Equation solving and storage

Kim Lau Nielsen

Department of Civil and Mechanical Engineering

Section of Solid Mechanics
Technical University of Denmark

You already tried this!

Methods

Equation solving

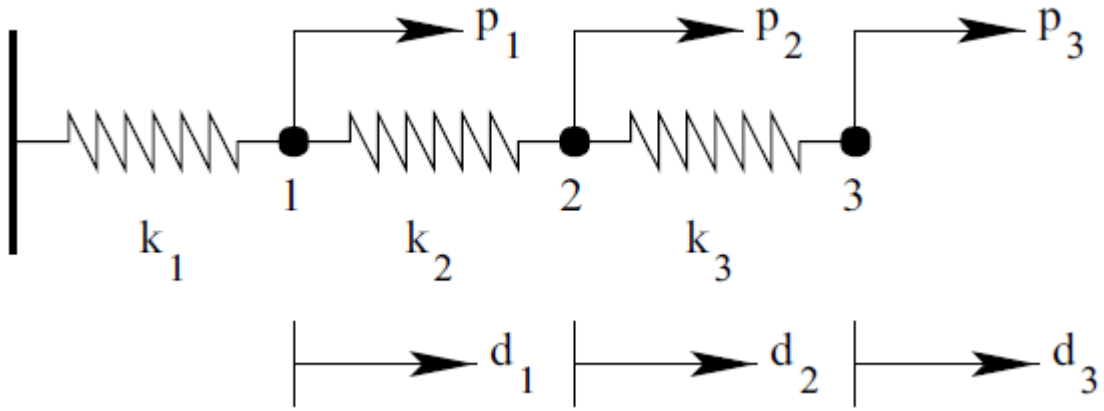
- Direct methods
  - Gauss elimination
  - LU factorization
  - Cholesky factorization
  - Frontal solvers (LU of sparse matrices. Assemble and elimination on one subset of elements at the time)
- Iterative methods

Storage

- Full storage
- Reduced storage
  - Band storage
  - Sparse storage

Gauss elimination & full storage

Simple 3 d.o.f system:



Global equation system:

$$\begin{bmatrix} k_1+k_2 & -k_2 & 0 \\ -k_2 & k_2+k_3 & -k_3 \\ 0 & -k_3 & k_3 \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} p_1 \\ p_2 \\ p_3 \end{Bmatrix}$$

Gauss elimination & full storage

Let; $k_1 = k_2 = k_3 = 1$, and $p_1 = p_2 = 0, p_3 = 1$, then:

$$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix}$$

Step 1: Elimination (of non-zeros below diagonal)

$$\begin{bmatrix} 2 & -1 & 0 \\ 0 & \frac{3}{2} & -1 \\ 0 & -1 & 1 \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix}$$

$$\begin{bmatrix} 2 & -1 & 0 \\ 0 & \frac{3}{2} & -1 \\ 0 & 0 & \frac{1}{3} \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix}$$

Upper triangular matrix

Gauss elimination & full storage

The system of equations is then:

$$\begin{bmatrix} 2 & -1 & 0 \\ 0 & \frac{3}{2} & -1 \\ 0 & 0 & \frac{1}{3} \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix}$$

Step 2: Backward substitution

$$\frac{1}{3}d_3 = 1 \Rightarrow \; d_3 = 3$$

$$\frac{3}{2}d_2 - d_3 = 0 \Rightarrow \; \frac{3}{2}d_2 = 3 \Rightarrow \; d_2 = 2$$

$$2d_1 - d_2 = 0 \Rightarrow \; 2d_1 = 2 \Rightarrow \; d_1 = 1$$

Additional steps required: pivoting (Interchange rows and columns)

Gauss elimination & full storage

Elimination:

```
DO s = 1, n-1
 DO i = s+1, n
  a = K(i,s)/K(s,s)
  P(i) = P(i) - a*P(s)
  DO j = s+1, n
   K(i,j) = K(i,j) - a*K(s,j)
  END DO
 END DO
END DO
```

$$\text{operations} = \frac{1}{6}n(n-1)(4n+7) \sim \frac{2}{3}n^3$$

Backward substitution:

```
D(n) = P(n)/K(n,n)
DO i = n-1, 1, -1
 a = P(i)
 DO j = i+1, n
  a = a - K(i,j)*D(j)
 END DO
 D(i) = a/K(i,i)
END DO
```

$$\text{operations} = n^2$$

LU factorization (Generalized Gauss elimination)

Original system of equations: $[K]\{D\} = \{P\}$

Factorization (elimination): $[L][U]\{D\} = \{P\}$

Lower triangular matrix

Upper triangular matrix

Forward substitution: $[L]\{z\} = \{P\}$

Backward substitution: $[U]\{D\} = \{z\}$

LU factorization (Generalized Gauss elimination)

Elimination:

```
DO s = 1, n-1
 DO i = s+1, n
  K(i,s) = K(i,s)/K(s,s)
  DO j = s+1, n
   K(i,j) = K(i,j) - K(i,s)*K(s,j)
  END DO
 END DO
END DO
```

$$\text{operations} \sim \frac{2}{3}n^3$$

Forward substitution:

```
DO i = 2, n
 a = 0
 DO j = 1, i-1
  a = a - K(i,j) * P(j)
 END DO
 P(i) = P(i) + a
END DO
```

Backward substitution:

```
P(n) = P(n)/K(n,n)
DO i = n-1, 1, -1
 a = 0
 DO j = i+1, n
  a = a - K(i,j)*P(j)
 END DO
 P(i) = a/K(i,i)
END DO
```

$$\text{operations} \sim 2n^2$$

Cholesky factorization (positive definite matrices)

Original equation: $[K]\{D\} = \{P\}$

Factorization: $[L][L]^T\{D\} = \{P\}$

Forward/backward substitution: $[L]\{z\} = \{P\}$ and $[L]^T\{D\} = \{z\}$

Computational costs (factorization)

$$\text{operations} \sim \frac{1}{3}n^3$$

Full storage (Basic approach)

Storage requirements for a 2D plane (square) structure with 99×99 (4 noded) elements

Number of nodes = 100×100 = 10000

Degrees of freedom = 2×10000 = 20000

Size of global stiffness matrix = 20000×20000 (= 400 million)

Size in memory (double precision: 8 bytes per number) ~ 3.2 Gb !!

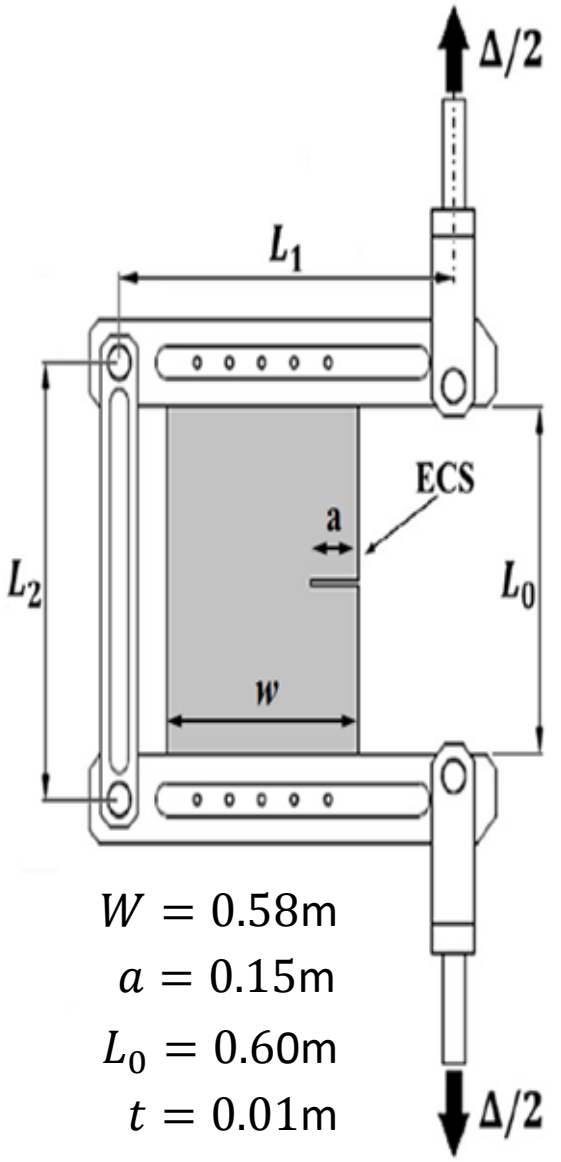
10 million d.o.fs in a 3D model

Size in memory (double precision: 8 bytes per number) ~ 800000 Gb !

Full Scale 3D Simulation with Tunned “Sweet Spot” Parameters

- Simonsen and Törnqvist (2004) plate

$N = 0.04$ and $f_0 = 0.014$



$W = 0.58\text{m}$

$a = 0.15\text{m}$

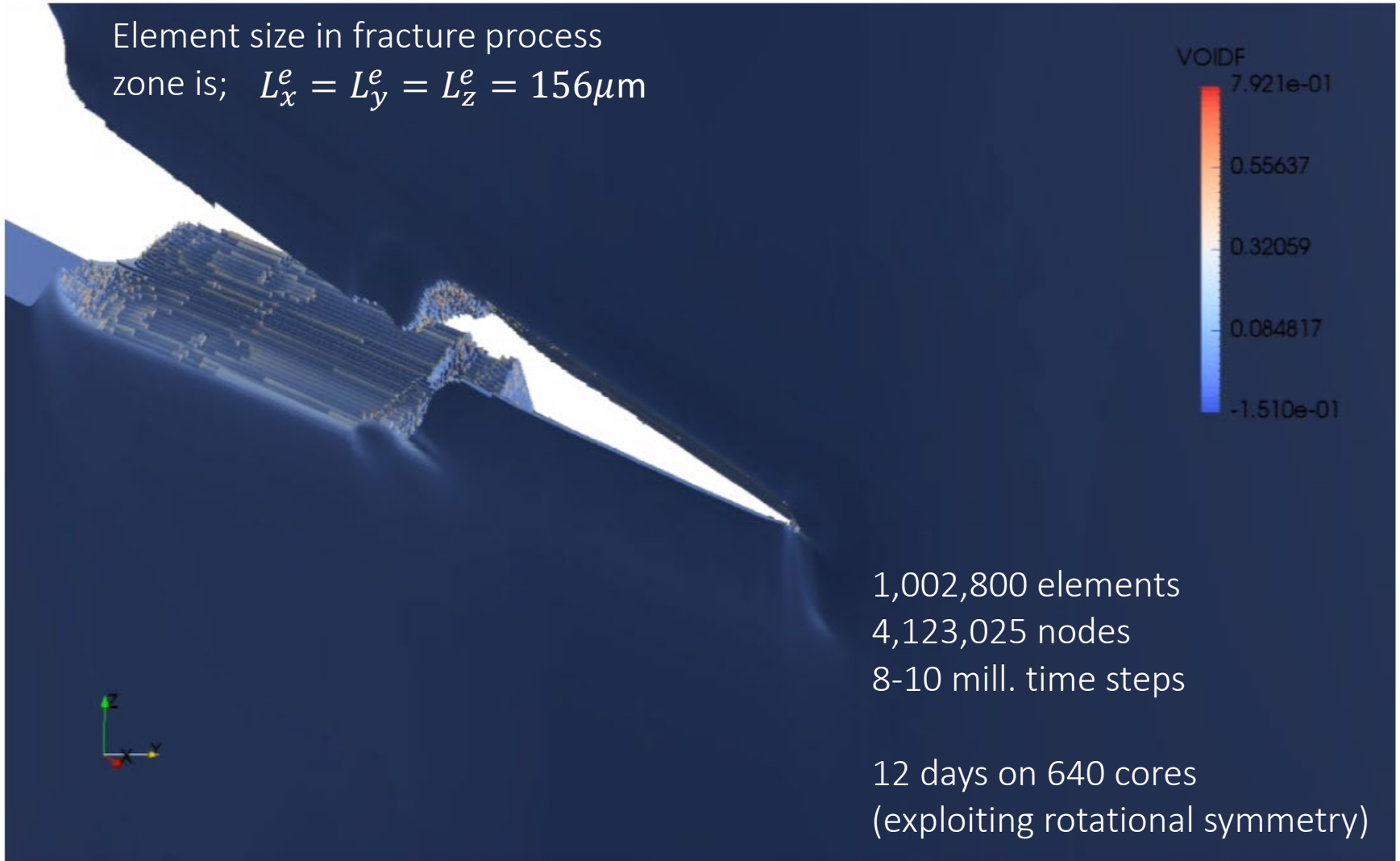
$L_0 = 0.60\text{m}$

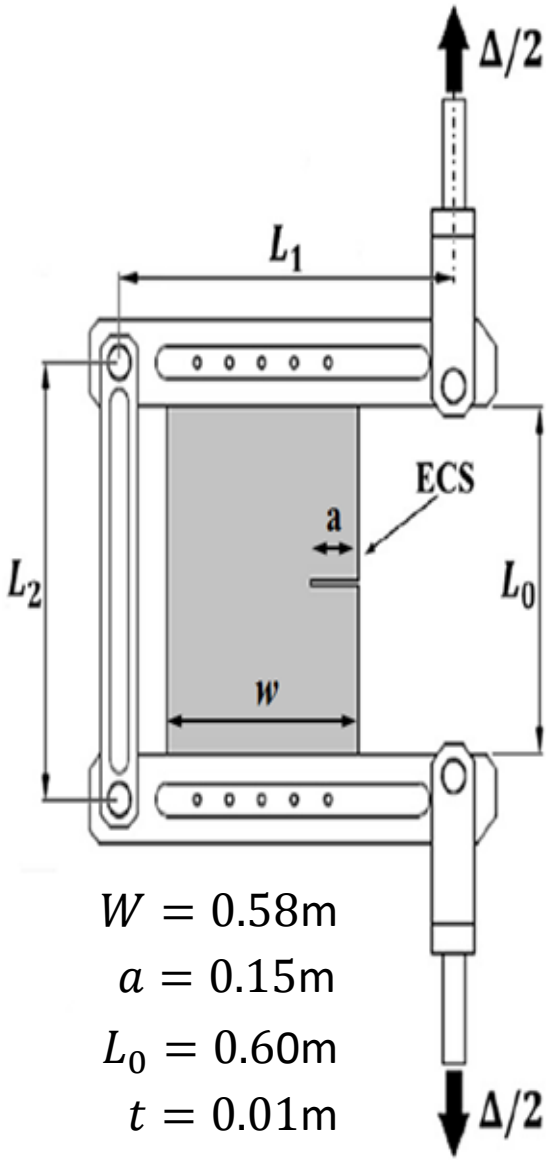
$t = 0.01\text{m}$

Full Scale 3D Simulation with Tunned “Sweet Spot” Parameters

- Simonsen and Törnqvist (2004) plate

$N = 0.04$ and $f_0 = 0.014$





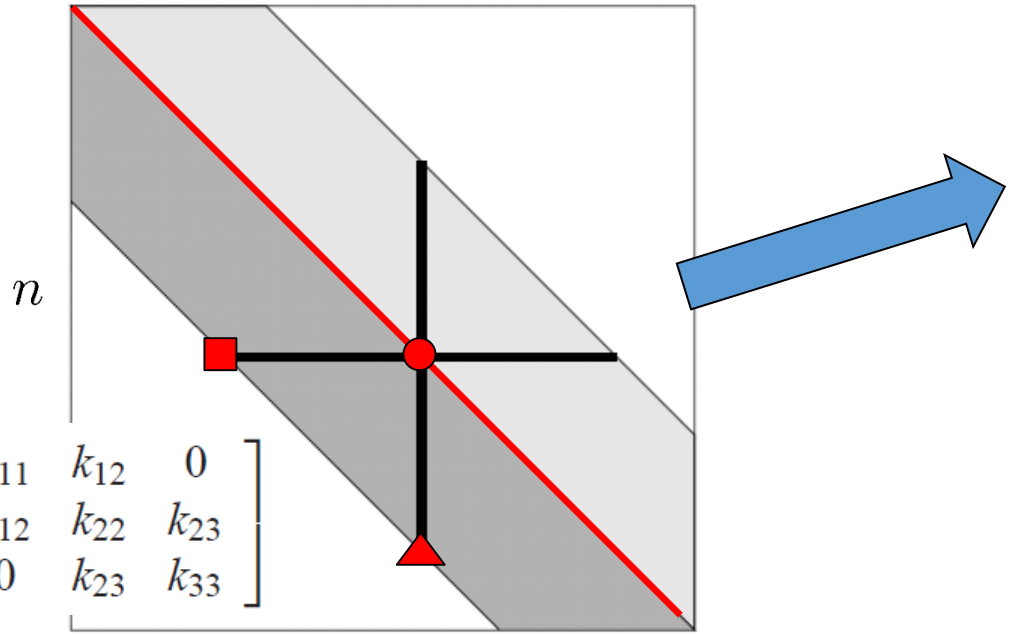
$W = 0.58$m
$a = 0.15$m
$L_0 = 0.60$m
$t = 0.01$m

Band storage

Original system of equations:

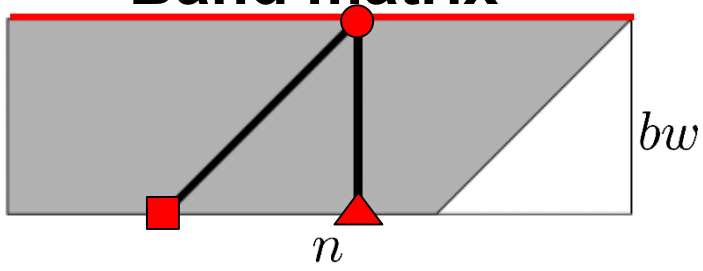
$$\begin{bmatrix} k_1+k_2 & -k_2 & 0 \\ -k_2 & k_2+k_3 & -k_3 \\ 0 & -k_3 & k_3 \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} p_1 \\ p_2 \\ p_3 \end{Bmatrix}$$

Square matrix



$$[K] = \begin{bmatrix} k_{11} & k_{12} & 0 \\ k_{12} & k_{22} & k_{23} \\ 0 & k_{23} & k_{33} \end{bmatrix}$$

Band matrix



bw : bandwidth

$$[\overline{K}] = \begin{bmatrix} k_{11} & k_{22} & k_{33} \\ k_{12} & k_{23} & \end{bmatrix}$$

Band storage

Storage requirements: $= n \times bw$

Computational costs for LU factorization:

Factorization: $\text{operations} \sim bw^2 \times \frac{1}{2}n$

Substitution: $\text{operations} \sim 2bw \times n$

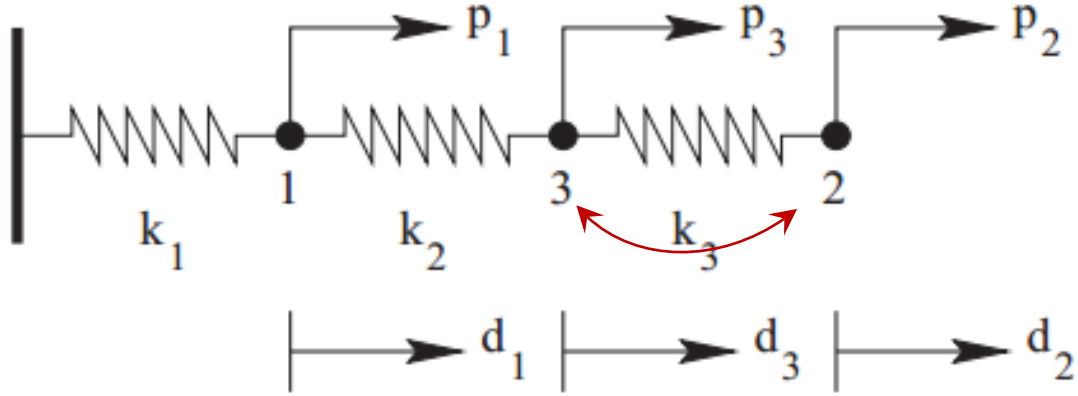
Compared to full storage, the savings for a 2D plane (square) structure with 99×99 (4 noded) elements (bw=204)

Storage: $\frac{bw}{n} = \frac{204}{20000} \approx 0.01$

Factorization: $\frac{3}{4}(\frac{bw}{n})^2 \approx 0.00008$

Band storage

Computational savings require careful node numbering !



Significant effect storage and factorization!

bw=3 for this system

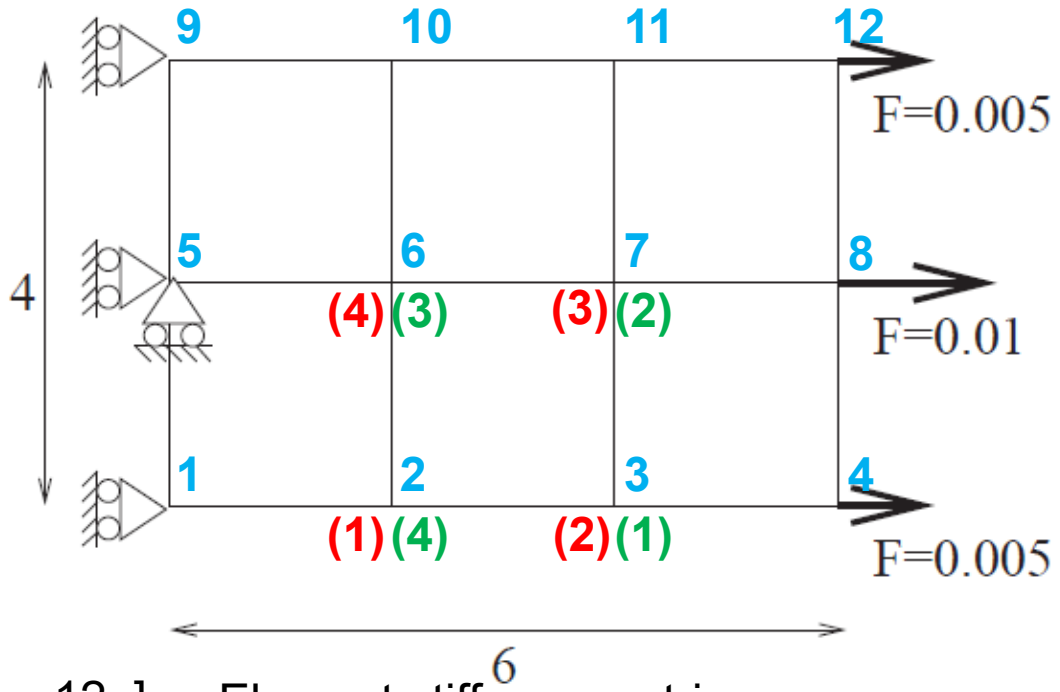
Programming tasks:

1) Compute bw
2) Assemble in band format
3) Correct for boundary conditions in band format
4) Call new band solver routines

Your tasks for today !

Assembling the banded stiffness matrix!

- Example from lecture notes



idof = [3 4 5 6 13 14 11 12]

idof = [5 6 13 14 11 12 3 4]

Element stiffness matrix

```
ke =

    11    12    13    14    15    16    17    18
    21    22    23    24    25    26    27    28
    31    32    33    34    35    36    37    38
    41    42    43    44    45    46    47    48
    51    52    53    54    55    56    57    58
    61    62    63    64    65    66    67    68
    71    72    73    74    75    76    77    78
    81    82    83    84    85    86    87    88
```

Assembling the banded stiffness matrix!

DTU

K =

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	11	12	13	14	0	0	0	0	17	18	15	16	0	0	0	0	0	0	0	0	0	0
0	0	21	22	23	24	0	0	0	0	27	28	25	26	0	0	0	0	0	0	0	0	0	0
0	0	31	32	33	34	0	0	0	0	37	38	35	36	0	0	0	0	0	0	0	0	0	0
0	0	41	42	43	44	0	0	0	0	47	48	45	46	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	71	72	73	74	0	0	0	0	77	78	75	76	0	0	0	0	0	0	0	0	0	0
0	0	81	82	83	84	0	0	0	0	87	88	85	86	0	0	0	0	0	0	0	0	0	0
0	0	51	52	53	54	0	0	0	0	57	58	55	56	0	0	0	0	0	0	0	0	0	0
0	0	61	62	63	64	0	0	0	0	67	68	65	66	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

idof = [3 4 5 6 13 14 11 12]

idof = [5 6 13 14 11 12 3 4]

Element stiffness matrix

ke =

11	12	13	14	15	16	17	18
21	22	23	24	25	26	27	28
31	32	33	34	35	36	37	38
41	42	43	44	45	46	47	48
51	52	53	54	55	56	57	58
61	62	63	64	65	66	67	68
71	72	73	74	75	76	77	78
81	82	83	84	85	86	87	88

Assembling the banded stiffness matrix!

K =

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	77	78	71	72	0	0	0	0	75	76	73	74	0	0	0	0	0	0	0	0	0	0
0	0	87	88	81	82	0	0	0	0	85	86	83	84	0	0	0	0	0	0	0	0	0	0
0	0	17	18	11	12	0	0	0	0	15	16	13	14	0	0	0	0	0	0	0	0	0	0
0	0	27	28	21	22	0	0	0	0	25	26	23	24	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	57	58	51	52	0	0	0	0	55	56	53	54	0	0	0	0	0	0	0	0	0	0
0	0	67	68	61	62	0	0	0	0	65	66	63	64	0	0	0	0	0	0	0	0	0	0
0	0	37	38	31	32	0	0	0	0	35	36	33	34	0	0	0	0	0	0	0	0	0	0
0	0	47	48	41	42	0	0	0	0	45	46	43	44	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

idof = [3 4 5 6 13 14 11 12]

idof = [5 6 13 14 11 12 3 4]

Element stiffness matrix

ke =

11	12	13	14	15	16	17	18
21	22	23	24	25	26	27	28
31	32	33	34	35	36	37	38
41	42	43	44	45	46	47	48
51	52	53	54	55	56	57	58
61	62	63	64	65	66	67	68
71	72	73	74	75	76	77	78
81	82	83	84	85	86	87	88

Alternative (better) storage

Skyline storage

Further exploitation of matrix structure (different bw for each column/row)

Sparse storage

Store only non-zero entries

Requires additional information on location of entries

Particular easy to use in Matlab-code (and efficient when combined with D = K\P command)

$$K = \text{zeros}(n, n) \quad \Rightarrow \quad K = \text{sparse}(n, n)$$

But: assembly is slow for large systems

Alternative (popular) solvers

Frontal solvers

Incomplete/partial factorization

Do not require assembly of global stiffness matrix

Iterative solvers

Do not require assembly of global stiffness matrix

Minimize the residual

Multigrid solvers

Alternate between meshes with different coarseness

Exercises

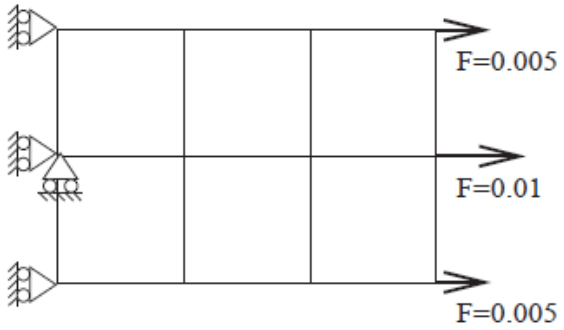
Exercise 6.1

Implement band storage of the global stiffness matrix and solve the FE-equation using ("bfactor" and "bsolve"). **Hint:** Remember to declare the size of the stiffness matrix in subroutine "initial". Test it on the problem from Figure 5.1 (it must also work for truss structures where you can compare with your Matlab code). There is a logical parameter named "banded" and a parameter for the bandwidth "bw" in the fedata module which are predefined for you. A few Fortran lines that will print the stiffness matrix in a compact format are

```
DO i = 1, bw
    print "(24(f4.2,tr1))", kmat(i,1:neqn)
END DO
PAUSE
```

Use "FlExtract" to generate Fortran input files for large (i.e. many elements) continuum structures and investigate time savings associated with the switch to band storage format. **Hint:** For time measurements in the Fortran code you may use the "stopwatch" routine given in the processor-module to measure cputime. The stopwatch is started by inserting the line "stopwatch('star')" in the Fortran code and the stopwatch is stopped (and the cpu-time is printed) by inserting the line "stopwatch('stop')" in the Fortran code. Switch off all debug options in the Fortran compiler during timing comparisons to get the best computational efficiency.

Exercises



Exercise 6.2

Compute element strains and stresses (e.g. ϵ_{11}, σ_{22}, *etc.) for the simple test case in Figure 6.4 and compare with analytical values.* **Hint:** Compute strains and stresses (in the centroid ($x = y = 0$)) in the subroutine "plane42rect_ss". Call the plotting routine "plot" with a vector containing the stresses (from the "displ" subroutine).
The "plane42rect_ss" routine is called with the command-line

```
call plane42rect_ss(xe, de, young, nu, estress, estrain)
```

where the input variables are element nodal coordinates, Young's modulus and Poisson's ratio and the outputs are the element stress and strain vectors.

Compute and plot Von Mises stresses (in the element centroids) and plot them for some more complex test cases (for example an L-shaped domain or a structure with an internal hole).

Compute and plot the principal stresses and directions (in radians) and store them in the vectors 'p1', 'p2' and 'angle'. **Hint:** The directions can be found using the Fortran function ψ =atan2(s,c)/2., where s and c are the sine and cosine values from Eq. 6.23. *Use the plotting routine "plot" to show the results graphically (see the subroutine 'plot' in the 'processor'-module for the options).*

Exercises

Exercise 6.3

Prepare your code to be able to evaluate global compliance as well as nodal stress values as function of discretization. A typical assignment is the cantilever problem in Figure 6.5. *a) Plot the compliance as a function of the element size and the square of the element size* h^2 *and b) plot the Von Mises stresses in points A and B as functions of the element size h and the square of the element size* h^2. **Hint:** the best agreement with theory is obtained by calculating the stresses in the upper element corners at points A and B.

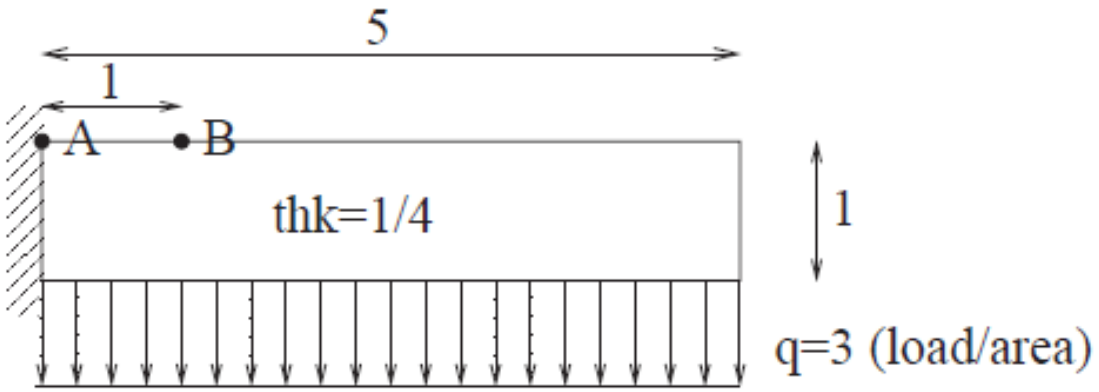


Figure 6.5: Cantilever test example.

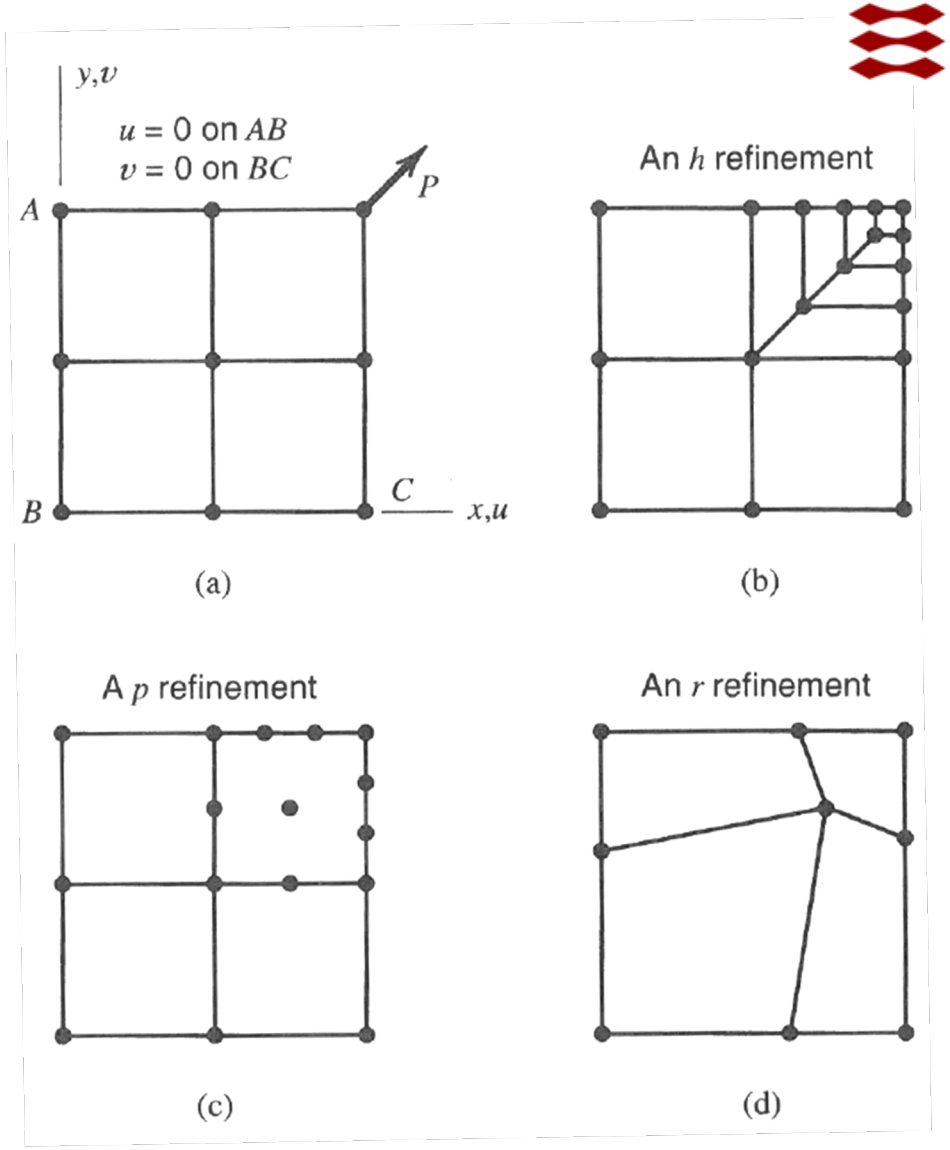
Mesh refinement

Three different approaches

h-refinement (add elements of the same type)

p-refinement (increase polynomial degree)

r-refinement (relocate existing nodes)



(p. 319 in Cook book)

Today you are asked to (using h-refinement):

1) Plot compliance vs. h and h^2

2) Plot stress vs. h and h^2