computer listing for subroutine ITER



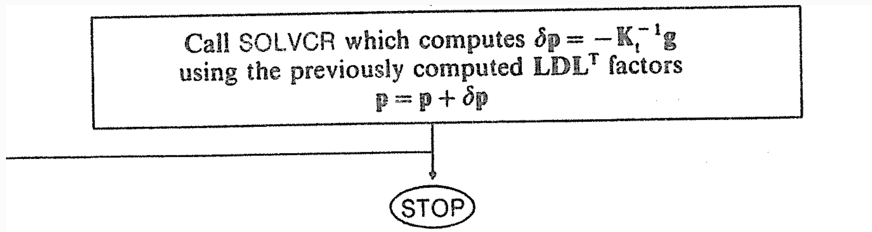

```
C          BELOW CHECKS CONVERGENCE
           FNORM=0.D0
           GNORM=0.D0
           RNORM=0.D0
           IDSP=0
           DO 20 I=1,NV
             IF (IBC(I).EQ.0) FNORM=FNORM+QEX(I)*QEX(I)
             IF (IBC(I).EQ.-1) IDSP=1
             RNORM=RNORM+REAC(I)*REAC(I)
             GNORM=GNORM+GM(I)*GM(I)
   20      CONTINUE
           FNORM=DSQRT(FNORM)
           GNORM=DSQRT(GNORM)
           RNORM=DSQRT(RNORM)
           BAS=MAX(FNORM,SMALL)
C          BELOW DISP. CONTROL
           IF (IDSP.EQ.1) BAS=MAX(RNORM,SMALL)
           BET=GNORM/BAS
           ITEM=ITE-1
           WRITE (IWR,1001) ITEM,BET
 1001      FORMAT(/1X,'ITERN. NO.= ',I5,'CONV. FAC.= ',G13.5/)
           IF (WRIT.EQ.1) WRITE (IWR,1003) (GM(I),I=1,NV)
 1003      FORMAT(/1X,'OUT-OF-BAL. FORCE VECTOR= ',1X,4G13.5/)
           IF (BET.LE.BETOK) GO TO 200
C
```

2.4.2 Flowchart and computer listing for subroutine ITER




```
            IF (ITERTY.EQ.1) THEN
              CALL BCON(AKTS,IBC,NV,GM,IWRIT,IWR)
C             ABOVE APPLIES B. CONDITIONS
              CALL CROUT(AKTS,D,NV,IWRIT,IWR)
C             ABOVE FORMS LDL(TRAN) FACTORISATION INTO AKTS AND D
            ENDIF
```

2.4.2 Flowchart and computer listing for subroutine ITER




```
C
          CALL SLOVCR(AKTS,D,GM,NV,IWRIT,IWR)
C         ABOVE KETS ITER. DISP. CHANGE IN GM
C
          DO 30 I=1,NV
            IF (IBC(I).EQ.0) THEN
              PT(I)=PT(I)+GM(I)
            ELSE
              PT(I)=QEX(I)
            ENDIF
   30     CONTINUE
C         ABOVE UPDATES DISPS.
          IF (WRIT.EQ.1) WRITE (IWR,1004) (PT(I)=I,NV)
 1004     FORMAT(/1X,'TOTAL DISPS ARE',1X,6G13.5/)
C
  100 CONTINUE
C
      WRITE (IWR,1002)
 1002 FORMAT(/1X,'FAILED TO CONVERGENCE****'/)
      STOP 'ITER 100'
C
  200 CONTINUE
      RETURN
      END
```


Thank you!