C A L F E M

A FINITE ELEMENT TOOLBOX
Version 3.4

P-E AUSTRELL, O DAHLBLOM, J LINDEMANN,
A OLSSON, K-G OLSSON, K PERSSON, H PETERSSON,
M RISTINMAA, G SANDBERG, P-A WERNBERG

C A L F E M

A FINITE ELEMENT TOOLBOX

Version 3.4

P-E AUSTRELL, O DAHLBLOM, J LINDEMANN,
A OLSSON, K-G OLSSON, K PERSSON, H PETERSSON,
M RISTINMAA, G SANDBERG, P-A WERNBERG

E-mail address:

 `calfem@byggmek.lth.se`

Homepage:

 `http://www.byggmek.lth.se/Calfem`

Contacts:
 The Division of Structural Mechanics
 Lund University
 PO Box 118
 SE–221 00 Lund
 SWEDEN
 Phone: +46 46 222 0000
 Fax: +46 46 222 4420

Preface

CALFEM® is an interactive computer program for teaching the finite element method (FEM). The name CALFEM is an abbreviation of "Computer Aided Learning of the Finite Element Method". The program can be used for different types of structural mechanics problems and field problems.

CALFEM, the program and its built-in philosophy have been developed at the Division of Structural Mechanics, Lund University, starting in the late 70's. Many coworkers, former and present, have been engaged in the development at different stages.

This release represents the latest development of CALFEM. The functions for finite element applications are all MATLAB functions (.m-files) as described in the MATLAB manual. We believe that this environment increases the versatility and handling of the program and, above all, the ease of teaching the finite element method. CALFEM also works with Octave, presently with exception for some graphical functions.

Lund, October 7, 2004

The authors

Contents

1 Introduction

The computer program CALFEM is a MATLAB toolbox for finite element applications. This manual concerns mainly the finite element functions, but it also contains descriptions of some often used MATLAB functions.

The finite element analysis can be carried out either interactively or in a batch oriented fashion. In the interactive mode the functions are evaluated one by one in the MATLAB command window. In the batch oriented mode a sequence of functions are written in a file named .m-file, and evaluated by writing the file name in the command window. The batch oriented mode is a more flexible way of performing finite element analysis because the .m-file can be written in an ordinary editor. This way of using CALFEM is recommended because it gives a structured organization of the functions. Changes and reruns are also easily executed in the batch oriented mode.

A command line consists typically of functions for vector and matrix operations, calls to functions in the CALFEM finite element library or commands for workspace operations. An example of a command line for a matrix operation is

$$\mathsf{C} = \mathsf{A} + \mathsf{B}'$$

where two matrices A and B' are added together and the result is stored in matrix C. The matrix B' is the transpose of B. An example of a call to the element library is

$$\mathsf{Ke} = \mathsf{bar1e(k)}$$

where the two-by-two element stiffness matrix $\mathbf{K}^e$ is computed for a spring element with spring stiffness k, and is stored in the variable Ke. The input argument is given within parentheses ( ) after the name of the function. Some functions have multiple input arguments and/or multiple output arguments. For example

$$[\mathsf{lambda}, \mathsf{X}] = \mathsf{eigen(K, M)}$$

computes the eigenvalues and eigenvectors to a pair of matrices K and M. The output variables - the eigenvalues stored in the vector lambda and the corresponding eigenvectors stored in the matrix X - are surrounded by brackets [ ] and separated by commas. The input arguments are given inside the parentheses and also separated by commas.

The statement

> help   *function*

provides information about purpose and syntax for the specified function.

The available functions are organized in groups as follows. Each group is described in a separate chapter.

Groups of functions	
General purpose commands	for managing variables, workspace, output etc
Matrix functions	for matrix handling
Material functions	for computing material matrices
Element functions	for computing element matrices and element forces
System functions	for setting up and solving systems of equations
Statement functions	for algorithm definitions
Graphics functions	for plotting

2 General purpose functions

The general purpose functions are used for managing variables and workspace, control of output etc. The functions listed here are a subset of the general purpose functions described in the MATLAB manual. The functions can be divided into the following groups

Managing commands and functions	
help	Online documentation
type	List .m-file
what	Directory listing of .m-, .mat- and .mex-files
...	Continuation
%	Write a comment line

Managing variables and the workspace	
clear	Remove variables from workspace
disp	Display variables in workspace on display screen
load	Retrieve variable from disk and load in workspace
save	Save matrix bank variable on disk
who,	List directory of variables in workspace
whos	

Working with files and controlling the command window	
diary	Save session in a named file
echo	Control output on the display screen
format	Control the output display format
quit	Stop execution and exit from the CALFEM program

Purpose:

Remove variables from workspace.

Syntax:

clear
clear *name1 name2 name3 ...*

Description:

clear removes all variables from workspace.

clear *name1 name2 name3 ...* removes specified variables from workspace.

Note:

This is a MATLAB built-in function. For more information about the clear function, type help clear.

Purpose:

Save session in a disk file.

Syntax:

diary *filename*
diary off
diary on

Description:

diary *filename* writes a copy of all subsequent keyboard input and most of the resulting output (but not graphs) on the named file. If the file *filename* already exists, the output is appended to the end of that file.

diary off stops storage of the output.

diary on turns it back on again, using the current filename or default filename diary if none has yet been specified.

The diary function may be used to store the current session for later runs. To make this possible, finish each command line with semicolon ';' to avoid the storage of intermediate results on the named diary file.

Note:

This is a MATLAB built-in function. For more information about the diary function, type help diary.

Purpose:

Display a variable in matrix bank on display screen.

Syntax:

disp(A)

Description:

disp(A) displays the matrix A on the display screen.

Note:

This is a MATLAB built-in function. For more information about the disp function, type help disp.

Purpose:

Control output on the display screen.

Syntax:

echo on
echo off
echo

Description:

echo on turns on echoing of commands inside Script-files.

echo off turns off echoing.

echo by itself, toggles the echo state.

Note:

This is a MATLAB built-in function. For more information about the echo function, type help echo.

Purpose:

Control the output display format.

Syntax:

See the listing below.

Description:

format controls the output format. By default, MATLAB displays numbers in a short format with five decimal digits.

Command	Result	Example
format short	5 digit scaled fixed point	3.1416
format long	15 digit scaled fixed point	3.14159265358979
format short e	5 digit floating point	3.1416e+000
format long e	16 digit floating point	3.141592653589793e+000

Note:

This is a MATLAB built-in function. For more information about the format function, type help format.

Purpose:

Display a description of purpose and syntax for a specific function.

Syntax:

help *function name*

Description:

help provides an online documentation for the specified function.

Example:

Typing

```
>> help bar1e
```

yields

```
  Ke=bar1e(ep)
 ---------------------------------------------------------
 PURPOSE
  Compute element stiffness matrix
  for spring (analog) element.

 INPUT:  ep = [k]; spring stiffness or analog quantity.

 OUTPUT: Ke : stiffness matrix, dim(Ke)= 2 x 2
 ---------------------------------------------------------
```

Note:

This is a MATLAB built-in function. For more information about the help function, type help help.

Purpose:

Retrieve variable from disk and load in workspace.

Syntax:

load *filename*
load *filename.ext*

Description:

load *filename* retrieves the variables from the binary file *filename*.mat.

load *filename.ext* reads the ASCII file *filename.ext* with numeric data arranged in m rows and n columns. The result is an m-by-n matrix residing in workspace with the name *filename*, i.e. with the extension stripped.

Note:

This is a MATLAB built-in function. For more information about the load function, type help load.

Purpose:

Terminate CALFEM session.

Syntax:

quit

Description:

quit *filename* terminates the CALFEM without saving the workspace.

Note:

This is a MATLAB built-in function. For more information about the quit function, type help quit.

Purpose:

Save workspace variables on disk.

Syntax:

save *filename*
save *filename variables*
save *filename variables* -ascii

Description:

save *filename* writes all variables residing in workspace in a binary file named *filename*.mat

save *filename variables* writes named variables, separated by blanks, in a binary file named *filename*.mat

save *filename variables* -ascii writes named variables in an ASCII file named *filename*.

Note:

This is a MATLAB built-in function. For more information about the save function, type help save.

Purpose:

List file.

Syntax:

type *filename*

Description:

type *filename* lists the specified file. Use path names in the usual way for your operating system. If a filename extension is not given, .m is added by default. This makes it convenient to list the contents of .m-files on the screen.

Note:

This is a MATLAB built-in function. For more information about the type function, type help type.

Purpose:

Directory listing of .m-files, .mat-files and .mex-files.

Syntax:

what
what *dirname*

Description:

what lists the .m-files, .mat-files and .mex-files in the current directory.

what *dirname* lists the files in directory *dirname* in the MATLAB search path. The syntax of the path depends on your operating system.

Note:

This is a MATLAB built-in function. For more information about the what function, type help what.

Purpose:

List directory of variables in matrix bank.

Syntax:

who
whos

Description:

who lists the variables currently in memory.

whos lists the current variables and their size.

Examples:

who

Your variables are:

A   B   C
K   M   X
k   lambda

whos

name	size	elements	bytes	density	complex
A	3-by-3	9	72	Full	No
B	3-by-3	9	72	Full	No
C	3-by-3	9	72	Full	No
K	20-by-20	400	3200	Full	No
M	20-by-20	400	3200	Full	No
X	20-by-20	400	3200	Full	No
k	1-by-1	1	8	Full	No
lambda	20-by-1	20	160	Full	No

Grand total is 1248 elements using 9984 bytes

Note:

These are MATLAB built-in functions. For more information about the functions, type help who or help whos.

Purpose:

Continuation.

Syntax:

...

Description:

An expression can be continued on the next line by using ... .

Note:

This is a MATLAB built-in function.

Purpose:

Write a comment line.

Syntax:

% *arbitrary text*

Description:

An arbitrary text can be written after the symbol %.

Note:

This is a MATLAB built-in character.

3   Matrix functions

The group of matrix functions comprises functions for vector and matrix operations and also functions for sparse matrix handling. MATLAB has two storage modes, full and sparse. Only nonzero entries and their indices are stored for sparse matrices. Sparse matrices are not created automatically. But once initiated, sparsity propagates. Operations on sparse matrices produce sparse matrices and operations on a mixture of sparse and full matrices also normally produce sparse matrices.

The following functions are described in this chapter:

Vector and matrix operations	
[ ] ( ) =	Special characters
' . , ;	Special characters
:	Create vectors and do matrix subscripting
+ − ∗ /	Matrix arithmetic
abs	Absolute value
det	Matrix determinant
diag	Diagonal matrices and diagonals of a matrix
inv	Matrix inverse
length	Vector length
max	Maximum element(s) of a matrix
min	Minimum element(s) of a matrix
ones	Generate a matrix of all ones
red	Reduce the size of a square matrix
size	Matrix dimensions
sqrt	Square root
sum	Sum of the elements of a matrix
zeros	Generate a zero matrix

Sparse matrix handling	
full	Convert sparse matrix to full matrix
sparse	Create sparse matrix
spy	Visualize sparsity structure

Purpose:

Special characters.

Syntax:

[ ] ( ) = ' . , ;

Description:

[ ]	Brackets are used to form vectors and matrices.
( )	Parentheses are used to indicate precedence in arithmetic expressions and to specify an element of a matrix.
=	Used in assignment statements.
'	Matrix transpose. X' is the transpose of X. If X is complex, the apostrophe sign performs complex conjugate as well. Do $X.'$ if only the transpose of the complex matrix is desired
.	Decimal point. $314/100$, 3.14 and $0.314e1$ are all the same.
,	Comma. Used to separate matrix subscripts and function arguments.
;	Semicolon. Used inside brackets to end rows. Used after an expression to suppress printing or to separate statements.

Examples:

By the statement

$$a = 2$$

the scalar a is assigned a value of 2. An element in a matrix may be assigned a value according to

$$A(2, 5) = 3$$

The statement

$$D = [\, 1 \ 2 \,; \ 3 \ 4]$$

results in matrix

$$D = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

stored in the matrix bank. To copy the contents of the matrix D to a matrix E, use

$$E = D$$

The character ' is used in the following statement to store the transpose of the matrix A in a new matrix F

$$F = A'$$

Note:

These are MATLAB built-in characters.

Purpose:

Create vectors and do matrix subscripting.

Description:

The colon operator uses the following rules to create regularly spaced vectors:

j : k   is the same as [j, j + 1, ... , k]

j : i : k is the same as [j, j + i, j + 2i, ... , k]

The colon notation may also be used to pick out selected rows, columns, and elements of vectors and matrices:

A(: , j) is the j :th column of A

A(i , :) is the i :th row of A

Examples:

The colon ':' used with integers

$$d = 1 : 4$$

results in a row vector

$$d = [\ 1\ \ 2\ \ 3\ \ 4\ ]$$

stored in the workspace.

The colon notation may be used to display selected rows and columns of a matrix on the terminal. For example, if we have created a 3-times-4 matrix D by the statement

$$D = [\ d\ ;\ 2 * d\ ;\ 3 * d\ ]$$

resulting in

$$D = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \end{bmatrix}$$

columns three and four are displayed by entering

$$D(: , 3 : 4)$$

resulting in

$$D(: , 3 : 4) = \begin{bmatrix} 3 & 4 \\ 6 & 8 \\ 9 & 12 \end{bmatrix}$$

In order to copy parts of the D matrix into another matrix the colon notation is used as

$$E(\ 3:4\ ,\ 2:3\ ) = D(\ 1:2\ ,\ 3:4\ )$$

Assuming the matrix E was a zero matrix before the statement is executed, the result will be

$$E = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 6 & 8 & 0 \end{bmatrix}$$

Note:

This is a MATLAB built-in character.

Purpose:

Matrix arithmetic.

Syntax:

A + B
A − B
A ∗ B
A/s

Description:

Matrix operations are defined by the rules of linear algebra.

Examples:

An example of a sequence of matrix-to-matrix operations is

$$D = A + B - C$$

A matrix-to-vector multiplication followed by a vector-to-vector subtraction may be defined by the statement

$$b = c - A * x$$

and finally, to scale a matrix by a scalar s we may use

$$B = A/s$$

Note:

These are MATLAB built-in operators.

Purpose:

Absolute value.

Syntax:

B=abs(A)

Description:

B=abs(A) computes the absolute values of the elements of matrix A and stores them in matrix B.

Examples:

Assume the matrix

$$C = \begin{bmatrix} -7 & 4 \\ -3 & -8 \end{bmatrix}$$

The statement D=abs(C) results in a matrix

$$D = \begin{bmatrix} 7 & 4 \\ 3 & 8 \end{bmatrix}$$

stored in the workspace.

Note:

This is a MATLAB built-in function. For more information about the abs function, type help abs.

Purpose:

Matrix determinant.

Syntax:

a=det(A)

Description:

a=det(A) computes the determinant of the matrix A and stores it in the scalar a.

Note:

This is a MATLAB built-in function. For more information about the det function, type help det.

Purpose:

Diagonal matrices and diagonals of a matrix.

Syntax:

M=diag(v)
v=diag(M)

Description:

For a vector v with n components, the statement M=diag(v) results in an $n \times n$ matrix M with the elements of v as the main diagonal.

For a $n \times n$ matrix M, the statement v=diag(M) results in a column vector v with n components formed by the main diagonal in M.

Note:

This is a MATLAB built-in function. For more information about the diag function, type help diag.

Purpose:

Convert sparse matrices to full storage class.

Syntax:

A=full(S)

Description:

A=full(S) converts the storage of a matrix from sparse to full. If A is already full, full(A) returns A.

Note:

This is a MATLAB built-in function. For more information about the full function, type help full.

Purpose:

Matrix inverse.

Syntax:

B=inv(A)

Description:

B=inv(A) computes the inverse of the square matrix A and stores the result in the matrix B.

Note:

This is a MATLAB built-in function. For more information about the inv function, type help inv.

Purpose:

Vector length.

Syntax:

n=length(x)

Description:

n=length(x) returns the dimension of the vector x.

Note:

This is a MATLAB built-in function. For more information about the length function, type help length.

Purpose:

Maximum element(s) of a matrix.

Syntax:

b=max(A)

Description:

For a vector a, the statement b=max(a) assigns the scalar b the maximum element of the vector a.

For a matrix A, the statement b=max(A) returns a row vector b containing the maximum elements found in each column vector in A.

The maximum element found in a matrix may thus be determined by c=max(max(A)).

Examples:

Assume the matrix B is defined as

$$B = \begin{bmatrix} -7 & 4 \\ -3 & -8 \end{bmatrix}$$

The statement d=max(B) results in a row vector

$$d = \begin{bmatrix} -3 & 4 \end{bmatrix}$$

The maximum element in the matrix B may be found by e=max(d) which results in the scalar e = 4.

Note:

This is a MATLAB built-in function. For more information about the max function, type help max.

Purpose:

Minimum element(s) of a matrix.

Syntax:

b=min(A)

Description:

For a vector a, the statement b=min(a) assigns the scalar b the minimum element of the vector a.

For a matrix A, the statement b=min(A) returns a row vector b containing the minimum elements found in each column vector in A.

The minimum element found in a matrix may thus be determined by c=min(min(A)).

Examples:

Assume the matrix B is defined as

$$B = \begin{bmatrix} -7 & 4 \\ -3 & -8 \end{bmatrix}$$

The statement d=min(B) results in a row vector

$$d = \begin{bmatrix} -7 & -8 \end{bmatrix}$$

The minimum element in the matrix B is then found by e=min(d), which results in the scalar $e = -8$.

Note:

This is a MATLAB built-in function. For more information about the min function, type help min.

Purpose:

Generate a matrix of all ones.

Syntax:

A=ones(m,n)

Description:

A=ones(m,n) results in an m-times-n matrix A with all ones.

Note:

This is a MATLAB built-in function. For more information about the ones function, type help ones.

Purpose:

Reduce the size of a square matrix by omitting rows and columns.

Syntax:

B=red(A,b)

Description:

B=red(A,b) reduces the square matrix A to a smaller matrix B by omitting rows and columns of A. The indices for rows and columns to be omitted are specified by the column vector b.

Examples:

Assume that the matrix A is defined as

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

and b as

$$b = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

The statement B=red(A,b) results in the matrix

$$B = \begin{bmatrix} 1 & 3 \\ 9 & 11 \end{bmatrix}$$

Purpose:

Matrix dimensions.

Syntax:

d=size(A)
[m,n]=size(A)

Description:

d=size(A) returns a vector with two integer components, d=[m,n], from the matrix A with dimensions m times n.

[m,n]=size(A) returns the dimensions m and n of the $m \times n$ matrix A.

Note:

This is a MATLAB built-in function. For more information about the size function, type help size.

Purpose:

Create sparse matrices.

Syntax:

S=sparse(A)
S=sparse(m,n)

Description:

S=sparse(A) converts a full matrix to sparse form by extracting all nonzero matrix elements. If S is already sparse, sparse(S) returns S.

S=sparse(m,n) generates an m-times-n sparse zero matrix.

Note:

This is a MATLAB built-in function. For more information about the sparse function, type help sparse.

Purpose:

Visualize matrix sparsity structure.

Syntax:

spy(S)

Description:

spy(S) plots the sparsity structure of any matrix S. S is usually a sparse matrix, but the function also accepts full matrices and the nonzero matrix elements are plotted.

Note:

This is a MATLAB built-in function. For more information about the spy function, type help spy.

Purpose:

Square root.

Syntax:

B=sqrt(A)

Description:

B=sqrt(A) computes the square root of the elements in matrix A and stores the result in matrix B.

Note:

This is a MATLAB built-in function. For more information about the sqrt function, type help sqrt.

Purpose:

Sum of the elements of a matrix.

Syntax:

b=sum(A)

Description:

For a vector a, the statement b=sum(a) results in a scalar a containing the sum of all elements of a.

For a matrix A, the statement b=sum(A) returns a row vector b containing the sum of the elements found in each column vector of A.

The sum of all elements of a matrix is determined by c=sum(sum(A)).

Note:

This is a MATLAB built-in function. For more information about the sum function, type help sum.

Purpose:

Generate a zero matrix.

Syntax:

A=zeros(m,n)

Description:

A=zeros(m,n) results in an m-times-n matrix A of zeros.

Note:

This is a MATLAB built-in function. For more information about the zeros function, type help zeros.

4   Material functions

The group of material functions comprises functions for constitutive models. The available models can treat linear elastic and isotropic hardening von Mises material. These material models are defined by the functions:

Material property functions	
hooke	Form linear elastic constitutive matrix
mises	Compute stresses and plastic strains for isotropic hardening von Mises material
dmises	Form elasto-plastic continuum matrix for isotropic hardening von Mises material

Purpose:

Compute material matrix for a linear elastic and isotropic material.

Syntax:

D = hooke(ptype,E,v)

Description:

hooke computes the material matrix D for a linear elastic and isotropic material.

The variable ptype is used to define the type of analysis.

$$
\text{ptype} = \begin{cases} 1 & \text{plane stress.} \\ 2 & \text{plane strain.} \\ 3 & \text{axisymmetry.} \\ 4 & \text{three dimensional analysis.} \end{cases}
$$

The material parameters E and v define the modulus of elasticity E and the Poisson's ratio ν, respectively.

For plane stress, ptype=1, D is formed as

$$
\mathbf{D} = \frac{E}{1 - \nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \dfrac{1 - \nu}{2} \end{bmatrix}
$$

For plane strain, ptype=2 and axisymmetry, ptype=3, D is formed as

$$
\mathbf{D} = \frac{E}{(1 + \nu)(1 - 2\nu)} \begin{bmatrix} 1 - \nu & \nu & \nu & 0 \\ \nu & 1 - \nu & \nu & 0 \\ \nu & \nu & 1 - \nu & 0 \\ 0 & 0 & 0 & \frac{1}{2}(1 - 2\nu) \end{bmatrix}
$$

For the three dimensional case, ptype=4, D is formed as

$$
\mathbf{D} = \frac{E}{(1 + \nu)(1 - 2\nu)} \begin{bmatrix} 1 - \nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1 - \nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1 - \nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2}(1 - 2\nu) & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2}(1 - 2\nu) & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2}(1 - 2\nu) \end{bmatrix}
$$

Purpose:

Compute stresses and plastic strains for an elasto-plastic isotropic hardening von Mises material.

Syntax:

[es,deps,st]=mises(ptype,mp,est,st)

Description:

mises computes updated stresses es, plastic strain increments deps, and state variables st for an elasto-plastic isotropic hardening von Mises material.

The input variable ptype is used to define the type of analysis, cf. hooke. The vector mp contains the material constants

$$\mathsf{mp} = [\; E \;\nu\; h \;]$$

where E is the modulus of elasticity, ν is the Poisson's ratio, and h is the plastic modulus. The input matrix est contains trial stresses obtained by using the elastic material matrix D in plants or some similar s-function, and the input vector st contains the state parameters

$$\mathsf{st} = [\; yi \;\sigma_y\; \epsilon_{eff}^p \;]$$

at the beginning of the step. The scalar yi states whether the material behaviour is elasto-plastic ($yi{=}1$), or elastic ($yi{=}0$). The current yield stress is denoted by σ_y and the effective plastic strain by ϵ_{eff}^p.

The output variables es and st contain updated values of es and st obtained by integration of the constitutive equations over the actual displacement step. The increments of the plastic strains are stored in the vector deps.

If es and st contain more than one row, then every row will be treated by the command.

Note:

It is not necessary to check whether the material behaviour is elastic or elasto-plastic, this test is done by the function. The computation is based on an Euler-Backward method, i.e. the radial return method.

Only the cases ptype=2, 3 and 4, are implemented.

Purpose:

Form the elasto-plastic continuum matrix for an isotropic hardening von Mises material.

Syntax:

D=dmises(ptype,mp,es,st)

Description:

dmises forms the elasto-plastic continuum matrix for an isotropic hardening von Mises material.

The input variable ptype is used to define the type of analysis, cf. hooke. The vector mp contains the material constants

$$\mathsf{mp} = [\, E \; \nu \; h \,]$$

where E is the modulus of elasticity, ν is the Poisson's ratio, and h is the plastic modulus. The matrix es contains current stresses obtained from plants or some similar s-function, and the vector st contains the current state parameters

$$\mathsf{st} = [\, yi \; \sigma_y \; \epsilon^p_{eff} \,]$$

where $yi=1$ if the material behaviour is elasto-plastic, and $yi=0$ if the material behaviour is elastic. The current yield stress is denoted by σ_y, and the current effective plastic strain by ϵ^p_{eff}.

Note:

Only the case ptype=2 is implemented.

5 Element functions

5.1 Introduction

The group of element functions contains functions for computation of element matrices and element forces for different element types. The element functions have been divided into the following groups

Spring element
Bar elements
Heat flow elements
Solid elements
Beam elements
Plate element

For each element type there is a function for computation of the element stiffness matrix $\mathbf{K}^e$. For most of the elements, an element load vector $\mathbf{f}^e$ can also be computed. These functions are identified by their last letter -e.

Using the function assem, the element stiffness matrices and element load vectors are assembled into a global stiffness matrix $\mathbf{K}$ and a load vector $\mathbf{f}$. Unknown nodal values of temperatures or displacements $\mathbf{a}$ are computed by solving the system of equations $\mathbf{Ka} = \mathbf{f}$ using the function solveq. A vector of nodal values of temperatures or displacements for a specific element is formed by the function extract.

When the element nodal values have been computed, the element flux or element stresses can be calculated using functions specific to the element type concerned. These functions are identified by their last letter -s.

For some elements, a function for computing the internal force vector is also available. These functions are identified by their last letter -f.

5.2 Spring element

The spring element, shown below, can be used for the analysis of one-dimensional spring systems and for a variety of analogous physical problems.
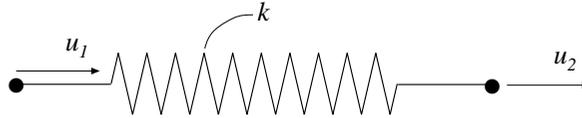
Quantities corresponding to the variables of the spring are listed in Table 1.

Problem type	Spring stiffness	Nodal displacement	Element force	Spring force
Spring	k	u	P	N
Bar	$\dfrac{EA}{L}$	u	P	N
Thermal conduction	$\dfrac{\lambda A}{L}$	T	$\bar{H}$	H
Electrical circuit	$\dfrac{1}{R}$	U	$\bar{I}$	I
Groundwater flow	$\dfrac{kA}{L}$	ϕ	$\bar{H}$	H
Pipe network	$\dfrac{\pi D^4}{128\mu L}$	p	$\bar{H}$	H

Table 1: *Analogous quantities*

| Interpretations of the spring element		
Problem type	Quantities	Designations
Spring	u_1, P_1 — k — u_2, P_2; N — N	k spring stiffness u displacement P element force N spring force
Bar	u_1, P_1 — u_2, P_2; E, A; L; N — N	L length E modulus of elasticity A area of cross section u displacement P element force N normal force
Thermal conduction	$\bar{H}_1$ T_1 — H — T_2 $\bar{H}_2$; λ; L	L length λ thermal conductivity T temperature $\bar{H}$ element heat flow H internal heat flow
Electrical circuit	U_1 — R — U_2; $\bar{I}_1$ $\bar{I}_2$; I	R resistance U potential $\bar{I}$ element current I internal current
Ground-water flow	$\bar{H}_1$ ϕ_1 — H — ϕ_2 $\bar{H}_2$; k; L	L length k permeability ϕ piezometric head $\bar{H}$ element water flow H internal water flow
Pipe network (laminar flow)	p_1 — D, μ — p_2; H; $\bar{H}_1$ $\bar{H}_2$; L	L length D pipe diameter μ viscosity p pressure $\bar{H}$ element fluid flow H internal fluid flow

Table 2: *Quantities used in different types of problems*

The following functions are available for the spring element:

Spring functions	
spring1e	Compute element matrix
spring1s	Compute spring force

Purpose:

Compute element stiffness matrix for a spring element.
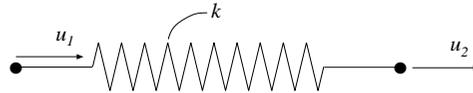
Syntax:

Ke=spring1e(ep)

Description:

spring1e provides the element stiffness matrix Ke for a spring element.

The input variable

$$\mathsf{ep} = [\, k \,]$$

supplies the spring stiffness k or the analog quantity defined in Table 1.

Theory:

The element stiffness matrix $\mathbf{K}^e$, stored in Ke, is computed according to

$$\mathbf{K}^e = \begin{bmatrix} k & -k \\ -k & k \end{bmatrix}$$

where k is defined by ep.

Purpose:

Compute spring force in a spring element.

Syntax:

es=spring1s(ep,ed)

Description:

spring1s computes the spring force es in a spring element.

The input variable ep is defined in spring1e and the element nodal displacements ed are obtained by the function extract.

The output variable

$$\textsf{es} = [\ N\ ]$$

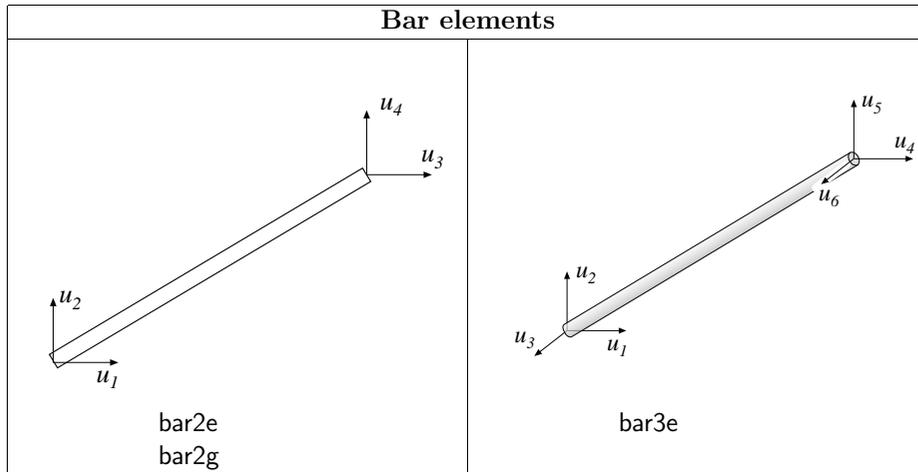contains the spring force N, or the analog quantity.

Theory:

The spring force N, or analog quantity, is computed according to

$$N = k\,[\ u_2 - u_1\ ]$$

5.3 Bar elements

Bar elements are available for one, two, and three dimensional analysis. For the one dimensional element, see the spring element.
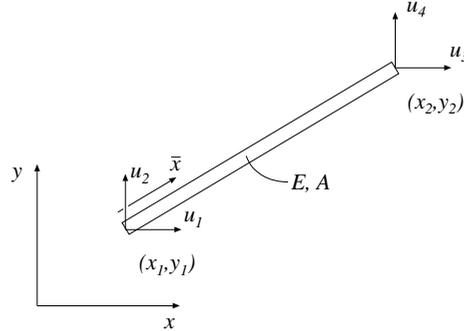

Bar elements

bar2e
bar2g

bar3e

Two dimensional bar functions	
bar2e	Compute element matrix
bar2g	Compute element matrix for geometric nonlinear element
bar2s	Compute normal force

Three dimensional bar functions	
bar3e	Compute element matrix
bar3s	Compute normal force

ELEMENT

Purpose:

Compute element stiffness matrix for a two dimensional bar element.

Syntax:

Ke=bar2e(ex,ey,ep)

Description:

bar2e provides the global element stiffness matrix Ke for a two dimensional bar element.

The input variables

$$\mathsf{ex} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \qquad \mathsf{ep} = \begin{bmatrix} E & A \end{bmatrix}$$
$$\mathsf{ey} = \begin{bmatrix} y_1 & y_2 \end{bmatrix}$$

supply the element nodal coordinates x_1, y_1, x_2, and y_2, the modulus of elasticity E, and the cross section area A.

Theory:

The element stiffness matrix $\mathbf{K}^e$, stored in Ke, is computed according to

$$\mathbf{K}^e = \mathbf{G}^T \, \bar{\mathbf{K}}^e \, \mathbf{G}$$

where

$$\bar{\mathbf{K}}^e = \frac{EA}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \qquad \mathbf{G} = \begin{bmatrix} n_{x\bar{x}} & n_{y\bar{x}} & 0 & 0 \\ 0 & 0 & n_{x\bar{x}} & n_{y\bar{x}} \end{bmatrix}$$

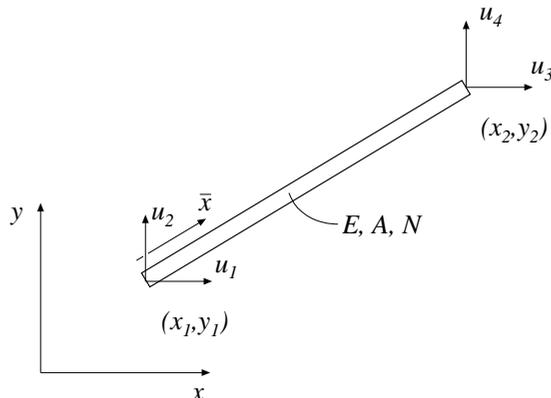The transformation matrix $\mathbf{G}$ contains the direction cosines

$$n_{x\bar{x}} = \frac{x_2 - x_1}{L} \qquad n_{y\bar{x}} = \frac{y_2 - y_1}{L}$$

where the length

$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Purpose:

Compute element stiffness matrix for a two dimensional geometric nonlinear bar.

Syntax:

Ke=bar2g(ex,ey,ep,N)

Description:

bar2g provides the element stiffness matrix Ke for a two dimensional geometric non-linear bar element.

The input variables ex, ey and ep are described in bar2e. The input variable

$$\mathsf{N} = [\, N \,]$$

contains the value of the normal force, which is positive in tension.

Theory:

The global element stiffness matrix $\mathbf{K}^e$, stored in Ke, is computed according to

$$\mathbf{K}^e = \mathbf{G}^T \, \bar{\mathbf{K}}^e \, \mathbf{G}$$

where

$$\bar{\mathbf{K}}^e = \frac{EA}{L} \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \frac{N}{L} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{G} = \begin{bmatrix} n_{x\bar{x}} & n_{y\bar{x}} & 0 & 0 \\ n_{x\bar{y}} & n_{y\bar{y}} & 0 & 0 \\ 0 & 0 & n_{x\bar{x}} & n_{y\bar{x}} \\ 0 & 0 & n_{x\bar{y}} & n_{y\bar{y}} \end{bmatrix}$$

The transformation matrix $\mathbf{G}$ contains the direction cosines
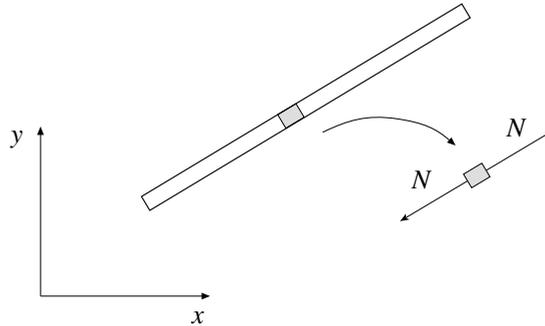
$$n_{x\bar{x}} = n_{y\bar{y}} = \frac{x_2 - x_1}{L} \qquad n_{y\bar{x}} = -n_{x\bar{y}} = \frac{y_2 - y_1}{L}$$

where the length

$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Purpose:

Compute normal force in a two dimensional bar element.

Syntax:

es=bar2s(ex,ey,ep,ed)

Description:

bar2s computes the normal force in the two dimensional bar elements bar2e and bar2g.

The input variables ex, ey, and ep are defined in bar2e and the element nodal displacements, stored in ed, are obtained by the function extract.

The output variable

$$\mathsf{es} = [\, N \,]$$

contains the normal force N.

Theory:

The normal force N is computed from

$$N = \frac{EA}{L} [\, -1 \quad 1 \,] \, \mathbf{G} \, \mathbf{a}^e$$
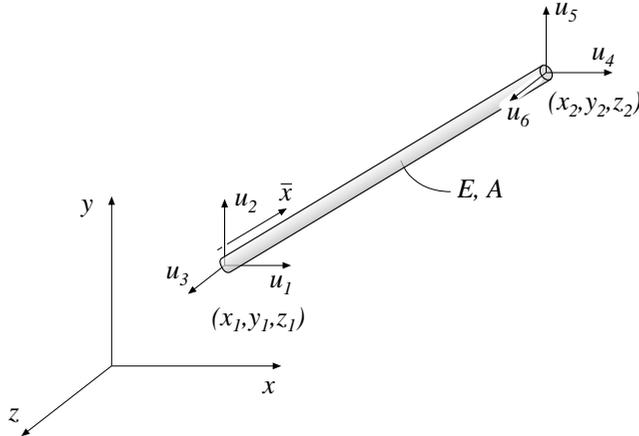
where E, A, L, and the transformation matrix $\mathbf{G}$ are defined in bar2e. The nodal displacements in global coordinates

$$\mathbf{a}^e = [\, u_1 \ \ u_2 \ \ u_3 \ \ u_4 \,]^T$$

are also shown in bar2e. Note that the transpose of $\mathbf{a}^e$ is stored in ed.

Purpose:

Compute element stiffness matrix for a three dimensional bar element.

Syntax:

Ke=bar3e(ex,ey,ez,ep)

Description:

bar3e provides the element stiffness matrix Ke for a three dimensional bar element.

The input variables

$$\text{ex} = [\, x_1 \;\; x_2 \,]$$
$$\text{ey} = [\, y_1 \;\; y_2 \,] \qquad \text{ep} = [\, E \;\; A \,]$$
$$\text{ez} = [\, z_1 \;\; z_2 \,]$$

supply the element nodal coordinates x_1, y_1, z_1, x_2 etc, the modulus of elasticity E, and the cross section area A.

Theory:

The global element stiffness matrix $\mathbf{K}^e$ is computed according to

$$\mathbf{K}^e = \mathbf{G}^T \, \bar{\mathbf{K}}^e \, \mathbf{G}$$

where

$$\bar{\mathbf{K}}^e = \frac{EA}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \qquad \mathbf{G} = \begin{bmatrix} n_{x\bar{x}} & n_{y\bar{x}} & n_{z\bar{x}} & 0 & 0 & 0 \\ 0 & 0 & 0 & n_{x\bar{x}} & n_{y\bar{x}} & n_{z\bar{x}} \end{bmatrix}$$
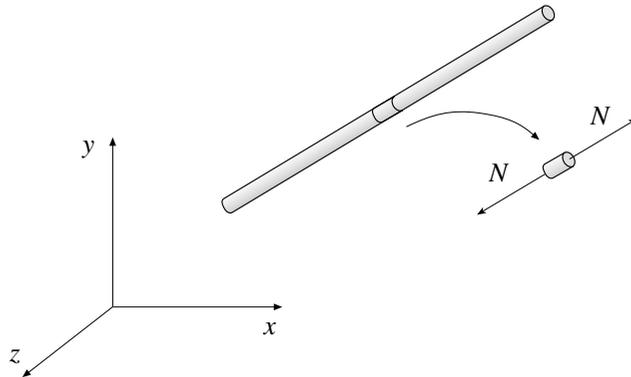
The transformation matrix $\mathbf{G}$ contains the direction cosines

$$n_{x\bar{x}} = \frac{x_2 - x_1}{L} \qquad n_{y\bar{x}} = \frac{y_2 - y_1}{L} \qquad n_{z\bar{x}} = \frac{z_2 - z_1}{L}$$

where the length $L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$.

Purpose:

Compute normal force in a three dimensional bar element.

Syntax:

es=bar3s(ex,ey,ez,ep,ed)

Description:

bar3s computes the normal force in a three dimensional bar element.

The input variables ex, ey, ez, and ep are defined in bar3e, and the element nodal displacements, stored in ed, are obtained by the function extract.

The output variable

$$\mathsf{es} = [\ N\ ]$$

contains the normal force N of the bar.

Theory:

The normal force N is computed from

$$N = \frac{EA}{L} [\ -1 \quad 1\ ] \mathbf{G}\, \mathbf{a}^e$$

where E, A, L, and the transformation matrix $\mathbf{G}$ are defined in bar3e. The nodal displacements in global coordinates

$$\mathbf{a}^e = [\ u_1 \ \ u_2 \ \ u_3 \ \ u_4 \ \ u_5 \ \ u_6\ ]^T$$

are also shown in bar3e. Note that the transpose of $\mathbf{a}^e$ is stored in ed.

5.4 Heat flow elements

Heat flow elements are available for one, two, and three dimensional analysis. For one dimensional heat flow the spring element spring1 is used.

A variety of important physical phenomena are described by the same differential equation as the heat flow problem. The heat flow element is thus applicable in modelling different physical applications. Table 3 below shows the relation between the primary variable **a**, the constitutive matrix **D**, and the load vector $\mathbf{f}_l$ for a chosen set of two dimensional physical problems.

Problem type	**a**	**D**	$\mathbf{f}_l$	Designation
Heat flow	T	λ_x , λ_y	Q	T = temperature λ_x , λ_y = thermal conductivity Q = heat supply
Groundwater flow	ϕ	k_x , k_y,	Q	ϕ = piezometric head k_x, k_y = permeabilities Q = fluid supply
St. Venant torsion	ϕ	$\dfrac{1}{G_{zy}}$, $\dfrac{1}{G_{zx}}$	2Θ	ϕ = stress function G_{zy}, G_{zx} = shear moduli Θ = angle of torsion per unit length

Table 3: *Problem dependent parameters*

Heat flow elements

flw2te

flw2qe
flw2i4e

flw2i8e

flw3i8e
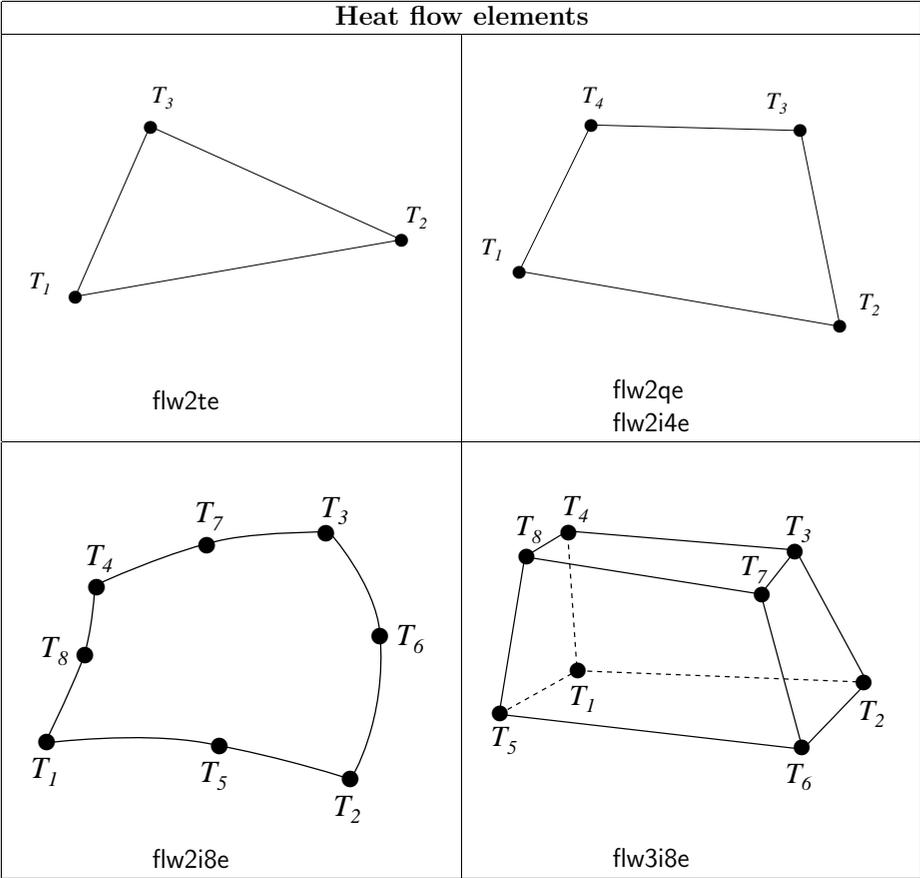
| 2D heat flow functions	
flw2te	Compute element matrices for a triangular element
flw2ts	Compute temperature gradients and flux
flw2qe	Compute element matrices for a quadrilateral element
flw2qs	Compute temperature gradients and flux
flw2i4e	Compute element matrices, 4 node isoparametric element
flw2i4s	Compute temperature gradients and flux
flw2i8e	Compute element matrices, 8 node isoparametric element
flw2i8s	Compute temperature gradients and flux

3D heat flow functions	
flw3i8e	Compute element matrices, 8 node isoparametric element
flw3i8s	Compute temperature gradients and flux

Purpose:

Compute element stiffness matrix for a triangular heat flow element.
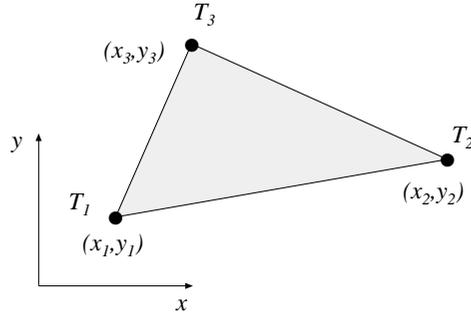
Syntax:

Ke=flw2te(ex,ey,ep,D)
[Ke,fe]=flw2te(ex,ey,ep,D,eq)

Description:

flw2te provides the element stiffness (conductivity) matrix Ke and the element load vector fe for a triangular heat flow element.

The element nodal coordinates x_1, y_1, x_2 etc, are supplied to the function by ex and ey, the element thickness t is supplied by ep and the thermal conductivities (or corresponding quantities) k_{xx}, k_{xy} etc are supplied by D.

$$\mathsf{ex} = \left[\begin{array}{ccc} x_1 & x_2 & x_3 \end{array}\right]$$
$$\mathsf{ey} = \left[\begin{array}{ccc} y_1 & y_2 & y_3 \end{array}\right] \qquad \mathsf{ep} = \left[\begin{array}{c} t \end{array}\right] \qquad \mathsf{D} = \left[\begin{array}{cc} k_{xx} & k_{xy} \\ k_{yx} & k_{yy} \end{array}\right]$$

If the scalar variable eq is given in the function, the element load vector fe is computed, using

$$\mathsf{eq} = \left[\begin{array}{c} Q \end{array}\right]$$

where Q is the heat supply per unit volume.

Theory:

The element stiffness matrix $\mathbf{K}^e$ and the element load vector $\mathbf{f}_l^e$, stored in Ke and fe, respectively, are computed according to

$$\mathbf{K}^e = (\mathbf{C}^{-1})^T \int_A \bar{\mathbf{B}}^T \, \mathbf{D} \, \bar{\mathbf{B}} \, t \, dA \ \mathbf{C}^{-1}$$
$$\mathbf{f}_l^e = (\mathbf{C}^{-1})^T \int_A \bar{\mathbf{N}}^T \, Q \, t \, dA$$

with the constitutive matrix $\mathbf{D}$ defined by D.

The evaluation of the integrals for the triangular element is based on the linear temperature approximation $T(x, y)$ and is expressed in terms of the nodal variables T_1, T_2 and T_3 as

$$T(x, y) = \mathbf{N}^e \mathbf{a}^e = \bar{\mathbf{N}} \, \mathbf{C}^{-1} \mathbf{a}^e$$

where

$$\bar{\mathbf{N}} = [\, 1 \ \ x \ \ y \,] \qquad \mathbf{C} = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix} \qquad \mathbf{a}^e = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix}$$

and hence it follows that

$$\bar{\mathbf{B}} = \nabla \bar{\mathbf{N}} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \nabla = \begin{bmatrix} \dfrac{\partial}{\partial x} \\[2mm] \dfrac{\partial}{\partial y} \end{bmatrix}$$

Evaluation of the integrals for the triangular element yields

$$\mathbf{K}^e = (\mathbf{C}^{-1})^T \, \bar{\mathbf{B}}^T \, \mathbf{D} \, \bar{\mathbf{B}} \, \mathbf{C}^{-1} \, t \, A$$

$$\mathbf{f}_l^e = \frac{Q A t}{3} [\, 1 \ \ 1 \ \ 1 \,]^T$$

where the element area A is determined as

$$A = \frac{1}{2} \det \mathbf{C}$$

Purpose:

Compute heat flux and temperature gradients in a triangular heat flow element.

Syntax:

[es,et]=flw2ts(ex,ey,D,ed)

Description:

flw2ts computes the heat flux vector es and the temperature gradient et (or corresponding quantities) in a triangular heat flow element.

The input variables ex, ey and the matrix D are defined in flw2te. The vector ed contains the nodal temperatures $\mathbf{a}^e$ of the element and is obtained by the function extract as

$$\mathsf{ed} = (\mathbf{a}^e)^T = [\, T_1 \ T_2 \ T_3 \,]$$

The output variables

$$\mathsf{es} = \mathbf{q}^T = [\, q_x \ q_y \,]$$

$$\mathsf{et} = (\nabla T)^T = \left[\ \frac{\partial T}{\partial x} \ \ \frac{\partial T}{\partial y} \ \right]$$

contain the components of the heat flux and the temperature gradient computed in the directions of the coordinate axis.

Theory:

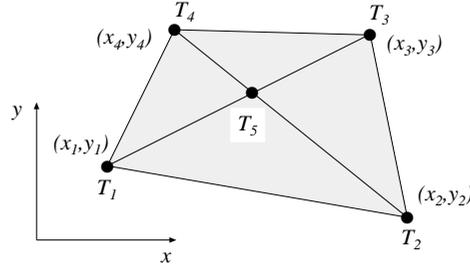The temperature gradient and the heat flux are computed according to

$$\nabla T = \bar{\mathbf{B}} \ \mathbf{C}^{-1} \ \mathbf{a}^e$$

$$\mathbf{q} = -\mathbf{D}\nabla T$$

where the matrices $\mathbf{D}$, $\bar{\mathbf{B}}$, and $\mathbf{C}$ are described in flw2te. Note that both the temperature gradient and the heat flux are constant in the element.

Purpose:

Compute element stiffness matrix for a quadrilateral heat flow element.

Syntax:

Ke=flw2qe(ex,ey,ep,D)
[Ke,fe]=flw2qe(ex,ey,ep,D,eq)

Description:

flw2qe provides the element stiffness (conductivity) matrix Ke and the element load vector fe for a quadrilateral heat flow element.

The element nodal coordinates x_1, y_1, x_2 etc, are supplied to the function by ex and ey, the element thickness t is supplied by ep and the thermal conductivities (or corresponding quantities) k_{xx}, k_{xy} etc are supplied by D.

$$\mathsf{ex} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}$$
$$\mathsf{ey} = \begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix} \qquad \mathsf{ep} = \begin{bmatrix} t \end{bmatrix} \qquad \mathsf{D} = \begin{bmatrix} k_{xx} & k_{xy} \\ k_{yx} & k_{yy} \end{bmatrix}$$

If the scalar variable eq is given in the function, the element load vector fe is computed, using

$$\mathsf{eq} = \begin{bmatrix} Q \end{bmatrix}$$

where Q is the heat supply per unit volume.

Theory:

In computing the element matrices, a fifth degree of freedom is introduced. The location of this extra degree of freedom is defined by the mean value of the coordinates in the corner points. Four sets of element matrices are calculated using flw2te. These matrices are then assembled and the fifth degree of freedom is eliminated by static condensation.

Purpose:

Compute heat flux and temperature gradients in a quadrilateral heat flow element.

Syntax:

[es,et]=flw2qs(ex,ey,ep,D,ed)
[es,et]=flw2qs(ex,ey,ep,D,ed,eq)

Description:

flw2qs computes the heat flux vector es and the temperature gradient et (or corresponding quantities) in a quadrilateral heat flow element.

The input variables ex, ey, eq and the matrix D are defined in flw2qe. The vector ed contains the nodal temperatures $\mathbf{a}^e$ of the element and is obtained by the function extract as

$$\mathsf{ed} = (\mathbf{a}^e)^T = [\ T_1 \ \ T_2 \ \ T_3 \ \ T_4\ ]$$

The output variables

$$\mathsf{es} = \mathbf{q}^T = [\ q_x \ q_y\ ]$$

$$\mathsf{et} = (\nabla T)^T = \left[\ \frac{\partial T}{\partial x}\ \ \frac{\partial T}{\partial y}\ \right]$$

contain the components of the heat flux and the temperature gradient computed in the directions of the coordinate axis.

Theory:

By assembling four triangular elements as described in flw2te a system of equations containing 5 degrees of freedom is obtained. From this system of equations the unknown temperature at the center of the element is computed. Then according to the description in flw2ts the temperature gradient and the heat flux in each of the four triangular elements are produced. Finally the temperature gradient and the heat flux of the quadrilateral element are computed as area weighted mean values from the values of the four triangular elements. If heat is supplied to the element, the element load vector eq is needed for the calculations.

Purpose:

Compute element stiffness matrix for a 4 node isoparametric heat flow element.
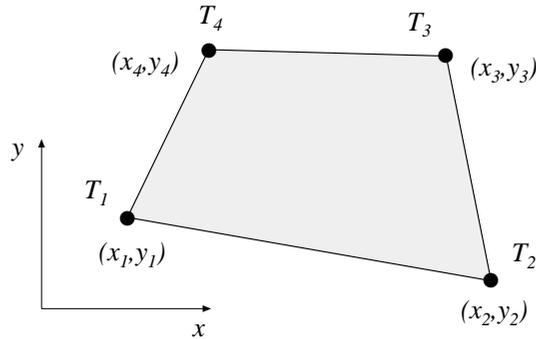
Syntax:

Ke=flw2i4e(ex,ey,ep,D)
[Ke,fe]=flw2i4e(ex,ey,ep,D,eq)

Description:

flw2i4e provides the element stiffness (conductivity) matrix Ke and the element load vector fe for a 4 node isoparametric heat flow element.

The element nodal coordinates x_1, y_1, x_2 etc, are supplied to the function by ex and ey. The element thickness t and the number of Gauss points n

$$(n \times n) \text{ integration points, } n = 1, 2, 3$$

are supplied to the function by ep and the thermal conductivities (or corresponding quantities) k_{xx}, k_{xy} etc are supplied by D.

$$\begin{aligned}
\mathsf{ex} &= \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} \\
\mathsf{ey} &= \begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix}
\end{aligned} \qquad \mathsf{ep} = \begin{bmatrix} t & n \end{bmatrix} \qquad \mathsf{D} = \begin{bmatrix} k_{xx} & k_{xy} \\ k_{yx} & k_{yy} \end{bmatrix}$$

If the scalar variable eq is given in the function, the element load vector fe is computed, using

$$\mathsf{eq} = \begin{bmatrix} Q \end{bmatrix}$$

where Q is the heat supply per unit volume.

Theory:

The element stiffness matrix $\mathbf{K}^e$ and the element load vector $\mathbf{f}_l^e$, stored in Ke and fe, respectively, are computed according to

$$\mathbf{K}^e = \int_A \mathbf{B}^{eT} \, \mathbf{D} \, \mathbf{B}^e \, t \, dA$$

$$\mathbf{f}_l^e = \int_A \mathbf{N}^{eT} \, Q \, t \, dA$$

with the constitutive matrix $\mathbf{D}$ defined by D.

The evaluation of the integrals for the isoparametric 4 node element is based on a temperature approximation $T(\xi, \eta)$, expressed in a local coordinates system in terms of the nodal variables T_1, T_2, T_3 and T_4 as

$$T(\xi, \eta) = \mathbf{N}^e \mathbf{a}^e$$

where

$$\mathbf{N}^e = [\, N_1^e \;\; N_2^e \;\; N_3^e \;\; N_4^e \,] \qquad \mathbf{a}^e = [\, T_1 \;\; T_2 \;\; T_3 \;\; T_4 \,]^T$$

The element shape functions are given by

$$N_1^e = \frac{1}{4}(1 - \xi)(1 - \eta) \qquad N_2^e = \frac{1}{4}(1 + \xi)(1 - \eta)$$

$$N_3^e = \frac{1}{4}(1 + \xi)(1 + \eta) \qquad N_4^e = \frac{1}{4}(1 - \xi)(1 + \eta)$$

The $\mathbf{B}^e$-matrix is given by

$$\mathbf{B}^e = \nabla \mathbf{N}^e = \begin{bmatrix} \dfrac{\partial}{\partial x} \\[2mm] \dfrac{\partial}{\partial y} \end{bmatrix} \mathbf{N}^e = (\mathbf{J}^T)^{-1} \begin{bmatrix} \dfrac{\partial}{\partial \xi} \\[2mm] \dfrac{\partial}{\partial \eta} \end{bmatrix} \mathbf{N}^e$$

where $\mathbf{J}$ is the Jacobian matrix

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} \\[2mm] \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} \end{bmatrix}$$

Evaluation of the integrals is done by Gauss integration.

Purpose:

Compute heat flux and temperature gradients in a 4 node isoparametric heat flow element.

Syntax:

[es,et,eci]=flw2i4s(ex,ey,ep,D,ed)

Description:

flw2i4s computes the heat flux vector es and the temperature gradient et (or corresponding quantities) in a 4 node isoparametric heat flow element.

The input variables ex, ey, ep and the matrix D are defined in flw2i4e. The vector ed contains the nodal temperatures $\mathbf{a}^e$ of the element and is obtained by extract as

$$\mathsf{ed} = (\mathbf{a}^e)^T = [\, T_1 \ \ T_2 \ \ T_3 \ \ T_4 \,]$$

The output variables

$$\mathsf{es} = \bar{\mathbf{q}}^T \quad = \begin{bmatrix} q_x^1 & q_y^1 \\ q_x^2 & q_y^2 \\ \vdots & \vdots \\ q_x^{n^2} & q_y^{n^2} \end{bmatrix}$$

$$\mathsf{et} = (\bar{\nabla}T)^T = \begin{bmatrix} \dfrac{\partial T^1}{\partial x} & \dfrac{\partial T^1}{\partial y} \\[2mm] \dfrac{\partial T^2}{\partial x} & \dfrac{\partial T^2}{\partial y} \\[2mm] \vdots & \vdots \\[2mm] \dfrac{\partial T^{n^2}}{\partial x} & \dfrac{\partial T^{n^2}}{\partial y} \end{bmatrix} \qquad \mathsf{eci} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_{n^2} & y_{n^2} \end{bmatrix}$$

contain the heat flux, the temperature gradient, and the coordinates of the integration points. The index n denotes the number of integration points used within the element, cf. flw2i4e.

Theory:

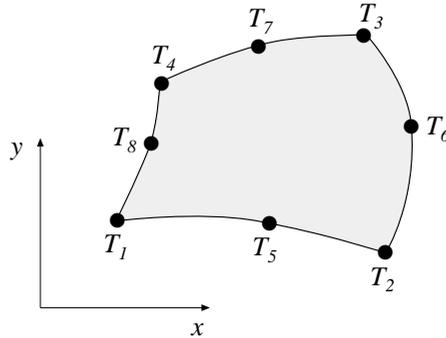The temperature gradient and the heat flux are computed according to

$$\nabla T = \mathbf{B}^e \, \mathbf{a}^e$$

$$\mathbf{q} = -\mathbf{D}\nabla T$$

where the matrices $\mathbf{D}$, $\mathbf{B}^e$, and $\mathbf{a}^e$ are described in flw2i4e, and where the integration points are chosen as evaluation points.

Purpose:

Compute element stiffness matrix for an 8 node isoparametric heat flow element.

Syntax:

Ke=flw2i8e(ex,ey,ep,D)
[Ke,fe]=flw2i8e(ex,ey,ep,D,eq)

Description:

flw2i8e provides the element stiffness (conductivity) matrix Ke and the element load vector fe for an 8 node isoparametric heat flow element.

The element nodal coordinates x_1, y_1, x_2 etc, are supplied to the function by ex and ey. The element thickness t and the number of Gauss points n

$(n \times n)$ integration points, $n = 1, 2, 3$

are supplied to the function by ep and the thermal conductivities (or corresponding quantities) k_{xx}, k_{xy} etc are supplied by D.

$$\text{ex} = [\, x_1 \ x_2 \ x_3 \ \ldots \ x_8 \,]$$
$$\text{ey} = [\, y_1 \ y_2 \ y_3 \ \ldots \ y_8 \,] \qquad \text{ep} = [\, t \ n \,] \qquad \text{D} = \begin{bmatrix} k_{xx} & k_{xy} \\ k_{yx} & k_{yy} \end{bmatrix}$$

If the scalar variable eq is given in the function, the vector fe is computed, using

$$\text{eq} = [\, Q \,]$$

where Q is the heat supply per unit volume.

Theory:

The element stiffness matrix $\mathbf{K}^e$ and the element load vector $\mathbf{f}_l^e$, stored in Ke and fe, respectively, are computed according to

$$\mathbf{K}^e = \int_A \mathbf{B}^{eT} \, \mathbf{D} \, \mathbf{B}^e \, t \, dA$$

$$\mathbf{f}_l^e = \int_A \mathbf{N}^{eT} \, Q \, t \, dA$$

with the constitutive matrix $\mathbf{D}$ defined by D.

The evaluation of the integrals for the 2D isoparametric 8 node element is based on a temperature approximation $T(\xi, \eta)$, expressed in a local coordinates system in terms of the nodal variables T_1 to T_8 as

$$T(\xi, \eta) = \mathbf{N}^e \mathbf{a}^e$$

where

$$\mathbf{N}^e = [\, N_1^e \ N_2^e \ N_3^e \ \ldots \ N_8^e \,] \qquad \mathbf{a}^e = [\, T_1 \ T_2 \ T_3 \ \ldots \ T_8 \,]^T$$

The element shape functions are given by

$$N_1^e = -\frac{1}{4}(1 - \xi)(1 - \eta)(1 + \xi + \eta) \qquad N_5^e = \frac{1}{2}(1 - \xi^2)(1 - \eta)$$

$$N_2^e = -\frac{1}{4}(1 + \xi)(1 - \eta)(1 - \xi + \eta) \qquad N_6^e = \frac{1}{2}(1 + \xi)(1 - \eta^2)$$

$$N_3^e = -\frac{1}{4}(1 + \xi)(1 + \eta)(1 - \xi - \eta) \qquad N_7^e = \frac{1}{2}(1 - \xi^2)(1 + \eta)$$

$$N_4^e = -\frac{1}{4}(1 - \xi)(1 + \eta)(1 + \xi - \eta) \qquad N_8^e = \frac{1}{2}(1 - \xi)(1 - \eta^2)$$

The $\mathbf{B}^e$-matrix is given by

$$\mathbf{B}^e = \nabla \mathbf{N}^e = \begin{bmatrix} \dfrac{\partial}{\partial x} \\ \dfrac{\partial}{\partial y} \end{bmatrix} \mathbf{N}^e = (\mathbf{J}^T)^{-1} \begin{bmatrix} \dfrac{\partial}{\partial \xi} \\ \dfrac{\partial}{\partial \eta} \end{bmatrix} \mathbf{N}^e$$

where $\mathbf{J}$ is the Jacobian matrix

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} \end{bmatrix}$$

Evaluation of the integrals is done by Gauss integration.

Purpose:

Compute heat flux and temperature gradients in an 8 node isoparametric heat flow element.

Syntax:

[es,et,eci]=flw2i8s(ex,ey,ep,D,ed)

Description:

flw2i8s computes the heat flux vector es and the temperature gradient et (or corresponding quantities) in an 8 node isoparametric heat flow element.

The input variables ex, ey, ep and the matrix D are defined in flw2i8e. The vector ed contains the nodal temperatures $\mathbf{a}^e$ of the element and is obtained by the function extract as

$$\mathsf{ed} = (\mathbf{a}^e)^T = [\, T_1 \ \ T_2 \ \ T_3 \ \ \dots \ \ T_8 \,]$$

The output variables

$$\mathsf{es} = \bar{\mathbf{q}}^T \quad = \begin{bmatrix} q_x^1 & q_y^1 \\ q_x^2 & q_y^2 \\ \vdots & \vdots \\ q_x^{n^2} & q_y^{n^2} \end{bmatrix}$$

$$\mathsf{et} = (\bar{\nabla} T)^T = \begin{bmatrix} \dfrac{\partial T^1}{\partial x} & \dfrac{\partial T^1}{\partial y} \\[2mm] \dfrac{\partial T^2}{\partial x} & \dfrac{\partial T^2}{\partial y} \\[1mm] \vdots & \vdots \\[1mm] \dfrac{\partial T^{n^2}}{\partial x} & \dfrac{\partial T^{n^2}}{\partial y} \end{bmatrix} \qquad \mathsf{eci} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_{n^2} & y_{n^2} \end{bmatrix}$$

contain the heat flux, the temperature gradient, and the coordinates of the integration points. The index n denotes the number of integration points used within the element, cf. flw2i8e.

Theory:

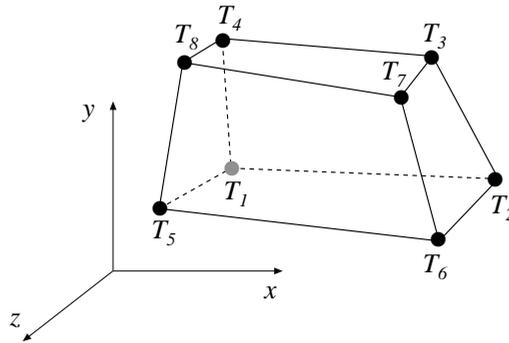The temperature gradient and the heat flux are computed according to

$$\nabla T = \mathbf{B}^e \, \mathbf{a}^e$$

$$\mathbf{q} = -\mathbf{D} \nabla T$$

where the matrices $\mathbf{D}$, $\mathbf{B}^e$, and $\mathbf{a}^e$ are described in flw2i8e, and where the integration points are chosen as evaluation points.

Purpose:

Compute element stiffness matrix for an 8 node isoparametric element.

Syntax:

Ke=flw3i8e(ex,ey,ez,ep,D)
[Ke,fe]=flw3i8e(ex,ey,ez,ep,D,eq)

Description:

flw3i8e provides the element stiffness (conductivity) matrix Ke and the element load vector fe for an 8 node isoparametric heat flow element.

The element nodal coordinates x_1, y_1, z_1 x_2 etc, are supplied to the function by ex, ey and ez. The number of Gauss points n

$$(n \times n \times n) \text{ integration points, } n = 1, 2, 3$$

are supplied to the function by ep and the thermal conductivities (or corresponding quantities) k_{xx}, k_{xy} etc are supplied by D.

$$\begin{aligned} \mathsf{ex} &= \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_8 \end{bmatrix} \\ \mathsf{ey} &= \begin{bmatrix} y_1 & y_2 & y_3 & \dots & y_8 \end{bmatrix} \\ \mathsf{ez} &= \begin{bmatrix} z_1 & z_2 & z_3 & \dots & z_8 \end{bmatrix} \end{aligned} \qquad \mathsf{ep} = \begin{bmatrix} n \end{bmatrix} \qquad \mathsf{D} = \begin{bmatrix} k_{xx} & k_{xy} & k_{xz} \\ k_{yx} & k_{yy} & k_{yz} \\ k_{zx} & k_{zy} & k_{zz} \end{bmatrix}$$

If the scalar variable eq is given in the function, the element load vector fe is computed, using

$$\mathsf{eq} = \begin{bmatrix} Q \end{bmatrix}$$

where Q is the heat supply per unit volume.

ELEMENT   5.4 − 14

Theory:

The element stiffness matrix $\mathbf{K}^e$ and the element load vector $\mathbf{f}_l^e$, stored in Ke and fe, respectively, are computed according to

$$\mathbf{K}^e = \int_V \mathbf{B}^{eT}\, \mathbf{D}\, \mathbf{B}^e\, dV$$

$$\mathbf{f}_l^e = \int_V \mathbf{N}^{eT}\, Q\, dV$$

with the constitutive matrix $\mathbf{D}$ defined by D.

The evaluation of the integrals for the 3D isoparametric 8 node element is based on a temperature approximation $T(\xi, \eta, \zeta)$, expressed in a local coordinates system in terms of the nodal variables T_1 to T_8 as

$$T(\xi, \eta, \zeta) = \mathbf{N}^e \mathbf{a}^e$$

where

$$\mathbf{N}^e = [\, N_1^e \ \ N_2^e \ \ N_3^e \ \ \dots \ \ N_8^e \,] \qquad \mathbf{a}^e = [\, T_1 \ \ T_2 \ \ T_3 \ \ \dots \ \ T_8 \,]^T$$

The element shape functions are given by

$$N_1^e = \frac{1}{8}(1-\xi)(1-\eta)(1-\zeta) \qquad N_2^e = \frac{1}{8}(1+\xi)(1-\eta)(1-\zeta)$$

$$N_3^e = \frac{1}{8}(1+\xi)(1+\eta)(1-\zeta) \qquad N_4^e = \frac{1}{8}(1-\xi)(1+\eta)(1-\zeta)$$

$$N_5^e = \frac{1}{8}(1-\xi)(1-\eta)(1+\zeta) \qquad N_6^e = \frac{1}{8}(1+\xi)(1-\eta)(1+\zeta)$$

$$N_7^e = \frac{1}{8}(1+\xi)(1+\eta)(1+\zeta) \qquad N_8^e = \frac{1}{8}(1-\xi)(1+\eta)(1+\zeta)$$

The $\mathbf{B}^e$-matrix is given by

$$\mathbf{B}^e = \nabla \mathbf{N}^e = \begin{bmatrix} \dfrac{\partial}{\partial x} \\[2mm] \dfrac{\partial}{\partial y} \\[2mm] \dfrac{\partial}{\partial z} \end{bmatrix} \mathbf{N}^e = (\mathbf{J}^T)^{-1} \begin{bmatrix} \dfrac{\partial}{\partial \xi} \\[2mm] \dfrac{\partial}{\partial \eta} \\[2mm] \dfrac{\partial}{\partial \zeta} \end{bmatrix} \mathbf{N}^e$$

where $\mathbf{J}$ is the Jacobian matrix

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} & \dfrac{\partial x}{\partial \zeta} \\[2mm] \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} & \dfrac{\partial y}{\partial \zeta} \\[2mm] \dfrac{\partial z}{\partial \xi} & \dfrac{\partial z}{\partial \eta} & \dfrac{\partial z}{\partial \zeta} \end{bmatrix}$$

Evaluation of the integrals is done by Gauss integration.

Purpose:

Compute heat flux and temperature gradients in an 8 node isoparametric heat flow element.

Syntax:

[es,et,eci]=flw3i8s(ex,ey,ez,ep,D,ed)

Description:

flw3i8s computes the heat flux vector es and the temperature gradient et (or corresponding quantities) in an 8 node isoparametric heat flow element.

The input variables ex, ey, ez, ep and the matrix D are defined in flw3i8e. The vector ed contains the nodal temperatures $\mathbf{a}^e$ of the element and is obtained by the function extract as

$$\text{ed} = (\mathbf{a}^e)^T = [\; T_1 \;\; T_2 \;\; T_3 \;\; \ldots \;\; T_8 \;]$$

The output variables

$$\text{es} = \bar{\mathbf{q}}^T = \begin{bmatrix} q_x^1 & q_y^1 & q_z^1 \\ q_x^2 & q_y^2 & q_z^2 \\ \vdots & \vdots & \vdots \\ q_x^{n^3} & q_y^{n^3} & q_z^{n^3} \end{bmatrix}$$

$$\text{et} = (\bar{\nabla} T)^T = \begin{bmatrix} \dfrac{\partial T^1}{\partial x} & \dfrac{\partial T^1}{\partial y} & \dfrac{\partial T^1}{\partial z} \\[2mm] \dfrac{\partial T^2}{\partial x} & \dfrac{\partial T^2}{\partial y} & \dfrac{\partial T^2}{\partial z} \\[2mm] \vdots & \vdots & \vdots \\[2mm] \dfrac{\partial T^{n^3}}{\partial x} & \dfrac{\partial T^{n^3}}{\partial y} & \dfrac{\partial T^{n^3}}{\partial z} \end{bmatrix} \qquad \text{eci} = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_{n^3} & y_{n^3} & z_{n^3} \end{bmatrix}$$

contain the heat flux, the temperature gradient, and the coordinates of the integration points. The index n denotes the number of integration points used within the element, cf. flw3i8e.

Theory:

The temperature gradient and the heat flux are computed according to

$$\nabla T = \mathbf{B}^e \, \mathbf{a}^e$$

$$\mathbf{q} = -\mathbf{D}\nabla T$$

where the matrices $\mathbf{D}$, $\mathbf{B}^e$, and $\mathbf{a}^e$ are described in flw3i8e, and where the integration points are chosen as evaluation points.

5.5 Solid elements

Solid elements are available for two dimensional analysis in plane stress (panels) and plane strain, and for general three dimensional analysis. In the two dimensional case there are a triangular three node element, a quadrilateral four node element, two rectangular four node elements, and quadrilateral isoparametric four and eight node elements. For three dimensional analysis there is an eight node isoparametric element.

The elements are able to deal with both isotropic and anisotropic materials. The triangular element and the three isoparametric elements can also be used together with a nonlinear material model. The material properties are specified by supplying the constitutive matrix **D** as an input variable to the element functions. This matrix can be formed by the functions described in Section 4.

Solid elements
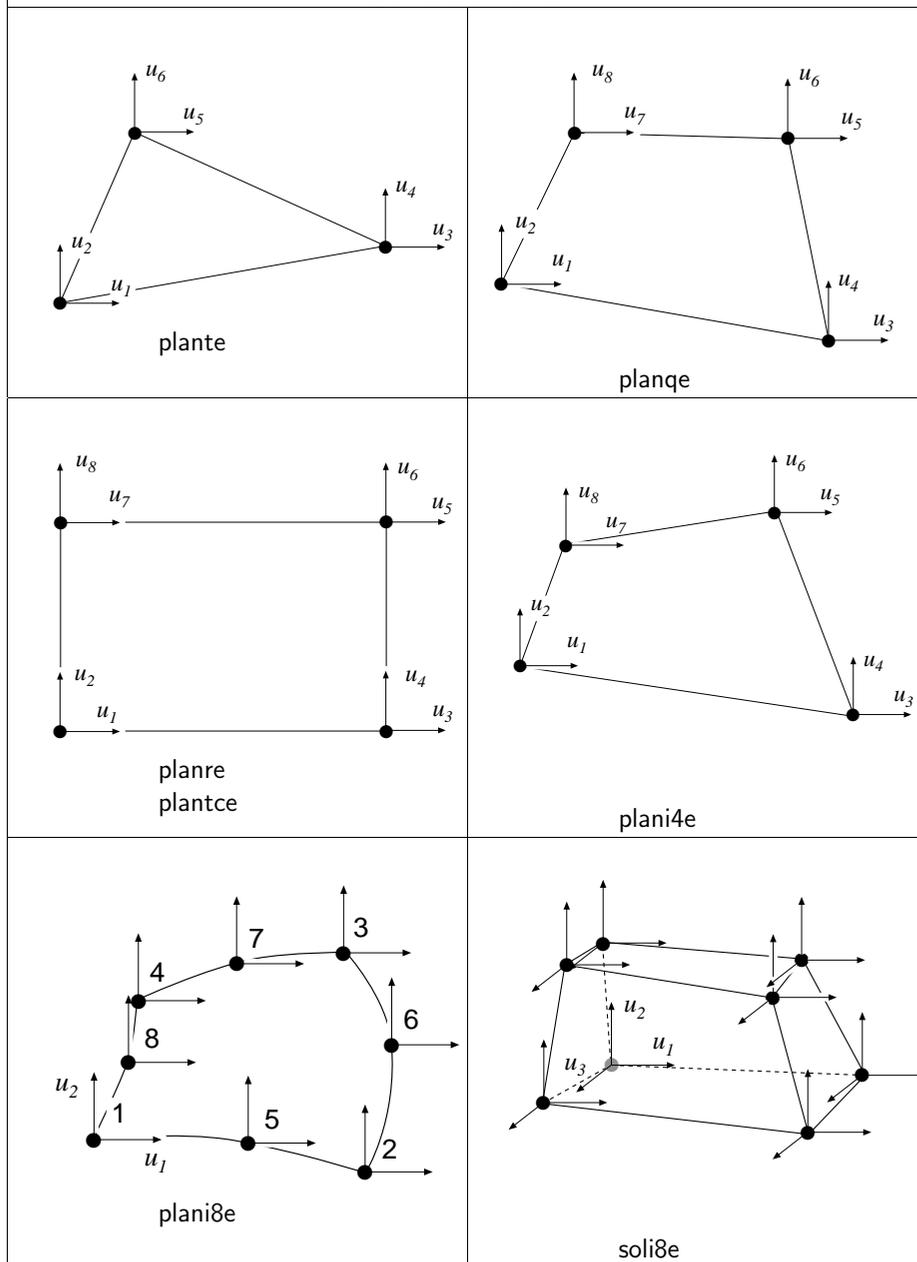
| 2D solid functions	
plante	Compute element matrices for a triangular element
plants	Compute stresses and strains
plantf	Compute internal element forces
planqe	Compute element matrices for a quadrilateral element
planqs	Compute stresses and strains
planre	Compute element matrices for a rectangular Melosh element
planrs	Compute stresses and strains
plantce	Compute element matrices for a rectangular Turner-Clough element
plantcs	Compute stresses and strains
plani4e	Compute element matrices, 4 node isoparametric element
plani4s	Compute stresses and strains
plani4f	Compute internal element forces
plani8e	Compute element matrices, 8 node isoparametric element
plani8s	Compute stresses and strains
plani8f	Compute internal element forces

3D solid functions	
soli8e	Compute element matrices, 8 node isoparametric element
soli8s	Compute stresses and strains
soli8f	Compute internal element forces

Purpose:

Compute element matrices for a triangular element in plane strain or plane stress.
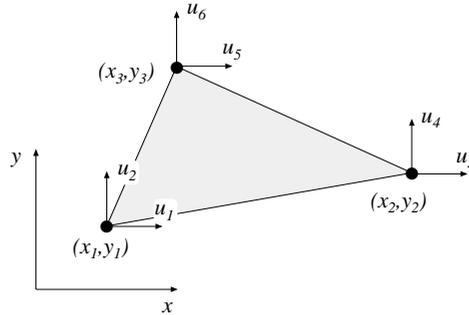
Syntax:

Ke=plante(ex,ey,ep,D)
[Ke,fe]=plante(ex,ey,ep,D,eq)

Description:

plante provides an element stiffness matrix Ke and an element load vector fe for a triangular element in plane strain or plane stress.

The element nodal coordinates x_1, y_1, x_2 etc. are supplied to the function by ex and ey. The type of analysis $ptype$ and the element thickness t are supplied by ep,

$$ptype = 1 \quad \text{plane stress}$$
$$ptype = 2 \quad \text{plane strain}$$

and the material properties are supplied by the constitutive matrix D. Any arbitrary D-matrix with dimensions from (3×3) to (6×6) may be given. For an isotropic elastic material the constitutive matrix can be formed by the function hooke, see Section 4.

$$\mathsf{ex} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}$$
$$\mathsf{ey} = \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} \qquad \mathsf{ep} = \begin{bmatrix} ptype & t \end{bmatrix}$$

$$\mathsf{D} = \begin{bmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{bmatrix} \quad or \quad \mathsf{D} = \begin{bmatrix} D_{11} & D_{12} & D_{13} & D_{14} & [D_{15}] & [D_{16}] \\ D_{21} & D_{22} & D_{23} & D_{24} & [D_{25}] & [D_{26}] \\ D_{31} & D_{32} & D_{33} & D_{34} & [D_{35}] & [D_{36}] \\ D_{41} & D_{42} & D_{43} & D_{44} & [D_{45}] & [D_{46}] \\ [D_{51}] & [D_{52}] & [D_{53}] & [D_{54}] & [D_{55}] & [D_{56}] \\ [D_{61}] & [D_{62}] & [D_{63}] & [D_{64}] & [D_{65}] & [D_{66}] \end{bmatrix}$$

If uniformly distributed loads are applied to the element, the element load vector fe is computed. The input variable

$$\mathsf{eq} = \begin{bmatrix} b_x \\ b_y \end{bmatrix}$$

containing loads per unit volume, b_x and b_y, is then given.

Theory:

The element stiffness matrix $\mathbf{K}^e$ and the element load vector $\mathbf{f}_l^e$, stored in Ke and fe, respectively, are computed according to

$$\mathbf{K}^e = (\mathbf{C}^{-1})^T \int_A \bar{\mathbf{B}}^T \, \mathbf{D} \, \bar{\mathbf{B}} \, t \, dA \, \mathbf{C}^{-1}$$

$$\mathbf{f}_l^e = (\mathbf{C}^{-1})^T \int_A \bar{\mathbf{N}}^T \, \mathbf{b} \, t \, dA$$

with the constitutive matrix $\mathbf{D}$ defined by D, and the body force vector $\mathbf{b}$ defined by eq.

The evaluation of the integrals for the triangular element is based on a linear displacement approximation $\mathbf{u}(x, y)$ and is expressed in terms of the nodal variables u_1, u_2, ... , u_6 as

$$\mathbf{u}(x, y) = \mathbf{N}^e \, \mathbf{a}^e = \bar{\mathbf{N}} \, \mathbf{C}^{-1} \, \mathbf{a}^e$$

where

$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} \qquad \bar{\mathbf{N}} = \begin{bmatrix} 1 & x & y & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x & y \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & x_1 & y_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_1 & y_1 \\ 1 & x_2 & y_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_2 & y_2 \\ 1 & x_3 & y_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_3 & y_3 \end{bmatrix} \qquad \mathbf{a}^e = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix}$$

The matrix $\bar{\mathbf{B}}$ is obtained as

$$\bar{\mathbf{B}} = \tilde{\nabla} \bar{\mathbf{N}} c \quad \text{where} \quad \tilde{\nabla} = \begin{bmatrix} \dfrac{\partial}{\partial x} & 0 \\ 0 & \dfrac{\partial}{\partial y} \\ \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} \end{bmatrix}$$

If a larger **D**-matrix than (3×3) is used for plane stress ($ptype = 1$), the **D**-matrix is reduced to a (3×3) matrix by static condensation using $\sigma_{zz} = \sigma_{xz} = \sigma_{yz} = 0$. These stress components are connected with the rows 3, 5 and 6 in the D-matrix respectively.

If a larger **D**-matrix than (3×3) is used for plane strain ($ptype = 2$), the **D**-matrix is reduced to a (3×3) matrix using $\varepsilon_{zz} = \gamma_{xz} = \gamma_{yz} = 0$. This implies that a (3×3) **D**-matrix is created by the rows and the columns 1, 2 and 4 from the original D-matrix.

Evaluation of the integrals for the triangular element yields

$$\mathbf{K}^e = (\mathbf{C}^{-1})^T \, \bar{\mathbf{B}}^T \, \mathbf{D} \, \bar{\mathbf{B}} \, \mathbf{C}^{-1} \, t \, A$$

$$\mathbf{f}_l^e = \frac{A\,t}{3} \, [\, b_x \ \ b_y \ \ b_x \ \ b_y \ \ b_x \ \ b_y \,]^T$$

where the element area A is determined as

$$A = \frac{1}{2} \det \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix}$$

Purpose:

Compute stresses and strains in a triangular element in plane strain or plane stress.
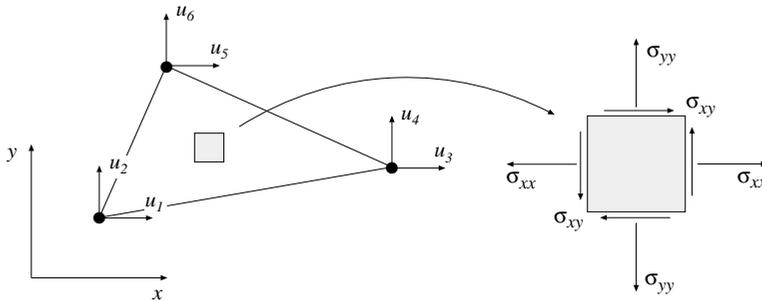
Syntax:

[es,et]=plants(ex,ey,ep,D,ed)

Description:

plants computes the stresses es and the strains et in a triangular element in plane strain or plane stress.

The input variables ex, ey, ep and D are defined in plante. The vector ed contains the nodal displacements $\mathbf{a}^e$ of the element and is obtained by the function extract as

$$\mathsf{ed} = (\mathbf{a}^e)^T = [\; u_1 \;\; u_2 \;\; \dots \;\; u_6 \;]$$

The output variables

$$\mathsf{es} = \boldsymbol{\sigma}^T = [\; \sigma_{xx} \; \sigma_{yy} \; [\sigma_{zz}] \; \sigma_{xy} \; [\sigma_{xz}] \; [\sigma_{yz}] \;]$$

$$\mathsf{et} = \boldsymbol{\varepsilon}^T = [\; \varepsilon_{xx} \; \varepsilon_{yy} \; [\varepsilon_{zz}] \; \gamma_{xy} \; [\gamma_{xz}] \; [\gamma_{yz}] \;]$$

contain the stress and strain components. The size of es and et follows the size of D. Note that for plane stress $\varepsilon_{zz} \neq 0$, and for plane strain $\sigma_{zz} \neq 0$.

Theory:

The strains and stresses are computed according to

$$\varepsilon = \bar{\mathbf{B}} \, \mathbf{C}^{-1} \, \mathbf{a}^e$$

$$\boldsymbol{\sigma} = \mathbf{D} \, \varepsilon$$

where the matrices $\mathbf{D}$, $\bar{\mathbf{B}}$, $\mathbf{C}$ and $\mathbf{a}^e$ are described in plante. Note that both the strains and the stresses are constant in the element.

Purpose:

Compute internal element force vector in a triangular element in plane strain or plane stress.

Syntax:

ef=plantf(ex,ey,ep,es)

Description:

plantf computes the internal element forces ef in a triangular element in plane strain or plane stress.

The input variables ex, ey and ep are defined in plante, and the input variable es is defined in plants.

The output variable

$$\mathsf{ef} = \mathbf{f}_i^{eT} = [\ f_{i1}\ f_{i2}\ \cdots\ f_{i6}\ ]$$

contains the components of the internal force vector.

Theory:

The internal force vector is computed according to

$$\mathbf{f}_i^e = (\mathbf{C}^{-1})^T \int_A \bar{\mathbf{B}}^T \boldsymbol{\sigma}\ t\ dA$$

where the matrices $\bar{\mathbf{B}}$ and $\mathbf{C}$ are defined in plante and $\boldsymbol{\sigma}$ is defined in plants.

Evaluation of the integral for the triangular element yields

$$\mathbf{f}_i^e = (\mathbf{C}^{-1})^T \bar{\mathbf{B}}^T\ \boldsymbol{\sigma}\ t\ A$$

Purpose:

Compute element matrices for a quadrilateral element in plane strain or plane stress.
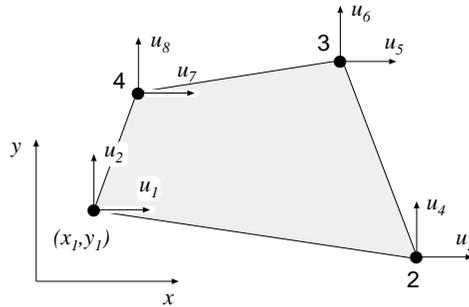
Syntax:

Ke=planqe(ex,ey,ep,D)
[Ke,fe]=planqe(ex,ey,ep,D,eq)

Description:

planqe provides an element stiffness matrix Ke and an element load vector fe for a quadrilateral element in plane strain or plane stress.

The element nodal coordinates x_1, y_1, x_2 etc. are supplied to the function by ex and ey. The type of analysis *ptype* and the element thickness t are supplied by ep,

$$ptype = 1 \quad \text{plane stress}$$
$$ptype = 2 \quad \text{plane strain}$$

and the material properties are supplied by the constitutive matrix D. Any arbitrary D-matrix with dimensions from (3×3) to (6×6) may be given. For an isotropic elastic material the constitutive matrix can be formed by the function hooke, see Section 4.

$$\text{ex} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}$$
$$\text{ey} = \begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix} \qquad \text{ep} = \begin{bmatrix} ptype & t \end{bmatrix}$$

$$\text{D} = \begin{bmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{bmatrix} \;\; or \;\; \text{D} = \begin{bmatrix} D_{11} & D_{12} & D_{13} & D_{14} & [D_{15}] & [D_{16}] \\ D_{21} & D_{22} & D_{23} & D_{24} & [D_{25}] & [D_{26}] \\ D_{31} & D_{32} & D_{33} & D_{34} & [D_{35}] & [D_{36}] \\ D_{41} & D_{42} & D_{43} & D_{44} & [D_{45}] & [D_{46}] \\ [D_{51}] & [D_{52}] & [D_{53}] & [D_{54}] & [D_{55}] & [D_{56}] \\ [D_{61}] & [D_{62}] & [D_{63}] & [D_{64}] & [D_{65}] & [D_{66}] \end{bmatrix}$$

If uniformly distributed loads are applied on the element, the element load vector fe is computed. The input variable

$$\mathsf{eq} = \left[\begin{array}{c} b_x \\ b_y \end{array} \right]$$

containing loads per unit volume, b_x and b_y, is then given.

Theory:

In computing the element matrices, two more degrees of freedom are introduced. The location of these two degrees of freedom is defined by the mean value of the coordinates at the corner points. Four sets of element matrices are calculated using plante. These matrices are then assembled and the two extra degrees of freedom are eliminated by static condensation.

Purpose:

Compute stresses and strains in a quadrilateral element in plane strain or plane stress.
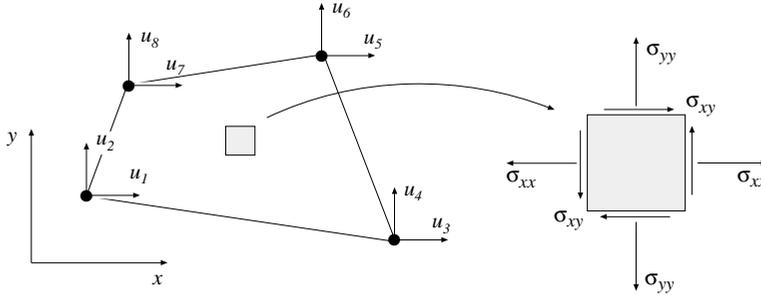
Syntax:

[es,et]=planqs(ex,ey,ep,D,ed)
[es,et]=planqs(ex,ey,ep,D,ed,eq)

Description:

planqs computes the stresses es and the strains et in a quadrilateral element in plane strain or plane stress.

The input variables ex, ey, ep, D and eq are defined in planqe. The vector ed contains the nodal displacements $\mathbf{a}^e$ of the element and is obtained by the function extract as

$$\text{ed} = (\mathbf{a}^e)^T = [\ u_1\ \ u_2\ \ \ldots\ \ u_8\ ]$$

If body forces are applied to the element the variable eq must be included.

The output variables

$$\text{es} = \boldsymbol{\sigma}^T = [\ \sigma_{xx}\ \sigma_{yy}\ [\sigma_{zz}]\ \sigma_{xy}\ [\sigma_{xz}]\ [\sigma_{yz}]\ ]$$

$$\text{et} = \boldsymbol{\varepsilon}^T = [\ \varepsilon_{xx}\ \varepsilon_{yy}\ [\varepsilon_{zz}]\ \gamma_{xy}\ [\gamma_{xz}]\ [\gamma_{yz}]\ ]$$

contain the stress and strain components. The size of es and et follows the size of D. Note that for plane stress $\varepsilon_{zz} \neq 0$, and for plane strain $\sigma_{zz} \neq 0$.

Theory:

By assembling triangular elements as described in planqe a system of equations containing 10 degrees of freedom is obtained. From this system of equations the two unknown displacements at the center of the element are computed. Then according to the description in plants the strain and stress components in each of the four triangular elements are produced. Finally the quadrilateral element strains and stresses are computed as area weighted mean values from the values of the four triangular elements. If uniformly distributed loads are applied on the element, the element load vector eq is needed for the calculations.

Purpose:

Compute element matrices for a rectangular (Melosh) element in plane strain or plane stress.
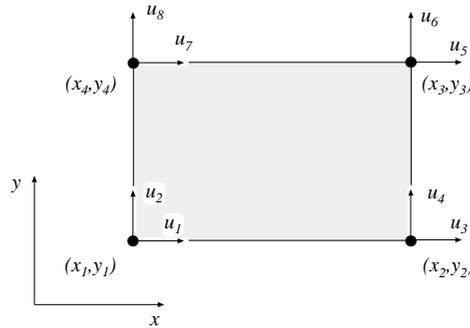
Syntax:

Ke=planre(ex,ey,ep,D)
[Ke,fe]=planre(ex,ey,ep,D,eq)

Description:

planre provides an element stiffness matrix Ke and an element load vector fe for a rectangular (Melosh) element in plane strain or plane stress. This element can only be used if the element edges are parallel to the coordinate axis.

The element nodal coordinates (x_1, y_1) and (x_3, y_3) are supplied to the function by ex and ey. The type of analysis $ptype$ and the element thickness t are supplied by ep,

$$ptype = 1 \quad \text{plane stress}$$
$$ptype = 2 \quad \text{plane strain}$$

and the material properties are supplied by the constitutive matrix D. Any arbitrary D-matrix with dimensions from (3×3) to (6×6) may be given. For an isotropic elastic material the constitutive matrix can be formed by the function hooke, see Section 4.

$$\mathsf{ex} = \begin{bmatrix} x_1 & x_3 \end{bmatrix}$$
$$\mathsf{ey} = \begin{bmatrix} y_1 & y_3 \end{bmatrix} \qquad \mathsf{ep} = \begin{bmatrix} ptype & t \end{bmatrix}$$

$$\mathsf{D} = \begin{bmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{bmatrix} \text{ or } \mathsf{D} = \begin{bmatrix} D_{11} & D_{12} & D_{13} & D_{14} & [D_{15}] & [D_{16}] \\ D_{21} & D_{22} & D_{23} & D_{24} & [D_{25}] & [D_{26}] \\ D_{31} & D_{32} & D_{33} & D_{34} & [D_{35}] & [D_{36}] \\ D_{41} & D_{42} & D_{43} & D_{44} & [D_{45}] & [D_{46}] \\ [D_{51}] & [D_{52}] & [D_{53}] & [D_{54}] & [D_{55}] & [D_{56}] \\ [D_{61}] & [D_{62}] & [D_{63}] & [D_{64}] & [D_{65}] & [D_{66}] \end{bmatrix}$$

If uniformly distributed loads are applied on the element, the element load vector fe is computed. The input variable

$$\mathsf{eq} = \begin{bmatrix} b_x \\ b_y \end{bmatrix}$$

containing loads per unit volume, b_x and b_y, is then given.

Theory:

The element stiffness matrix $\mathbf{K}^e$ and the element load vector $\mathbf{f}_l^e$, stored in Ke and fe, respectively, are computed according to

$$\mathbf{K}^e = \int_A \mathbf{B}^{eT} \mathbf{D} \mathbf{B}^e \, t \, dA$$

$$\mathbf{f}_l^e = \int_A \mathbf{N}^{eT} \mathbf{b} \, t \, dA$$

with the constitutive matrix $\mathbf{D}$ defined by D, and the body force vector $\mathbf{b}$ defined by eq.

The evaluation of the integrals for the rectangular element is based on a bilinear displacement approximation $\mathbf{u}(x, y)$ and is expressed in terms of the nodal variables u_1, u_2, ..., u_8 as

$$\mathbf{u}(x, y) = \mathbf{N}^e \, \mathbf{a}^e$$

where

$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} \quad \mathbf{N}^e = \begin{bmatrix} N_1^e & 0 & N_2^e & 0 & N_3^e & 0 & N_4^e & 0 \\ 0 & N_1^e & 0 & N_2^e & 0 & N_3^e & 0 & N_4^e \end{bmatrix} \quad \mathbf{a}^e = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_8 \end{bmatrix}$$

With a local coordinate system located at the center of the element, the element shape functions $N_1^e - N_4^e$ are obtained as

$$N_1^e = \frac{1}{4ab}(x - x_2)(y - y_4)$$

$$N_2^e = -\frac{1}{4ab}(x - x_1)(y - y_3)$$

$$N_3^e = \frac{1}{4ab}(x - x_4)(y - y_2)$$

$$N_4^e = -\frac{1}{4ab}(x - x_3)(y - y_1)$$

where

$$a = \frac{1}{2}(x_3 - x_1) \quad \text{and} \quad b = \frac{1}{2}(y_3 - y_1)$$

The matrix $\mathbf{B}^e$ is obtained as

$$\mathbf{B}^e = \tilde{\nabla}\mathbf{N}^e \qquad \text{where} \quad \tilde{\nabla} = \begin{bmatrix} \dfrac{\partial}{\partial x} & 0 \\[2mm] 0 & \dfrac{\partial}{\partial y} \\[2mm] \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} \end{bmatrix}$$

If a larger $\mathbf{D}$-matrix than (3×3) is used for plane stress ($ptype = 1$), the $\mathbf{D}$-matrix is reduced to a (3×3) matrix by static condensation using $\sigma_{zz} = \sigma_{xz} = \sigma_{yz} = 0$. These stress components are connected with the rows 3, 5 and 6 in the D-matrix respectively.

If a larger $\mathbf{D}$-matrix than (3×3) is used for plane strain ($ptype = 2$), the $\mathbf{D}$-matrix is reduced to a (3×3) matrix using $\varepsilon_{zz} = \gamma_{xz} = \gamma_{yz} = 0$. This implies that a (3×3) $\mathbf{D}$-matrix is created by the rows and the columns 1, 2 and 4 from the original D-matrix.

Evaluation of the integrals for the rectangular element can be done either analytically or numerically by use of a 2×2 point Gauss integration. The element load vector $\mathbf{f}_l^e$ yields

$$\mathbf{f}_l^e = abt \begin{bmatrix} b_x \\ b_y \\ b_x \\ b_y \\ b_x \\ b_y \\ b_x \\ b_y \end{bmatrix}$$

Purpose:

Compute stresses and strains in a rectangular (Melosh) element in plane strain or plane stress.
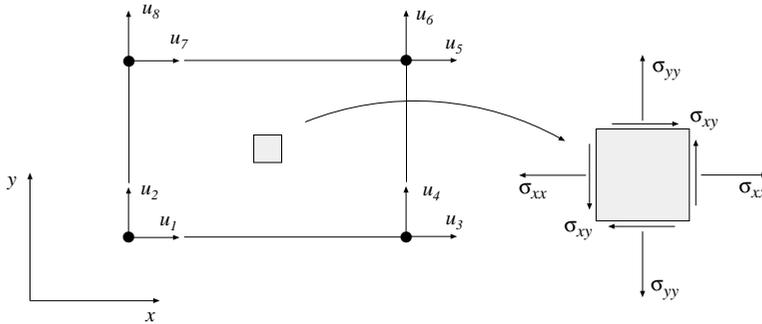
Syntax:

[es,et]=planrs(ex,ey,ep,D,ed)

Description:

planrs computes the stresses es and the strains et in a rectangular (Melosh) element in plane strain or plane stress. The stress and strain components are computed at the center of the element.

The input variables ex, ey, ep and D are defined in planre. The vector ed contains the nodal displacements $\mathbf{a}^e$ of the element and is obtained by the function extract as

$$\mathsf{ed} = (\mathbf{a}^e)^T = [\ u_1 \ \ u_2 \ \ \ldots \ \ u_8\ ]$$

The output variables

$$\mathsf{es} = \boldsymbol{\sigma}^T = [\ \sigma_{xx}\ \sigma_{yy}\ [\sigma_{zz}]\ \sigma_{xy}\ [\sigma_{xz}]\ [\sigma_{yz}]\ ]$$

$$\mathsf{et} = \boldsymbol{\varepsilon}^T = [\ \varepsilon_{xx}\ \varepsilon_{yy}\ [\varepsilon_{zz}]\ \gamma_{xy}\ [\gamma_{xz}]\ [\gamma_{yz}]\ ]$$

contain the stress and strain components. The size of es and et follows the size of D. Note that for plane stress $\varepsilon_{zz} \neq 0$, and for plane strain $\sigma_{zz} \neq 0$.

Theory:

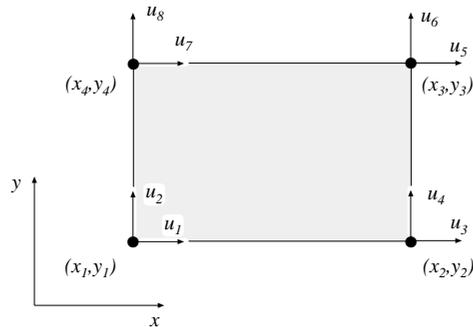The strains and stresses are computed according to

$$\boldsymbol{\varepsilon} = \mathbf{B}^e\,\mathbf{a}^e$$

$$\boldsymbol{\sigma} = \mathbf{D}\,\boldsymbol{\varepsilon}$$

where the matrices $\mathbf{D}$, $\mathbf{B}^e$, and $\mathbf{a}^e$ are described in planre, and where the evaluation point (x, y) is chosen to be at the center of the element.

Purpose:

Compute element matrices for a rectangular (Turner-Clough) element in plane strain or plane stress.

Syntax:

Ke=plantce(ex,ey,ep)
[Ke,fe]=plantce(ex,ey,ep,eq)

Description:

plantce provides an element stiffness matrix Ke and an element load vector fe for a rectangular (Turner-Clough) element in plane strain or plane stress. This element can only be used if the material is isotropic and if the element edges are parallel to the coordinate axis.

The element nodal coordinates (x_1, y_1) and (x_3, y_3) are supplied to the function by ex and ey. The state of stress $ptype$, the element thickness t and the material properties E and ν are supplied by ep. For plane stress $ptype = 1$ and for plane strain $ptype = 2$.

$$\mathsf{ex} = \begin{bmatrix} x_1 & x_3 \end{bmatrix} \qquad \mathsf{ep} = \begin{bmatrix} ptype & t & E & \nu \end{bmatrix}$$
$$\mathsf{ey} = \begin{bmatrix} y_1 & y_3 \end{bmatrix}$$

If uniformly distributed loads are applied to the element, the element load vector fe is computed. The input variable

$$\mathsf{eq} = \begin{bmatrix} b_x \\ b_y \end{bmatrix}$$

containing loads per unit volume, b_x and b_y, is then given.

Theory:

The element stiffness matrix $\mathbf{K}^e$ and the element load vector $\mathbf{f}_l^e$, stored in Ke and fe, respectively, are computed according to

$$\mathbf{K}^e = \int_A \mathbf{B}^{eT}\, \mathbf{D}\, \mathbf{B}^e\, t\, dA$$

$$\mathbf{f}_l^e = \int_A \mathbf{N}^{eT}\, \mathbf{b}\, t\, dA$$

where the constitutive matrix $\mathbf{D}$ is described in hooke, see Section 4, and the body force vector $\mathbf{b}$ is defined by eq.

The evaluation of the integrals for the Turner-Clough element is based on a displacement field $\mathbf{u}(x,y)$ built up of a bilinear displacement approximation superposed by bubble functions in order to create a linear stress field over the element. The displacement field is expressed in terms of the nodal variables u_1, u_2, ..., u_8 as

$$\mathbf{u}(x,y) = \mathbf{N}^e\, \mathbf{a}^e$$

where

$$\mathbf{u} = \left[\begin{array}{c} u_x \\ u_y \end{array}\right] \quad \mathbf{N}^e = \left[\begin{array}{cccccccc} N_1^e & N_5^e & N_2^e & N_5^e & N_3^e & N_5^e & N_4^e & N_5^e \\ N_6^e & N_1^e & N_6^e & N_2^e & N_6^e & N_3^e & N_6^e & N_4^e \end{array}\right] \quad \mathbf{a}^e = \left[\begin{array}{c} u_1 \\ u_2 \\ \vdots \\ u_8 \end{array}\right]$$

With a local coordinate system located at the center of the element, the element shape functions $N_1^e - N_6^e$ are obtained as

$$N_1^e = \frac{1}{4ab}(a-x)(b-y)$$

$$N_2^e = \frac{1}{4ab}(a+x)(b-y)$$

$$N_3^e = \frac{1}{4ab}(a+x)(b+y)$$

$$N_4^e = \frac{1}{4ab}(a-x)(b+y)$$

$$N_5^e = \frac{1}{8ab}\left[(b^2-y^2)+\nu(a^2-x^2)\right]$$

$$N_6^e = \frac{1}{8ab}\left[(a^2-x^2)+\nu(b^2-y^2)\right]$$

where

$$a = \frac{1}{2}(x_3-x_1) \quad \text{and} \quad b = \frac{1}{2}(y_3-y_1)$$

The matrix $\mathbf{B}^e$ is obtained as

$$\mathbf{B}^e = \tilde{\nabla}\mathbf{N}^e \qquad \text{where} \quad \tilde{\nabla} = \begin{bmatrix} \dfrac{\partial}{\partial x} & 0 \\[2mm] 0 & \dfrac{\partial}{\partial y} \\[2mm] \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} \end{bmatrix}$$

Evaluation of the integrals for the Turner-Clough element can be done either analytically or numerically by use of a 2×2 point Gauss integration. The element load vector $\mathbf{f}_l^e$ yields

$$\mathbf{f}_l^e = abt \begin{bmatrix} b_x \\ b_y \\ b_x \\ b_y \\ b_x \\ b_y \\ b_x \\ b_y \end{bmatrix}$$

Purpose:

Compute stresses and strains in a Turner-Clough element in plane strain or plane stress.
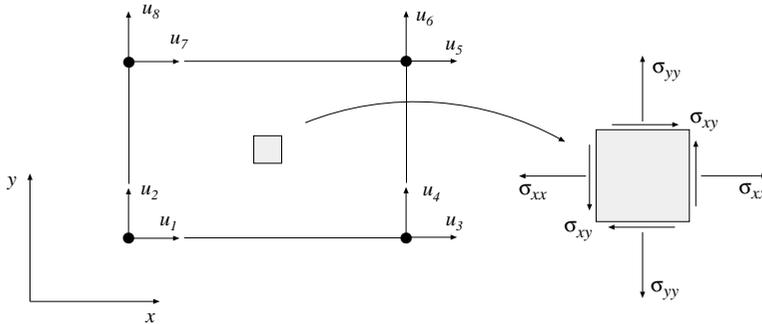
Syntax:

[es,et]=plantcs(ex,ey,ep,ed)

Description:

plantcs computes the stresses es and the strains et in a rectangular Turner-Clough element in plane strain or plane stress. The stress and strain components are computed at the center of the element.

The input variables ex, ey, and ep are defined in plantce. The vector ed contains the nodal displacements $\mathbf{a}^e$ of the element and is obtained by the function extract as

$$\mathsf{ed} = (\mathbf{a}^e)^T = [\, u_1 \;\; u_2 \;\; \ldots \;\; u_8 \,]$$

The output variables

$$\mathsf{es} = \boldsymbol{\sigma}^T = [\, \sigma_{xx} \; \sigma_{yy} \; [\sigma_{zz}] \; \sigma_{xy} \; [\sigma_{xz}] \; [\sigma_{yz}] \,]$$

$$\mathsf{et} = \boldsymbol{\varepsilon}^T = [\, \varepsilon_{xx} \; \varepsilon_{yy} \; [\varepsilon_{zz}] \; \gamma_{xy} \; [\gamma_{xz}] \; [\gamma_{yz}] \,]$$

contain the stress and strain components. The size of es and et follows the size of D. Note that for plane stress $\varepsilon_{zz} \neq 0$, and for plane strain $\sigma_{zz} \neq 0$.

Theory:

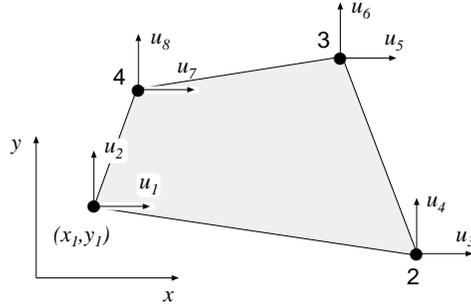The strains and stresses are computed according to

$$\varepsilon = \mathbf{B}^e \, \mathbf{a}^e$$

$$\boldsymbol{\sigma} = \mathbf{D} \, \varepsilon$$

where the matrices $\mathbf{D}$, $\mathbf{B}^e$, and $\mathbf{a}^e$ are described in plantce, and where the evaluation point (x, y) is chosen to be at the center of the element.

Purpose:

Compute element matrices for a 4 node isoparametric element in plane strain or plane stress.

Syntax:

Ke=plani4e(ex,ey,ep,D)
[Ke,fe]=plani4e(ex,ey,ep,D,eq)

Description:

plani4e provides an element stiffness matrix Ke and an element load vector fe for a 4 node isoparametric element in plane strain or plane stress.

The element nodal coordinates x_1, y_1, x_2 etc. are supplied to the function by ex and ey. The type of analysis $ptype$, the element thickness t, and the number of Gauss points n are supplied by ep.

$$
\begin{array}{lll}
ptype = 1 & \text{plane stress} & (n \times n) \text{ integration points} \\
ptype = 2 & \text{plane strain} & n = 1, 2, 3
\end{array}
$$

The material properties are supplied by the constitutive matrix D. Any arbitrary D-matrix with dimensions from (3×3) to (6×6) maybe given. For an isotropic elastic material the constitutive matrix can be formed by the function hooke, see Section 4.

$$
\begin{aligned}
\text{ex} &= \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} \\
\text{ey} &= \begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix}
\end{aligned}
\qquad
\text{ep} = \begin{bmatrix} ptype & t & n \end{bmatrix}
$$

$$
\mathsf{D} = \begin{bmatrix}
D_{11} & D_{12} & D_{13} \\
D_{21} & D_{22} & D_{23} \\
D_{31} & D_{32} & D_{33}
\end{bmatrix}
\text{ or }
\mathsf{D} = \begin{bmatrix}
D_{11} & D_{12} & D_{13} & D_{14} & [D_{15}] & [D_{16}] \\
D_{21} & D_{22} & D_{23} & D_{24} & [D_{25}] & [D_{26}] \\
D_{31} & D_{32} & D_{33} & D_{34} & [D_{35}] & [D_{36}] \\
D_{41} & D_{42} & D_{43} & D_{44} & [D_{45}] & [D_{46}] \\
[D_{51}] & [D_{52}] & [D_{53}] & [D_{54}] & [D_{55}] & [D_{56}] \\
[D_{61}] & [D_{62}] & [D_{63}] & [D_{64}] & [D_{65}] & [D_{66}]
\end{bmatrix}
$$

If different D_i -matrices are used in the Gauss points these D_i -matrices are stored in a global vector D. For numbering of the Gauss points, see eci in plani4s.

$$D = \begin{bmatrix} D_1 \\ D_2 \\ \vdots \\ D_{n^2} \end{bmatrix}$$

If uniformly distributed loads are applied to the element, the element load vector fe is computed. The input variable

$$eq = \begin{bmatrix} b_x \\ b_y \end{bmatrix}$$

containing loads per unit volume, b_x and b_y, is then given.

Theory:

The element stiffness matrix $\mathbf{K}^e$ and the element load vector $\mathbf{f}_l^e$, stored in Ke and fe, respectively, are computed according to

$$\mathbf{K}^e = \int_A \mathbf{B}^{eT} \, \mathbf{D} \, \mathbf{B}^e \, t \, dA$$

$$\mathbf{f}_l^e = \int_A \mathbf{N}^{eT} \, \mathbf{b} \, t \, dA$$

with the constitutive matrix $\mathbf{D}$ defined by D, and the body force vector $\mathbf{b}$ defined by eq.

The evaluation of the integrals for the isoparametric 4 node element is based on a displacement approximation $\mathbf{u}(\xi, \eta)$, expressed in a local coordinates system in terms of the nodal variables u_1, u_2, ..., u_8 as

$$\mathbf{u}(\xi, \eta) = \mathbf{N}^e \, \mathbf{a}^e$$

where

$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} \quad \mathbf{N}^e = \begin{bmatrix} N_1^e & 0 & N_2^e & 0 & N_3^e & 0 & N_4^e & 0 \\ 0 & N_1^e & 0 & N_2^e & 0 & N_3^e & 0 & N_4^e \end{bmatrix} \quad \mathbf{a}^e = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_8 \end{bmatrix}$$

The element shape functions are given by

$$N_1^e = \frac{1}{4}(1 - \xi)(1 - \eta) \qquad N_2^e = \frac{1}{4}(1 + \xi)(1 - \eta)$$

$$N_3^e = \frac{1}{4}(1 + \xi)(1 + \eta) \qquad N_4^e = \frac{1}{4}(1 - \xi)(1 + \eta)$$

The matrix $\mathbf{B}^e$ is obtained as

$$\mathbf{B}^e = \tilde{\nabla}\mathbf{N}^e \qquad \text{where} \quad \tilde{\nabla} = \begin{bmatrix} \dfrac{\partial}{\partial x} & 0 \\[2mm] 0 & \dfrac{\partial}{\partial y} \\[2mm] \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} \end{bmatrix}$$

and where

$$\begin{bmatrix} \dfrac{\partial}{\partial x} \\[2mm] \dfrac{\partial}{\partial y} \end{bmatrix} = (\mathbf{J}^T)^{-1} \begin{bmatrix} \dfrac{\partial}{\partial \xi} \\[2mm] \dfrac{\partial}{\partial \eta} \end{bmatrix} \qquad\qquad \mathbf{J} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} \\[2mm] \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} \end{bmatrix}$$

If a larger $\mathbf{D}$-matrix than (3×3) is used for plane stress ($ptype = 1$), the $\mathbf{D}$-matrix is reduced to a (3×3) matrix by static condensation using $\sigma_{zz} = \sigma_{xz} = \sigma_{yz} = 0$. These stress components are connected with the rows 3, 5 and 6 in the D-matrix respectively.

If a larger $\mathbf{D}$-matrix than (3×3) is used for plane strain ($ptype = 2$), the $\mathbf{D}$-matrix is reduced to a (3×3) matrix using $\varepsilon_{zz} = \gamma_{xz} = \gamma_{yz} = 0$. This implies that a (3×3) $\mathbf{D}$-matrix is created by the rows and the columns 1, 2 and 4 from the original D-matrix.

Evaluation of the integrals is done by Gauss integration.

Purpose:

Compute stresses and strains in a 4 node isoparametric element in plane strain or plane stress.
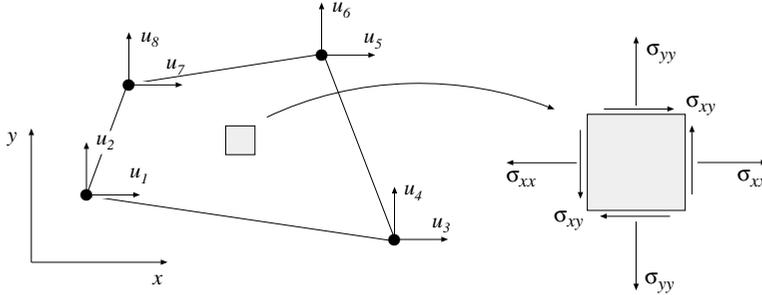
Syntax:

[es,et,eci]=plani4s(ex,ey,ep,D,ed)

Description:

plani4s computes stresses es and the strains et in a 4 node isoparametric element in plane strain or plane stress.

The input variables ex, ey, ep and the matrix D are defined in plani4e. The vector ed contains the nodal displacements $\mathbf{a}^e$ of the element and is obtained by the function extract as

$$\mathsf{ed} = (\mathbf{a}^e)^T = [\, u_1 \;\; u_2 \;\; \ldots \;\; u_8 \,]$$

The output variables

$$\mathsf{es} = \boldsymbol{\sigma}^T = \begin{bmatrix} \sigma^1_{xx} & \sigma^1_{yy} & [\sigma^1_{zz}] & \sigma^1_{xy} & [\sigma^1_{xz}] & [\sigma^1_{yz}] \\ \sigma^2_{xx} & \sigma^2_{yy} & [\sigma^2_{zz}] & \sigma^2_{xy} & [\sigma^2_{xz}] & [\sigma^2_{yz}] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \sigma^{n^2}_{xx} & \sigma^{n^2}_{yy} & [\sigma^{n^2}_{zz}] & \sigma^{n^2}_{xy} & [\sigma^{n^2}_{xz}] & [\sigma^{n^2}_{yz}] \end{bmatrix}$$

$$\mathsf{et} = \boldsymbol{\varepsilon}^T = \begin{bmatrix} \varepsilon^1_{xx} & \varepsilon^1_{yy} & [\varepsilon^1_{zz}] & \gamma^1_{xy} & [\gamma^1_{xz}] & [\gamma^1_{yz}] \\ \varepsilon^2_{xx} & \varepsilon^2_{yy} & [\varepsilon^2_{zz}] & \gamma^2_{xy} & [\gamma^2_{xz}] & [\gamma^2_{yz}] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \varepsilon^{n^2}_{xx} & \varepsilon^{n^2}_{yy} & [\varepsilon^{n^2}_{zz}] & \gamma^{n^2}_{xy} & [\gamma^{n^2}_{xz}] & [\gamma^{n^2}_{yz}] \end{bmatrix} \qquad \mathsf{eci} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_{n^2} & y_{n^2} \end{bmatrix}$$

contain the stress and strain components, and the coordinates of the integration points. The index n denotes the number of integration points used within the element, cf. plani4e. The number of columns in es and et follows the size of D. Note that for plane stress $\varepsilon_{zz} \neq 0$, and for plane strain $\sigma_{zz} \neq 0$.

Theory:

The strains and stresses are computed according to

$$\boldsymbol{\varepsilon} = \mathbf{B}^e \, \mathbf{a}^e$$

$$\boldsymbol{\sigma} = \mathbf{D} \, \boldsymbol{\varepsilon}$$

where the matrices $\mathbf{D}$, $\mathbf{B}^e$, and $\mathbf{a}^e$ are described in plani4e, and where the integration points are chosen as evaluation points.

Purpose:

Compute internal element force vector in a 4 node isoparametric element in plane strain or plane stress.

Syntax:

ef=plani4f(ex,ey,ep,es)

Description:

plani4f computes the internal element forces ef in a 4 node isoparametric element in plane strain or plane stress.

The input variables ex, ey and ep are defined in plani4e, and the input variable es is defined in plani4s.

The output variable

$$\mathsf{ef} = \mathbf{f}_i^{eT} = [\ f_{i1}\ f_{i2}\ \cdots\ f_{i8}\ ]$$

contains the components of the internal force vector.

Theory:
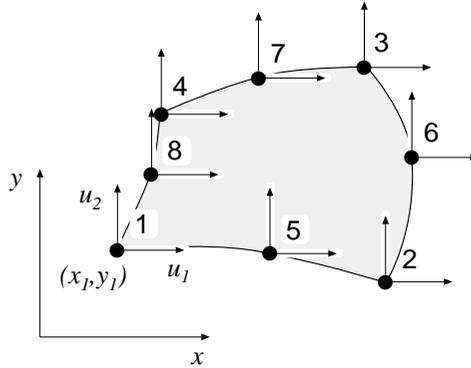
The internal force vector is computed according to

$$\mathbf{f}_i^e = \int_A \mathbf{B}^{eT} \boldsymbol{\sigma}\ t\ dA$$

where the matrices $\mathbf{B}^e$ and $\boldsymbol{\sigma}$ are defined in plani4e and plani4s, respectively.

Evaluation of the integral is done by Gauss integration.

Purpose:

Compute element matrices for an 8 node isoparametric element in plane strain or plane stress.

Syntax:

Ke=plani8e(ex,ey,ep,D)
[Ke,fe]=plani8e(ex,ey,ep,D,eq)

Description:

plani8e provides an element stiffness matrix Ke and an element load vector fe for an 8 node isoparametric element in plane strain or plane stress.

The element nodal coordinates x_1, y_1, x_2 etc. are supplied to the function by ex and ey. The type of analysis *ptype*, the element thickness t, and the number of Gauss points n are supplied by ep.

$$\begin{array}{lll} ptype = 1 & \text{plane stress} & (n \times n) \text{ integration points} \\ ptype = 2 & \text{plane strain} & n = 1, 2, 3 \end{array}$$

The material properties are supplied by the constitutive matrix D. Any arbitrary D-matrix with dimensions from (3×3) to (6×6) may be given. For an isotropic elastic material the constitutive matrix can be formed by the function hooke, see Section 4.

$$\text{ex} = \begin{bmatrix} x_1 & x_2 & \ldots & x_8 \end{bmatrix}$$
$$\text{ey} = \begin{bmatrix} y_1 & y_2 & \ldots & y_8 \end{bmatrix} \qquad \text{ep} = \begin{bmatrix} ptype & t & n \end{bmatrix}$$

$$\text{D} = \begin{bmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{bmatrix} \text{ or } \text{D} = \begin{bmatrix} D_{11} & D_{12} & D_{13} & D_{14} & [D_{15}] & [D_{16}] \\ D_{21} & D_{22} & D_{23} & D_{24} & [D_{25}] & [D_{26}] \\ D_{31} & D_{32} & D_{33} & D_{34} & [D_{35}] & [D_{36}] \\ D_{41} & D_{42} & D_{43} & D_{44} & [D_{45}] & [D_{46}] \\ [D_{51}] & [D_{52}] & [D_{53}] & [D_{54}] & [D_{55}] & [D_{56}] \\ [D_{61}] & [D_{62}] & [D_{63}] & [D_{64}] & [D_{65}] & [D_{66}] \end{bmatrix}$$

If different D_i -matrices are used in the Gauss points these D_i -matrices are stored in a global vector D. For numbering of the Gauss points, see eci in plani8s.

$$D = \begin{bmatrix} D_1 \\ D_2 \\ \vdots \\ D_{n^2} \end{bmatrix}$$

If uniformly distributed loads are applied to the element, the element load vector fe is computed. The input variable

$$eq = \begin{bmatrix} b_x \\ b_y \end{bmatrix}$$

containing loads per unit volume, b_x and b_y, is then given.

Theory:

The element stiffness matrix $\mathbf{K}^e$ and the element load vector $\mathbf{f}^e_l$, stored in Ke and fe, respectively, are computed according to

$$\mathbf{K}^e = \int_A \mathbf{B}^{eT}\,\mathbf{D}\,\mathbf{B}^e\,t\,dA$$

$$\mathbf{f}^e_l = \int_A \mathbf{N}^{eT}\,\mathbf{b}\,t\,dA$$

with the constitutive matrix $\mathbf{D}$ defined by D, and the body force vector $\mathbf{b}$ defined by eq.

The evaluation of the integrals for the isoparametric 8 node element is based on a displacement approximation $\mathbf{u}(\xi,\eta)$, expressed in a local coordinates system in terms of the nodal variables u_1, u_2, ..., u_{16} as

$$\mathbf{u}(\xi,\eta) = \mathbf{N}^e\,\mathbf{a}^e$$

where

$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} \quad \mathbf{N}^e = \begin{bmatrix} N_1^e & 0 & N_2^e & 0 & \dots & N_8^e & 0 \\ 0 & N_1^e & 0 & N_2^e & \dots & 0 & N_8^e \end{bmatrix} \quad \mathbf{a}^e = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{16} \end{bmatrix}$$

The element shape functions are given by

$$N_1^e = -\frac{1}{4}(1-\xi)(1-\eta)(1+\xi+\eta) \qquad N_5^e = \frac{1}{2}(1-\xi^2)(1-\eta)$$

$$N_2^e = -\frac{1}{4}(1+\xi)(1-\eta)(1-\xi+\eta) \qquad N_6^e = \frac{1}{2}(1+\xi)(1-\eta^2)$$

$$N_3^e = -\frac{1}{4}(1+\xi)(1+\eta)(1-\xi-\eta) \qquad N_7^e = \frac{1}{2}(1-\xi^2)(1+\eta)$$

$$N_4^e = -\frac{1}{4}(1-\xi)(1+\eta)(1+\xi-\eta) \qquad N_8^e = \frac{1}{2}(1-\xi)(1-\eta^2)$$

The matrix $\mathbf{B}^e$ is obtained as

$$\mathbf{B}^e = \tilde{\nabla}\mathbf{N}^e \qquad \text{where} \quad \tilde{\nabla} = \begin{bmatrix} \dfrac{\partial}{\partial x} & 0 \\ 0 & \dfrac{\partial}{\partial y} \\ \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} \end{bmatrix}$$
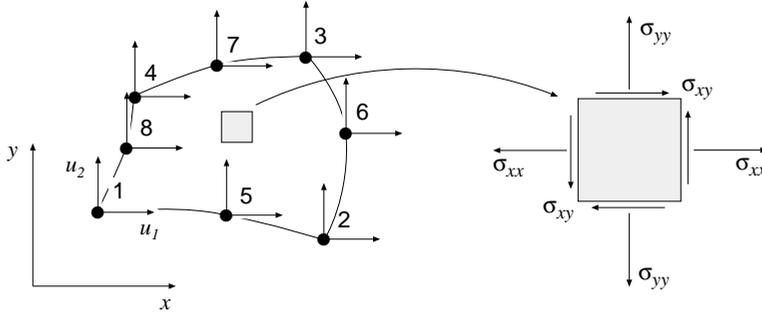
and where

$$\begin{bmatrix} \dfrac{\partial}{\partial x} \\ \dfrac{\partial}{\partial y} \end{bmatrix} = (\mathbf{J}^T)^{-1} \begin{bmatrix} \dfrac{\partial}{\partial \xi} \\ \dfrac{\partial}{\partial \eta} \end{bmatrix} \qquad\qquad \mathbf{J} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} \end{bmatrix}$$

If a larger $\mathbf{D}$-matrix than (3×3) is used for plane stress ($ptype = 1$), the $\mathbf{D}$-matrix is reduced to a (3×3) matrix by static condensation using $\sigma_{zz} = \sigma_{xz} = \sigma_{yz} = 0$. These stress components are connected with the rows 3, 5 and 6 in the D-matrix respectively.

If a larger $\mathbf{D}$-matrix than (3×3) is used for plane strain ($ptype = 2$), the $\mathbf{D}$-matrix is reduced to a (3×3) matrix using $\varepsilon_{zz} = \gamma_{xz} = \gamma_{yz} = 0$. This implies that a (3×3) $\mathbf{D}$-matrix is created by the rows and the columns 1, 2 and 4 from the original D-matrix.

Evaluation of the integrals is done by Gauss integration.

Purpose:

Compute stresses and strains in an 8 node isoparametric element in plane strain or plane stress.

Syntax:

[es,et,eci]=plani8s(ex,ey,ep,D,ed)

Description:

plani8s computes stresses es and the strains et in an 8 node isoparametric element in plane strain or plane stress.

The input variables ex, ey, ep and the matrix D are defined in plani8e. The vector ed contains the nodal displacements $\mathbf{a}^e$ of the element and is obtained by the function extract as

$$\mathsf{ed} = (\mathbf{a}^e)^T = [\, u_1 \ \ u_2 \ \ \ldots \ \ u_{16}\,]$$

The output variables

$$\mathsf{es} = \boldsymbol{\sigma}^T = \begin{bmatrix} \sigma_{xx}^1 & \sigma_{yy}^1 & [\sigma_{zz}^1] & \sigma_{xy}^1 & [\sigma_{xz}^1] & [\sigma_{yz}^1] \\ \sigma_{xx}^2 & \sigma_{yy}^2 & [\sigma_{zz}^2] & \sigma_{xy}^2 & [\sigma_{xz}^2] & [\sigma_{yz}^2] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \sigma_{xx}^{n^2} & \sigma_{yy}^{n^2} & [\sigma_{zz}^{n^2}] & \sigma_{xy}^{n^2} & [\sigma_{xz}^{n^2}] & [\sigma_{yz}^{n^2}] \end{bmatrix}$$

$$\mathsf{et} = \boldsymbol{\varepsilon}^T = \begin{bmatrix} \varepsilon_{xx}^1 & \varepsilon_{yy}^1 & [\varepsilon_{zz}^1] & \gamma_{xy}^1 & [\gamma_{xz}^1] & [\gamma_{yz}^1] \\ \varepsilon_{xx}^2 & \varepsilon_{yy}^2 & [\varepsilon_{zz}^2] & \gamma_{xy}^2 & [\gamma_{xz}^2] & [\gamma_{yz}^2] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \varepsilon_{xx}^{n^2} & \varepsilon_{yy}^{n^2} & [\varepsilon_{zz}^{n^2}] & \gamma_{xy}^{n^2} & [\gamma_{xz}^{n^2}] & [\gamma_{yz}^{n^2}] \end{bmatrix} \qquad \mathsf{eci} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_{n^2} & y_{n^2} \end{bmatrix}$$

contain the stress and strain components, and the coordinates of the integration points. The index n denotes the number of integration points used within the element, cf. plani8e. The number of columns in es and et follows the size of D. Note that for plane stress $\varepsilon_{zz} \neq 0$, and for plane strain $\sigma_{zz} \neq 0$.

Theory:

The strains and stresses are computed according to

$$\boldsymbol{\varepsilon} = \mathbf{B}^e \, \mathbf{a}^e$$

$$\boldsymbol{\sigma} = \mathbf{D} \, \boldsymbol{\varepsilon}$$

where the matrices $\mathbf{D}$, $\mathbf{B}^e$, and $\mathbf{a}^e$ are described in plani8e, and where the integration points are chosen as evaluation points.

Purpose:

Compute internal element force vector in an 8 node isoparametric element in plane strain or plane stress.

Syntax:

ef=plani8f(ex,ey,ep,es)

Description:

plani8f computes the internal element forces ef in an 8 node isoparametric element in plane strain or plane stress.

The input variables ex, ey and ep are defined in plani8e, and the input variable es is defined in plani8s.

The output variable

$$\mathsf{ef} = \mathbf{f}_i^{eT} = [\; f_{i1}\; f_{i2}\; \cdots\; f_{i16}\;]$$

contains the components of the internal force vector.

Theory:

The internal force vector is computed according to

$$\mathbf{f}_i^e = \int_A \mathbf{B}^{eT}\boldsymbol{\sigma}\; t\; dA$$

where the matrices $\mathbf{B}^e$ and $\boldsymbol{\sigma}$ are defined in plani8e and plani8s, respectively.

Evaluation of the integral is done by Gauss integration.

Purpose:

Compute element matrices for an 8 node isoparametric solid element.
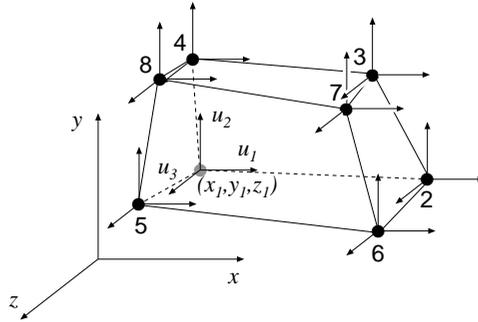
Syntax:

Ke=soli8e(ex,ey,ez,ep,D)
[Ke,fe]=soli8e(ex,ey,ez,ep,D,eq)

Description:

soli8e provides an element stiffness matrix Ke and an element load vector fe for an 8 node isoparametric solid element.

The element nodal coordinates x_1, y_1, z_1, x_2 etc. are supplied to the function by ex, ey and ez, and the number of Gauss points n are supplied by ep.

$$(n \times n) \text{ integration points, } n = 1, 2, 3$$

The material properties are supplied by the constitutive matrix D. Any arbitrary D-matrix with dimensions (6×6) may be given. For an isotropic elastic material the constitutive matrix can be formed by the function hooke, see Section 4.

$$
\begin{aligned}
\mathsf{ex} &= \begin{bmatrix} x_1 & x_2 & \ldots & x_8 \end{bmatrix} \\
\mathsf{ey} &= \begin{bmatrix} y_1 & y_2 & \ldots & y_8 \end{bmatrix} \\
\mathsf{ez} &= \begin{bmatrix} z_1 & z_2 & \ldots & z_8 \end{bmatrix}
\end{aligned}
\qquad
\mathsf{ep} = \begin{bmatrix} n \end{bmatrix}
\qquad
\mathsf{D} = \begin{bmatrix}
D_{11} & D_{12} & \cdots & D_{16} \\
D_{21} & D_{22} & \cdots & D_{26} \\
\vdots & \vdots & \ddots & \vdots \\
D_{61} & D_{62} & \cdots & D_{66}
\end{bmatrix}
$$

If different D_i -matrices are used in the Gauss points these D_i -matrices are stored in a global vector D. For numbering of the Gauss points, see eci in soli8s.

$$
\mathsf{D} = \begin{bmatrix}
\mathsf{D}_1 \\
\mathsf{D}_2 \\
\vdots \\
\mathsf{D}_{n^3}
\end{bmatrix}
$$

If uniformly distributed loads are applied to the element, the element load vector fe is computed. The input variable

$$\mathsf{eq} = \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}$$

containing loads per unit volume, b_x, b_y, and b_z, is then given.

Theory:

The element stiffness matrix $\mathbf{K}^e$ and the element load vector $\mathbf{f}_l^e$, stored in Ke and fe, respectively, are computed according to

$$\mathbf{K}^e = \int_V \mathbf{B}^{eT} \, \mathbf{D} \, \mathbf{B}^e \, dV$$

$$\mathbf{f}_l^e = \int_V \mathbf{N}^{eT} \, \mathbf{b} \, dV$$

with the constitutive matrix $\mathbf{D}$ defined by D, and the body force vector $\mathbf{b}$ defined by eq.

The evaluation of the integrals for the isoparametric 8 node solid element is based on a displacement approximation $\mathbf{u}(\xi, \eta, \zeta)$, expressed in a local coordinates system in terms of the nodal variables u_1, u_2, ..., u_{24} as

$$\mathbf{u}(\xi, \eta, \zeta) = \mathbf{N}^e \, \mathbf{a}^e$$

where

$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \quad \mathbf{N}^e = \begin{bmatrix} N_1^e & 0 & 0 & N_2^e & 0 & 0 & \dots & N_8^e & 0 & 0 \\ 0 & N_1^e & 0 & 0 & N_2^e & 0 & \dots & 0 & N_8^e & 0 \\ 0 & 0 & N_1^e & 0 & 0 & N_2^e & \dots & 0 & 0 & N_8^e \end{bmatrix} \quad \mathbf{a}^e = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{24} \end{bmatrix}$$

The element shape functions are given by

$$N_1^e = \frac{1}{8}(1 - \xi)(1 - \eta)(1 - \zeta) \qquad N_5^e = \frac{1}{8}(1 - \xi)(1 - \eta)(1 + \zeta)$$

$$N_2^e = \frac{1}{8}(1 + \xi)(1 - \eta)(1 - \zeta) \qquad N_6^e = \frac{1}{8}(1 + \xi)(1 - \eta)(1 + \zeta)$$

$$N_3^e = \frac{1}{8}(1 + \xi)(1 + \eta)(1 - \zeta) \qquad N_7^e = \frac{1}{8}(1 + \xi)(1 + \eta)(1 + \zeta)$$

$$N_4^e = \frac{1}{8}(1 - \xi)(1 + \eta)(1 - \zeta) \qquad N_8^e = \frac{1}{8}(1 - \xi)(1 + \eta)(1 + \zeta)$$

The $\mathbf{B}^e$-matrix is obtained as
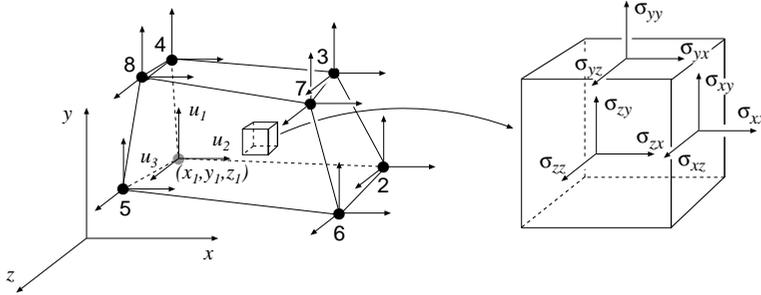
$$\mathbf{B}^e = \tilde{\nabla} \mathbf{N}^e$$

where

$$\tilde{\nabla} = \begin{bmatrix} \dfrac{\partial}{\partial x} & 0 & 0 \\[2mm] 0 & \dfrac{\partial}{\partial y} & 0 \\[2mm] 0 & 0 & \dfrac{\partial}{\partial z} \\[2mm] \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} & 0 \\[2mm] \dfrac{\partial}{\partial z} & 0 & \dfrac{\partial}{\partial x} \\[2mm] 0 & \dfrac{\partial}{\partial z} & \dfrac{\partial}{\partial y} \end{bmatrix} \qquad\qquad \begin{bmatrix} \dfrac{\partial}{\partial x} \\[2mm] \dfrac{\partial}{\partial y} \\[2mm] \dfrac{\partial}{\partial z} \end{bmatrix} = (\mathbf{J}^T)^{-1} \begin{bmatrix} \dfrac{\partial}{\partial \xi} \\[2mm] \dfrac{\partial}{\partial \eta} \\[2mm] \dfrac{\partial}{\partial \zeta} \end{bmatrix}$$

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} & \dfrac{\partial x}{\partial \zeta} \\[2mm] \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} & \dfrac{\partial y}{\partial \zeta} \\[2mm] \dfrac{\partial z}{\partial \xi} & \dfrac{\partial z}{\partial \eta} & \dfrac{\partial z}{\partial \zeta} \end{bmatrix}$$

Evaluation of the integrals is done by Gauss integration.

Purpose:

Compute stresses and strains in an 8 node isoparametric solid element.

Syntax:

[es,et,eci]=soli8s(ex,ey,ez,ep,D,ed)

Description:

soli8s computes stresses es and the strains et in an 8 node isoparametric solid element.

The input variables ex, ey, ez, ep and the matrix D are defined in soli8e. The vector ed contains the nodal displacements $\mathbf{a}^e$ of the element and is obtained by the function extract as

$$\mathsf{ed} = (\mathbf{a}^e)^T = [\; u_1 \;\; u_2 \;\; \ldots \;\; u_{24} \;]$$

The output variables

$$\mathsf{es} = \boldsymbol{\sigma}^T = \begin{bmatrix} \sigma_{xx}^1 & \sigma_{yy}^1 & \sigma_{zz}^1 & \sigma_{xy}^1 & \sigma_{xz}^1 & \sigma_{yz}^1 \\ \sigma_{xx}^2 & \sigma_{yy}^2 & \sigma_{zz}^2 & \sigma_{xy}^2 & \sigma_{xz}^2 & \sigma_{yz}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \sigma_{xx}^{n^3} & \sigma_{yy}^{n^3} & \sigma_{zz}^{n^3} & \sigma_{xy}^{n^3} & \sigma_{xz}^{n^3} & \sigma_{yz}^{n^3} \end{bmatrix}$$

$$\mathsf{et} = \boldsymbol{\varepsilon}^T = \begin{bmatrix} \varepsilon_{xx}^1 & \varepsilon_{yy}^1 & \varepsilon_{zz}^1 & \gamma_{xy}^1 & \gamma_{xz}^1 & \gamma_{yz}^1 \\ \varepsilon_{xx}^2 & \varepsilon_{yy}^2 & \varepsilon_{zz}^2 & \gamma_{xy}^2 & \gamma_{xz}^2 & \gamma_{yz}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \varepsilon_{xx}^{n^3} & \varepsilon_{yy}^{n^3} & \varepsilon_{zz}^{n^3} & \gamma_{xy}^{n^3} & \gamma_{xz}^{n^3} & \gamma_{yz}^{n^3} \end{bmatrix} \qquad \mathsf{eci} = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_{n^3} & y_{n^3} & z_{n^3} \end{bmatrix}$$

contain the stress and strain components, and the coordinates of the integration points. The index n denotes the number of integration points used within the element, cf. soli8e.

Theory:

The strains and stresses are computed according to

$$\boldsymbol{\varepsilon} = \mathbf{B}^e\,\mathbf{a}^e$$

$$\boldsymbol{\sigma} = \mathbf{D}\,\boldsymbol{\varepsilon}$$

where the matrices $\mathbf{D}$, $\mathbf{B}^e$, and $\mathbf{a}^e$ are described in soli8e, and where the integration points are chosen as evaluation points.

Purpose:

Compute internal element force vector in an 8 node isoparametric solid element.

Syntax:

ef=soli8f(ex,ey,ez,ep,es)

Description:

soli8f computes the internal element forces ef in an 8 node isoparametric solid element.

The input variables ex, ey, ez and ep are defined in soli8e, and the input variable es is defined in soli8s.

The output variable

$$\mathsf{ef} = \mathbf{f}_i^{eT} = [\ f_{i1}\ f_{i2}\ \ldots\ f_{i24}\ ]$$

contains the components of the internal force vector.

Theory:

The internal force vector is computed according to

$$\mathbf{f}_i^e = \int_V \mathbf{B}^{eT}\boldsymbol{\sigma}\ dV$$

where the matrices $\mathbf{B}$ and $\boldsymbol{\sigma}$ are defined in soli8e and soli8s, respectively.

Evaluation of the integral is done by Gauss integration.

5.6 Beam elements

Beam elements are available for two, and three dimensional linear static analysis. Two dimensional beam elements for nonlinear geometric and dynamic analysis are also available.
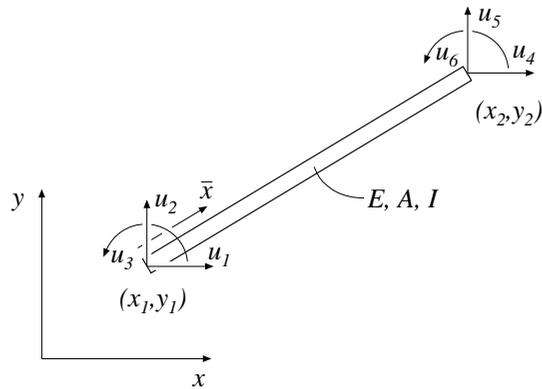
Beam elements

beam2e
beam2t
beam2w
beam2g
beam2d

beam3e
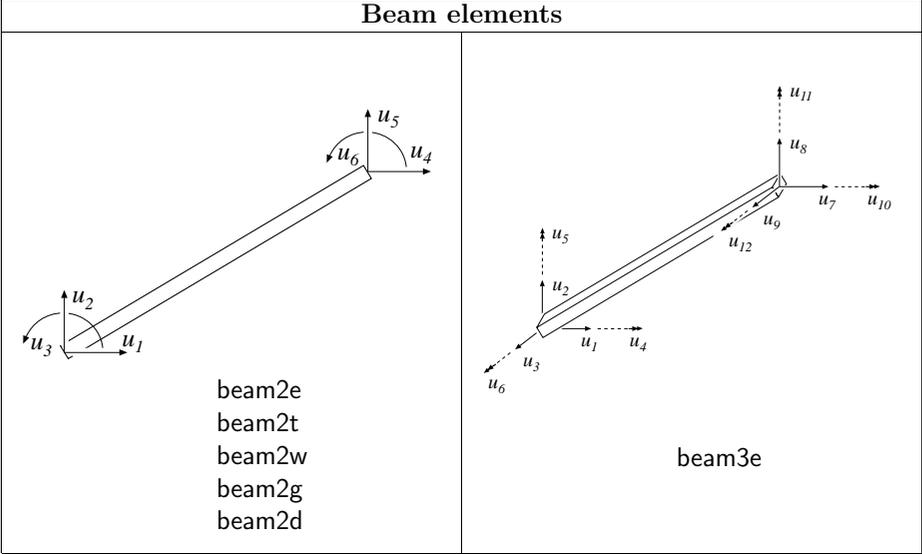
2D beam elements	
beam2e	Compute element matrices
beam2s	Compute section forces
beam2t	Compute element matrices for Timoshenko beam element
beam2ts	Compute section forces for Timoshenko beam element
beam2w	Compute element matrices for beam element on elastic foundation
beam2ws	Compute section forces for beam element on elastic foundation
beam2g	Compute element matrices with respect to geometric nonlinearity
beam2gs	Compute section forces for geometric nonlinear beam element
beam2d	Compute element matrices, dynamic analysis
beam2ds	Compute section forces, dynamic analysis

3D beam elements	
beam3e	Compute element matrices
beam3s	Compute section forces

Purpose:

    Compute element stiffness matrix for a two dimensional beam element.

Syntax:

    Ke=beam2e(ex,ey,ep)
    [Ke,fe]=beam2e(ex,ey,ep,eq)

Description:

    beam2e provides the global element stiffness matrix Ke for a two dimensional beam element.

    The input variables

$$\text{ex} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \qquad \text{ep} = \begin{bmatrix} E & A & I \end{bmatrix}$$
$$\text{ey} = \begin{bmatrix} y_1 & y_2 \end{bmatrix}$$

supply the element nodal coordinates x_1, y_1, x_2, and y_2, the modulus of elasticity E, the cross section area A, and the moment of inertia I.

    The element load vector fe can also be computed if uniformly distributed loads are applied to the element. The optional input variable

$$\text{eq} = \begin{bmatrix} q_{\bar{x}} & q_{\bar{y}} \end{bmatrix}$$

then contains the distributed loads per unit length, $q_{\bar{x}}$ and $q_{\bar{y}}$.
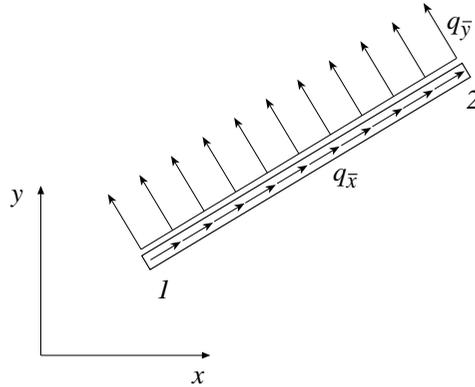
Theory:

The element stiffness matrix $\mathbf{K}^e$, stored in Ke, is computed according to

$$\mathbf{K}^e = \mathbf{G}^T \bar{\mathbf{K}}^e \mathbf{G}$$

where

$$
\bar{\mathbf{K}}^e =
\begin{bmatrix}
\frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\
0 & \frac{12EI}{L^3} & \frac{6EI}{L^2} & 0 & -\frac{12EI}{L^3} & \frac{6EI}{L^2} \\
0 & \frac{6EI}{L^2} & \frac{4EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{2EI}{L} \\
-\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\
0 & -\frac{12EI}{L^3} & -\frac{6EI}{L^2} & 0 & \frac{12EI}{L^3} & -\frac{6EI}{L^2} \\
0 & \frac{6EI}{L^2} & \frac{2EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{4EI}{L}
\end{bmatrix}
$$

$$
\mathbf{G} =
\begin{bmatrix}
n_{x\bar{x}} & n_{y\bar{x}} & 0 & 0 & 0 & 0 \\
n_{x\bar{y}} & n_{y\bar{y}} & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & n_{x\bar{x}} & n_{y\bar{x}} & 0 \\
0 & 0 & 0 & n_{x\bar{y}} & n_{y\bar{y}} & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
$$

The transformation matrix $\mathbf{G}$ contains the direction cosines

$$n_{x\bar{x}} = n_{y\bar{y}} = \frac{x_2 - x_1}{L} \qquad n_{y\bar{x}} = -n_{x\bar{y}} = \frac{y_2 - y_1}{L}$$

where the length

$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

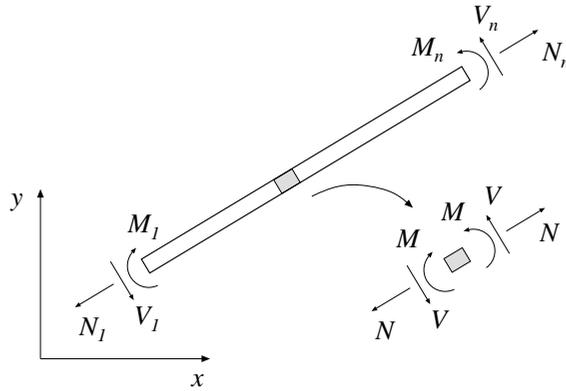The element load vector $\mathbf{f}_l^e$, stored in fe, is computed according to

$$\mathbf{f}_l^e = \mathbf{G}^T \bar{\mathbf{f}}_l^e$$

where

$$\bar{\mathbf{f}}_l^e = \begin{bmatrix} \dfrac{q_{\bar{x}}L}{2} \\[2mm] \dfrac{q_{\bar{y}}L}{2} \\[2mm] \dfrac{q_{\bar{y}}L^2}{12} \\[2mm] \dfrac{q_{\bar{x}}L}{2} \\[2mm] \dfrac{q_{\bar{y}}L}{2} \\[2mm] -\dfrac{q_{\bar{y}}L^2}{12} \end{bmatrix}$$

Purpose:

Compute section forces in a two dimensional beam element.

Syntax:

es=beam2s(ex,ey,ep,ed)
es=beam2s(ex,ey,ep,ed,eq)
[es,edi,eci]=beam2s(ex,ey,ep,ed,eq,n)

Description:

beam2s computes the section forces and displacements in local directions along the beam element beam2e.

The input variables ex, ey, ep and eq are defined in beam2e, and the element displacements, stored in ed, are obtained by the function extract. If distributed loads are applied to the element, the variable eq must be included. The number of evaluation points for section forces and displacements are determined by n. If n is omitted, only the ends of the beam are evaluated.

The output variables

$$\text{es} = [\ \mathbf{N} \ \ \mathbf{V} \ \ \mathbf{M} \ ] \qquad \text{edi} = [\ \bar{\mathbf{u}} \ \ \bar{\mathbf{v}} \ ] \qquad \text{eci} = [\ \bar{\mathbf{x}} \ ]$$

consist of column matrices that contain the section forces, the displacements, and the evaluation points on the local x-axis. The explicit matrix expressions are

$$\text{es} = \begin{bmatrix} N_1 & V_1 & M_1 \\ N_2 & V_2 & M_2 \\ \vdots & \vdots & \vdots \\ N_n & V_n & M_n \end{bmatrix} \qquad \text{edi} = \begin{bmatrix} \bar{u}_1 & \bar{v}_1 \\ \bar{u}_2 & \bar{v}_2 \\ \vdots & \vdots \\ \bar{u}_n & \bar{v}_n \end{bmatrix} \qquad \text{eci} = \begin{bmatrix} 0 \\ \bar{x}_2 \\ \vdots \\ \bar{x}_{n-1} \\ L \end{bmatrix}$$

where L is the length of the beam element.

Theory:

The evaluation of the section forces is based on the solutions of the basic equations

$$EA\frac{d^2\bar{u}}{d\bar{x}^2} + q_{\bar{x}} = 0 \qquad\qquad EI\frac{d^4\bar{v}}{d\bar{x}^4} - q_{\bar{y}} = 0$$

From these equations, the displacements along the beam element are obtained as the sum of the homogeneous and the particular solutions

$$\mathbf{u} = \left[\begin{array}{c} \bar{u}(\bar{x}) \\ \bar{v}(\bar{x}) \end{array} \right] = \mathbf{u}_h + \mathbf{u}_p$$

where

$$\mathbf{u}_h = \bar{\mathbf{N}}\,\mathbf{C}^{-1}\,\mathbf{G}\,\mathbf{a}^e \qquad\qquad \mathbf{u}_p = \left[\begin{array}{c} \bar{u}_p(\bar{x}) \\ \bar{v}_p(\bar{x}) \end{array} \right] = \left[\begin{array}{c} \dfrac{q_{\bar{x}}L\bar{x}}{2EA}(1 - \dfrac{\bar{x}}{L}) \\[3mm] \dfrac{q_{\bar{y}}L^2\bar{x}^2}{24EI}(1 - \dfrac{\bar{x}}{L})^2 \end{array} \right]$$

and

$$\bar{\mathbf{N}} = \left[\begin{array}{cccccc} 1 & \bar{x} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \bar{x} & \bar{x}^2 & \bar{x}^3 \end{array} \right] \quad \mathbf{C} = \left[\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & L & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & L & L^2 & L^3 \\ 0 & 0 & 0 & 1 & 2L & 3L^2 \end{array} \right] \quad \mathbf{a}^e = \left[\begin{array}{c} u_1 \\ u_2 \\ \vdots \\ u_6 \end{array} \right]$$

The transformation matrix $\mathbf{G}^e$ and nodal displacements $\mathbf{a}^e$ are described in beam2e. Note that the transpose of $\mathbf{a}^e$ is stored in ed.

Finally the section forces are obtained from

$$N = EA\frac{d\bar{u}}{d\bar{x}} \qquad\qquad V = -EI\frac{d^3\bar{v}}{d\bar{x}^3} \qquad\qquad M = EI\frac{d^2\bar{v}}{d\bar{x}^2}$$
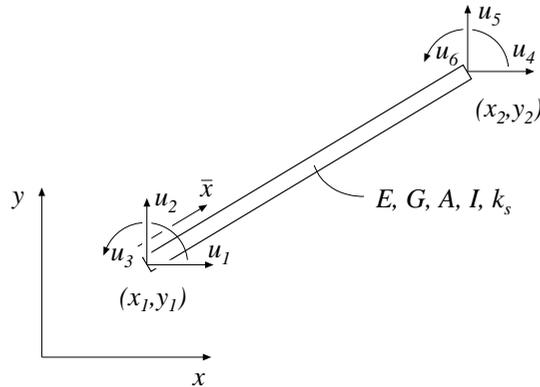
Examples:

Section forces or element displacements can easily be plotted. The bending moment M along the beam is plotted by

```
>> plot(eci,es(:,3))
```

Purpose:

Compute element stiffness matrix for a two dimensional Timoshenko beam element.

Syntax:

Ke=beam2t(ex,ey,ep)
[Ke,fe]=beam2t(ex,ey,ep,eq)

Description:

beam2t provides the global element stiffness matrix Ke for a two dimensional Timoshenko beam element.

The input variables

$$\mathsf{ex} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}$$
$$\mathsf{ey} = \begin{bmatrix} y_1 & y_2 \end{bmatrix} \qquad \mathsf{ep} = \begin{bmatrix} E & G & A & I & k_s \end{bmatrix}$$

supply the element nodal coordinates x_1, y_1, x_2, and y_2, the modulus of elasticity E, the shear modulus G, the cross section area A, the moment of inertia I and the shear correction factor k_s.

The element load vector fe can also be computed if uniformly distributed loads are applied to the element. The optional input variable

$$\mathsf{eq} = \begin{bmatrix} q_{\bar{x}} & q_{\bar{y}} \end{bmatrix}$$

then contains the distributed loads per unit length, $q_{\bar{x}}$ and $q_{\bar{y}}$.
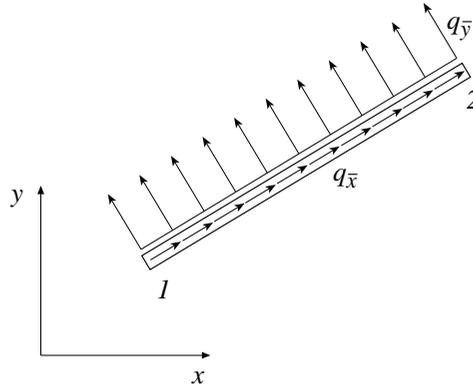
Theory:

The element stiffness matrix $\mathbf{K}^e$, stored in Ke, is computed according to

$$\mathbf{K}^e = \mathbf{G}^T \bar{\mathbf{K}}^e \mathbf{G}$$

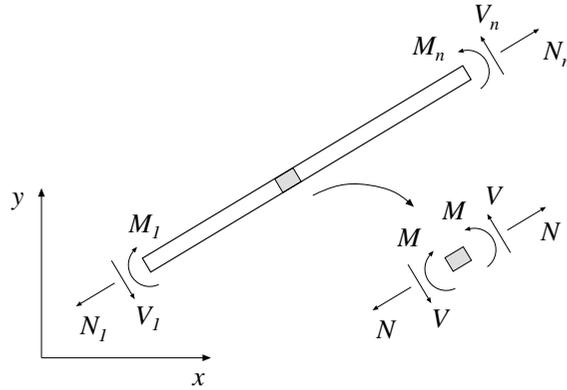where $\mathbf{G}$ is described in beam2e, and $\bar{\mathbf{K}}^e$ is given by

$$\bar{\mathbf{K}}^e = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\[2mm] 0 & \frac{12EI}{L^3(1+\mu)} & \frac{6EI}{L^2(1+\mu)} & 0 & -\frac{12EI}{L^3(1+\mu)} & \frac{6EI}{L^2(1+\mu)} \\[2mm] 0 & \frac{6EI}{L^2(1+\mu)} & \frac{4EI(1+\frac{\mu}{4})}{L(1+\mu)} & 0 & -\frac{6EI}{L^2(1+\mu)} & \frac{2EI(1-\frac{\mu}{2})}{L(1+\mu)} \\[2mm] -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\[2mm] 0 & -\frac{12EI}{L^3(1+\mu)} & -\frac{6EI}{L^2(1+\mu)} & 0 & \frac{12EI}{L^3(1+\mu)} & -\frac{6EI}{L^2(1+\mu)} \\[2mm] 0 & \frac{6EI}{L^2(1+\mu)} & \frac{2EI(1-\frac{\mu}{2})}{L(1+\mu)} & 0 & -\frac{6EI}{L^2(1+\mu)} & \frac{4EI(1+\frac{\mu}{4})}{L(1+\mu)} \end{bmatrix}$$

with

$$\mu = \frac{12EI}{L^2 G A k_s}$$

Purpose:

Compute section forces in a two dimensional Timoshenko beam element.

Syntax:

es=beam2ts(ex,ey,ep,ed)
es=beam2ts(ex,ey,ep,ed,eq)
[es,edi,eci]=beam2ts(ex,ey,ep,ed,eq,n)

Description:

beam2ts computes the section forces and displacements in local directions along the beam element beam2t.

The input variables ex, ey, ep and eq are defined in beam2t. The element displacements, stored in ed, are obtained by the function extract. If distributed loads are applied to the element, the variable eq must be included. The number of evaluation points for section forces and displacements are determined by n. If n is omitted, only the ends of the beam are evaluated.

The output variables

$$\text{es} = [\,\mathbf{N\ V\ M}\,] \qquad \text{edi} = [\,\bar{\mathbf{u}}\ \bar{\mathbf{v}}\,] \qquad \text{eci} = [\bar{\mathbf{x}}]$$

consist of column matrices that contain the section forces, the displacements and rotation of the cross section (note that the rotation θ is not equal to $\frac{d\bar{v}}{d\bar{x}}$), and the evaluation points on the local x-axis. The explicit matrix expressions are

$$\text{es} = \begin{bmatrix} N_1 & V_1 & M_1 \\ N_2 & V_2 & M_2 \\ \vdots & \vdots & \vdots \\ N_n & V_n & M_n \end{bmatrix} \qquad \text{edi} = \begin{bmatrix} \bar{u}_1 & \bar{v}_1 & \theta_1 \\ \bar{u}_2 & \bar{v}_2 & \theta_2 \\ \vdots & \vdots & \vdots \\ \bar{u}_n & \bar{v}_n & \theta_n \end{bmatrix} \qquad \text{eci} = \begin{bmatrix} 0 \\ \bar{x}_2 \\ \vdots \\ \bar{x}_{n-1} \\ L \end{bmatrix}$$

where L is the length of the beam element.

Theory:

The evaluation of the section forces is based on the solutions of the basic equations

$$EA\frac{d^2\bar{u}}{d\bar{x}^2} + q_{\bar{x}} = 0 \qquad EI\frac{d^3\theta}{d\bar{x}^3} - q_{\bar{y}} = 0 \qquad EI\frac{d^4\bar{v}}{d\bar{x}^4} - q_{\bar{y}} = 0$$

(The equations are valid if $q_{\bar{y}}$ is not more than a linear function of $\bar{x}$). From these equations, the displacements along the beam element are obtained as the sum of the homogeneous and the particular solutions

$$\mathbf{u} = \begin{bmatrix} \bar{u}(\bar{x}) \\ \bar{v}(\bar{x}) \\ \theta(\bar{x}) \end{bmatrix} = \mathbf{u}_h + \mathbf{u}_p$$

where

$$\mathbf{u}_h = \bar{\mathbf{N}}\,\mathbf{C}^{-1}\,\mathbf{G}\,\mathbf{a}^e \qquad \mathbf{u}_p = \begin{bmatrix} \bar{u}_p(\bar{x}) \\ \bar{v}_p(\bar{x}) \\ \theta_p(\bar{x}) \end{bmatrix} = \begin{bmatrix} \dfrac{q_{\bar{x}}L\bar{x}}{2EA}(1 - \dfrac{\bar{x}}{L}) \\ \dfrac{q_{\bar{y}}L^2\bar{x}^2}{24EI}(1 - \dfrac{\bar{x}}{L})^2 + \dfrac{q_{\bar{y}}L\bar{x}}{2GA\,k_s}(1 - \dfrac{\bar{x}}{L}) \\ \dfrac{q_{\bar{y}}L^2\bar{x}}{12EI}(1 - \dfrac{2\bar{x}}{L})(1 - \dfrac{\bar{x}}{L}) \end{bmatrix}$$

and

$$\bar{\mathbf{N}} = \begin{bmatrix} 1 & \bar{x} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \bar{x} & \bar{x}^2 & \bar{x}^3 \\ 0 & 0 & 0 & 1 & 2x & 3(x^2 + 2\alpha) \end{bmatrix} \qquad \alpha = \frac{EI}{GA\,k_s}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 6\alpha \\ 1 & L & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & L & L^2 & L^3 \\ 0 & 0 & 0 & 1 & 2L & 3(L^2 + 2\alpha) \end{bmatrix} \qquad \mathbf{a}^e = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_6 \end{bmatrix}$$
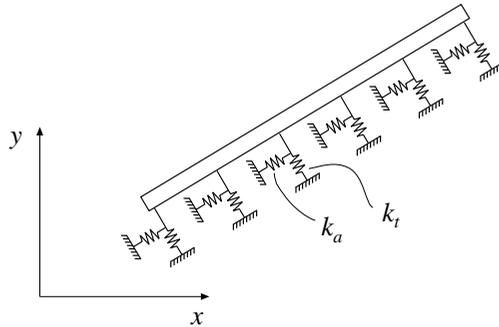
The transformation matrix $\mathbf{G}$ and nodal displacements $\mathbf{a}^e$ are described in beam2e. Note that the transpose of $\mathbf{a}^e$ is stored in ed.

Finally the section forces are obtained from

$$N = EA\frac{d\bar{u}}{d\bar{x}} \qquad V = GA\,k_s(\frac{d\bar{v}}{d\bar{x}} - \theta) \qquad M = EI\frac{d\theta}{d\bar{x}}$$

Purpose:

Compute element stiffness matrix for a two dimensional beam element on elastic foundation.

Syntax:

Ke=beam2w(ex,ey,ep)
[Ke,fe]=beam2w(ex,ey,ep,eq)

Description:

beam2w provides the global element stiffness matrix Ke for a two dimensional beam element on elastic foundation.

The input variables ex and ey are described in beam2e, and

$$\mathsf{ep} = [\; E \;\; A \;\; I \;\; k_a \;\; k_t \;]$$

contains the modulus of elasticity E, the cross section area A, the moment of inertia I, the spring stiffness in the axial direction k_a, and the spring stiffness in the transverse direction k_t.

The element load vector fe can also be computed if uniformly distributed loads are applied to the element. The optional input variable eq, described in beam2e, contains the distributed loads.

Theory:

The element stiffness matrix $\mathbf{K}^e$, stored in Ke, is computed according to

$$\mathbf{K}^e = \mathbf{G}^T(\bar{\mathbf{K}}^e + \bar{\mathbf{K}}^e_s)\mathbf{G}$$

where $\mathbf{G}$ and $\bar{\mathbf{K}}^e$ are described in beam2e.
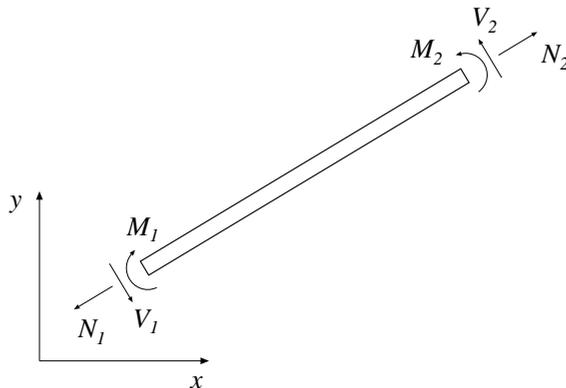
The matrix $\bar{\mathbf{K}}^e_s$ is given by

$$\bar{\mathbf{K}}^e_s = \frac{L}{420}\begin{bmatrix} 140k_a & 0 & 0 & 70k_a & 0 & 0 \\ 0 & 156k_t & 22k_tL & 0 & 54k_t & -13k_tL \\ 0 & 22k_tL & 4k_tL^2 & 0 & 13k_tL & -3k_tL^2 \\ 70k_a & 0 & 0 & 140k_a & 0 & 0 \\ 0 & 54k_t & 13k_tL & 0 & 156k_t & -22k_tL \\ 0 & -13k_tL & -3k_tL^2 & 0 & -22k_tL & 4k_tL^2 \end{bmatrix}$$

where the length

$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Purpose:

Compute section forces in a two dimensional beam element on elastic foundation.

Syntax:

es=beam2ws(ex,ey,ep,ed)
es=beam2ws(ex,ey,ep,ed,eq)

Description:

beam2ws computes the section forces at the ends of the beam element on elastic foundation beam2w.

The input variables ex, ey, ep and eq are defined in beam2w, and the element displacements, stored in ed, are obtained by the function extract. If distributed loads are applied to the element the variable eq must be included.

The output variable

$$\text{es} = \left[\begin{array}{ccc} N_1 & V_1 & M_1 \\ N_2 & V_2 & M_2 \end{array} \right]$$

contains the section forces at the ends of the beam.

Theory:

The section forces at the ends of the beam are obtained from the element force vector

$$\bar{\mathbf{P}} = \begin{bmatrix} -N_1 & -V_1 & -M_1 & N_2 & V_2 & M_2 \end{bmatrix}^T$$

computed according to

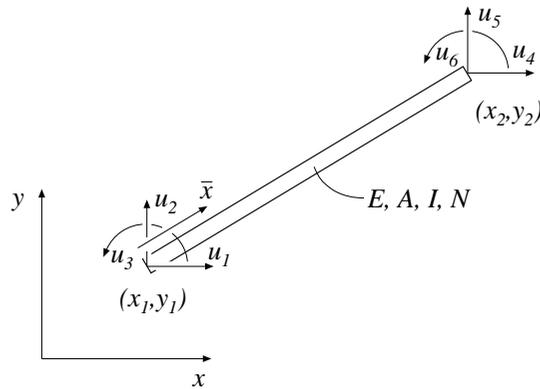$$\bar{\mathbf{P}} = (\bar{\mathbf{K}}^e + \bar{\mathbf{K}}_s^e)\, \mathbf{G}\, \mathbf{a}^e - \bar{\mathbf{f}}_l^e$$

The matrices $\bar{\mathbf{K}}^e$ and $\mathbf{G}$, are described in beam2e, and the matrix $\bar{\mathbf{K}}_s^e$ is described in beam2w. The nodal displacements

$$\mathbf{a}^e = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 & u_5 & u_6 \end{bmatrix}^T$$

are shown in beam2e. Note that the transpose of $\mathbf{a}^e$ is stored in ed.

Purpose:

Compute element stiffness matrix for a two dimensional nonlinear beam element.

Syntax:

Ke=beam2g(ex,ey,ep,N)
[Ke,fe]=beam2g(ex,ey,ep,N,eq)

Description:

beam2g provides the global element stiffness matrix Ke for a two dimensional beam element with respect to geometrical nonlinearity.

The input variables ex, ey, and ep are described in beam2e, and

$$\mathsf{N} = [\ N\ ]$$

contains the value of the predefined normal force N, which is positive in tension.

The element load vector fe can also be computed if a uniformly distributed transverse load is applied to the element. The optional input variable

$$\mathsf{eq} = [\ q_{\bar{y}}\ ]$$

then contains the distributed transverse load per unit length, $q_{\bar{y}}$. Note that eq is a scalar and not a vector as in beam2e

Theory:

The element stiffness matrix $\mathbf{K}^e$, stored in the variable Ke, is computed according to

$$\mathbf{K}^e = \mathbf{G}^T \bar{\mathbf{K}}^e \mathbf{G}$$

where $\mathbf{G}$ is described in beam2e, and $\bar{\mathbf{K}}^e$ is given by

$$
\bar{\mathbf{K}}^e =
\begin{bmatrix}
\frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\
0 & \frac{12EI}{L^3}\phi_5 & \frac{6EI}{L^2}\phi_2 & 0 & -\frac{12EI}{L^3}\phi_5 & \frac{6EI}{L^2}\phi_2 \\
0 & \frac{6EI}{L^2}\phi_2 & \frac{4EI}{L}\phi_3 & 0 & -\frac{6EI}{L^2}\phi_2 & \frac{2EI}{L}\phi_4 \\
-\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\
0 & -\frac{12EI}{L^3}\phi_5 & -\frac{6EI}{L^2}\phi_2 & 0 & \frac{12EI}{L^3}\phi_5 & -\frac{6EI}{L^2}\phi_2 \\
0 & \frac{6EI}{L^2}\phi_2 & \frac{2EI}{L}\phi_4 & 0 & -\frac{6EI}{L^2}\phi_2 & \frac{4EI}{L}\phi_3
\end{bmatrix}
$$

For axial compression $(N < 0)$, we have

$$\phi_1 = \frac{kL}{2}\cot\frac{kL}{2} \qquad \phi_2 = \frac{1}{12}\frac{k^2L^2}{(1-\phi_1)} \qquad \phi_3 = \frac{1}{4}\phi_1 + \frac{3}{4}\phi_2$$

$$\phi_4 = -\frac{1}{2}\phi_1 + \frac{3}{2}\phi_2 \qquad \phi_5 = \phi_1\phi_2$$

with

$$k = \frac{\pi}{L}\sqrt{\rho}$$

For axial tension $(N > 0)$, we have

$$\phi_1 = \frac{kL}{2}\coth\frac{kL}{2} \qquad \phi_2 = -\frac{1}{12}\frac{k^2L^2}{(1-\phi_1)} \qquad \phi_3 = \frac{1}{4}\phi_1 + \frac{3}{4}\phi_2$$

$$\phi_4 = -\frac{1}{2}\phi_1 + \frac{3}{2}\phi_2 \qquad \phi_5 = \phi_1\phi_2$$

with

$$k = \frac{\pi}{L}\sqrt{-\rho}$$

The parameter ρ is given by

$$\rho = -\frac{NL^2}{\pi^2 EI}$$

The equivalent nodal loads $\mathbf{f}_l^e$ stored in the variable fe are computed according to

$$\mathbf{f}_l^e = \mathbf{G}^T \bar{\mathbf{f}}_l^e$$

where

$$\bar{\mathbf{f}}_l^e = qL \begin{bmatrix} 0 & \dfrac{1}{2} & \dfrac{L}{12}\psi & 0 & \dfrac{1}{2} & -\dfrac{L}{12}\psi \end{bmatrix}^T$$
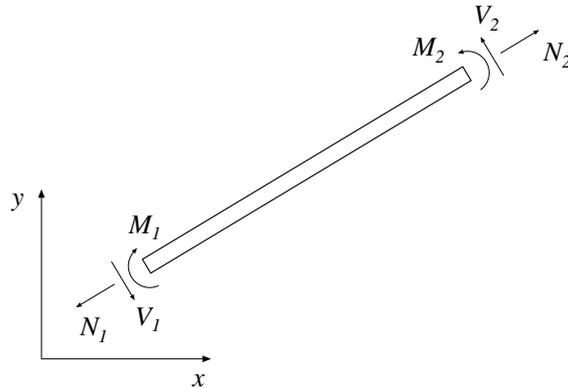
For an axial compressive force, we have

$$\psi = 6\left(\frac{2}{(kL)^2} - \frac{1 + \cos kL}{kL\sin kL}\right)$$

and for an axial tensile force

$$\psi = 6\left(\frac{1 + \cosh kL}{kL\sinh kL} - \frac{2}{(kL)^2}\right)$$

Purpose:

Compute section forces in a two dimensional nonlinear beam element.

Syntax:

es=beam2gs(ex,ey,ep,ed,N)
es=beam2gs(ex,ey,ep,ed,N,eq)

Description:

beam2gs computes the section forces at the ends of the nonlinear beam element beam2g.

The input variables ex, ey, and ep are defined in beam2e, and the variables N and eq in beam2g. The element displacements, stored in ed, are obtained by the function extract. If a distributed transversal load is applied to the element, the variable eq must be included.

The output variable

$$
\mathsf{es} = \left[\begin{array}{ccc} N_1 & V_1 & M_1 \\ N_2 & V_2 & M_2 \end{array} \right]
$$

contains the section forces at the ends of the beam.

Theory:

The section forces at the ends of the beam are obtained from the element force vector

$$\bar{\mathbf{P}} = [\ -N_1 \quad -V_1 \quad -M_1 \quad N_2 \ V_2 \ M_2\ ]^T$$

computed according to

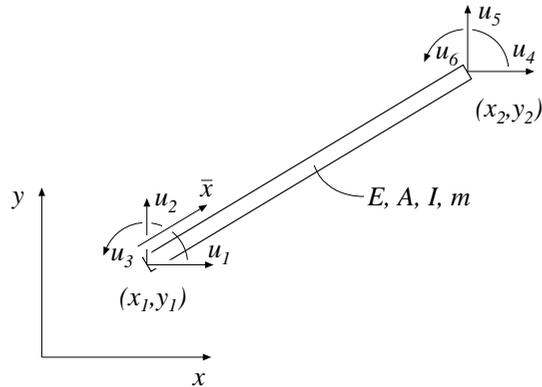$$\bar{\mathbf{P}} = \bar{\mathbf{K}}^e \ \mathbf{G} \ \mathbf{a}^e - \bar{\mathbf{f}}_l^e$$

The matrix $\mathbf{G}$ is described in beam2e. The matrix $\bar{\mathbf{K}}^e$ and the nodal displacements

$$\mathbf{a}^e = [\ u_1 \ u_2 \ u_3 \ u_4 \ u_5 \ u_6\ ]^T$$

are described in beam2g. Note that the transpose of $\mathbf{a}^e$ is stored in ed.

Purpose:

Compute element stiffness, mass and damping matrices for a two dimensional beam element.

Syntax:

[Ke,Me]=beam2d(ex,ey,ep)
[Ke,Me,Ce]=beam2d(ex,ey,ep)

Description:

beam2d provides the global element stiffness matrix Ke, the global element mass matrix Me, and the global element damping matrix Ce, for a two dimensional beam element.

The input variables ex and ey are described in beam2e, and

$$ \mathsf{ep} = [\; E \;\; A \;\; I \;\; m \;\; [\; a_0 \;\; a_1 \;] \;] $$

contains the modulus of elasticity E, the cross section area A, the moment of inertia I, the mass per unit length m, and the Raleigh damping coefficients a_0 and a_1. If a_0 and a_1 are omitted, the element damping matrix Ce is not computed.

Theory:

The element stiffness matrix $\mathbf{K}^e$, the element mass matrix $\mathbf{M}^e$ and the element damping matrix $\mathbf{C}^e$, stored in the variables Ke, Me and Ce, respectively, are computed according to

$$\mathbf{K}^e = \mathbf{G}^T\bar{\mathbf{K}}^e\mathbf{G} \qquad \mathbf{M}^e = \mathbf{G}^T\bar{\mathbf{M}}^e\mathbf{G} \qquad \mathbf{C}^e = \mathbf{G}^T\bar{\mathbf{C}}^e\mathbf{G}$$

where $\mathbf{G}$ and $\bar{\mathbf{K}}^e$ are described in beam2e.
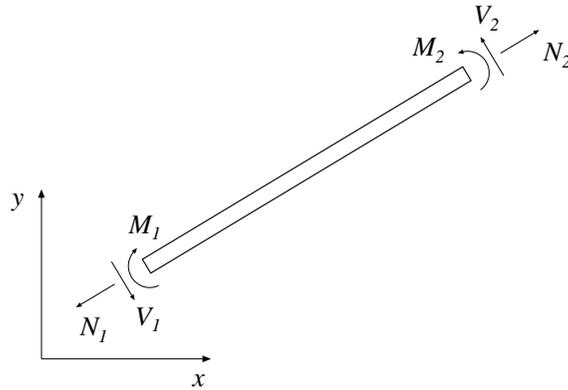
The matrix $\bar{\mathbf{M}}^e$ is given by

$$\bar{\mathbf{M}}^e = \frac{mL}{420}\begin{bmatrix} 140 & 0 & 0 & 70 & 0 & 0 \\ 0 & 156 & 22L & 0 & 54 & -13L \\ 0 & 22L & 4L^2 & 0 & 13L & -3L^2 \\ 70 & 0 & 0 & 140 & 0 & 0 \\ 0 & 54 & 13L & 0 & 156 & -22L \\ 0 & -13L & -3L^2 & 0 & -22L & 4L^2 \end{bmatrix}$$

and the matrix $\bar{\mathbf{C}}^e$ is computed by combining $\bar{\mathbf{K}}^e$ and $\bar{\mathbf{M}}^e$

$$\bar{\mathbf{C}}^e = a_0\bar{\mathbf{M}}^e + a_1\bar{\mathbf{K}}^e$$

Purpose:

Compute section forces for a two dimensional beam element in dynamic analysis.

Syntax:

es=beam2ds(ex,ey,ep,ed,ev,ea)

Description:

beam2ds computes the section forces at the ends of the dynamic beam element beam2d.

The input variables ex, ey, and ep are defined in beam2d. The element displacements, the element velocities, and the element accelerations, stored in ed, ev, and ea respectively, are obtained by the function extract.

The output variable

$$\text{es} = \left[\begin{array}{ccc} N_1 & V_1 & M_1 \\ N_2 & V_2 & M_2 \end{array} \right]$$

contains the section forces at the ends of the beam.

Theory:

The section forces at the ends of the beam are obtained from the element force vector

$$\bar{\mathbf{P}} = \begin{bmatrix} -N_1 & -V_1 & -M_1 & N_2 & V_2 & M_2 \end{bmatrix}^T$$

computed according to

$$\bar{\mathbf{P}} = \bar{\mathbf{K}}^e \, \mathbf{G} \, \mathbf{a}^e + \bar{\mathbf{C}}^e \, \mathbf{G} \, \dot{\mathbf{a}}^e + \bar{\mathbf{M}}^e \, \mathbf{G} \, \ddot{\mathbf{a}}^e$$

The matrices $\bar{\mathbf{K}}^e$ and $\mathbf{G}$ are described in beam2e, and the matrices $\bar{\mathbf{M}}^e$ and $\bar{\mathbf{C}}^e$ are described in beam2d. The nodal displacements

$$\mathbf{a}^e = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 & u_5 & u_6 \end{bmatrix}^T$$

shown in beam2d also define the directions of the nodal velocities

$$\dot{\mathbf{a}}^e = \begin{bmatrix} \dot{u}_1 & \dot{u}_2 & \dot{u}_3 & \dot{u}_4 & \dot{u}_5 & \dot{u}_6 \end{bmatrix}^T$$

and the nodal accelerations

$$\ddot{\mathbf{a}}^e = \begin{bmatrix} \ddot{u}_1 & \ddot{u}_2 & \ddot{u}_3 & \ddot{u}_4 & \ddot{u}_5 & \ddot{u}_6 \end{bmatrix}^T$$

Note that the transposes of $\mathbf{a}^e$, $\dot{\mathbf{a}}^e$, and $\ddot{\mathbf{a}}^e$ are stored in ed, ev, and ea respectively.

Purpose:

Compute element stiffness matrix for a three dimensional beam element.
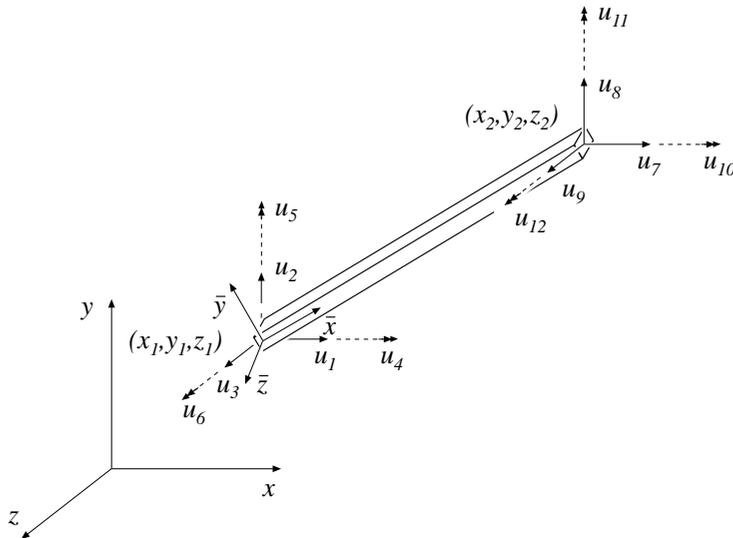
Syntax:

Ke=beam3e(ex,ey,ez,eo,ep)
[Ke,fe]=beam3e(ex,ey,ez,eo,ep,eq)

Description:

beam3e provides the global element stiffness matrix Ke for a three dimensional beam element.

The input variables

$$\mathsf{ex} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}$$
$$\mathsf{ey} = \begin{bmatrix} y_1 & y_2 \end{bmatrix} \qquad \mathsf{eo} = \begin{bmatrix} x_{\bar{z}} & y_{\bar{z}} & z_{\bar{z}} \end{bmatrix}$$
$$\mathsf{ez} = \begin{bmatrix} z_1 & z_2 \end{bmatrix}$$

supply the element nodal coordinates x_1, y_1, etc. as well as the direction of the local beam coordinate system $(\bar{x}, \bar{y}, \bar{z})$. By giving a global vector $(x_{\bar{z}}, y_{\bar{z}}, z_{\bar{z}})$ parallel with the positive local $\bar{z}$ axis of the beam, the local beam coordinate system is defined. The variable

$$\mathsf{ep} = \begin{bmatrix} E & G & A & I_{\bar{y}} & I_{\bar{z}} & K_v \end{bmatrix}$$

supplies the modulus of elasticity E, the shear modulus G, the cross section area A, the moment of inertia with respect to the $\bar{y}$ axis I_y, the moment of inertia with respect to the $\bar{z}$ axis I_z, and St Venant torsional stiffness K_v.

The element load vector fe can also be computed if uniformly distributed loads are applied to the element. The optional input variable

$$\text{eq} = \begin{bmatrix} q_{\bar{x}} & q_{\bar{y}} & q_{\bar{z}} & q_{\bar{\omega}} \end{bmatrix}$$

then contains the distributed loads. The positive directions of $q_{\bar{x}}$, $q_{\bar{y}}$, and $q_{\bar{z}}$ follow the local beam coordinate system. The distributed torque $q_{\bar{\omega}}$ is positive if directed in the local $\bar{x}$-direction, i.e. from local $\bar{y}$ to local $\bar{z}$. All the loads are per unit length.

Theory:

The element stiffness matrix $\mathbf{K}^e$ is computed according to

$$\mathbf{K}^e = \mathbf{G}^T \bar{\mathbf{K}}^e \mathbf{G}$$

where

$$\bar{\mathbf{K}}^e = \begin{bmatrix}
k_1 & 0 & 0 & 0 & 0 & 0 & -k_1 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{12EI_{\bar{z}}}{L^3} & 0 & 0 & 0 & \frac{6EI_{\bar{z}}}{L^2} & 0 & -\frac{12EI_{\bar{z}}}{L^3} & 0 & 0 & 0 & \frac{6EI_{\bar{z}}}{L^2} \\
0 & 0 & \frac{12EI_{\bar{y}}}{L^3} & 0 & -\frac{6EI_{\bar{y}}}{L^2} & 0 & 0 & 0 & -\frac{12EI_{\bar{y}}}{L^3} & 0 & -\frac{6EI_{\bar{y}}}{L^2} & 0 \\
0 & 0 & 0 & k_2 & 0 & 0 & 0 & 0 & 0 & -k_2 & 0 & 0 \\
0 & 0 & -\frac{6EI_{\bar{y}}}{L^2} & 0 & \frac{4EI_{\bar{y}}}{L} & 0 & 0 & 0 & \frac{6EI_{\bar{y}}}{L^2} & 0 & \frac{2EI_{\bar{y}}}{L} & 0 \\
0 & \frac{6EI_{\bar{z}}}{L^2} & 0 & 0 & 0 & \frac{4EI_{\bar{z}}}{L} & 0 & -\frac{6EI_{\bar{z}}}{L^2} & 0 & 0 & 0 & \frac{2EI_{\bar{z}}}{L} \\
-k_1 & 0 & 0 & 0 & 0 & 0 & k_1 & 0 & 0 & 0 & 0 & 0 \\
0 & -\frac{12EI_{\bar{z}}}{L^3} & 0 & 0 & 0 & -\frac{6EI_{\bar{z}}}{L^2} & 0 & \frac{12EI_{\bar{z}}}{L^3} & 0 & 0 & 0 & -\frac{6EI_{\bar{z}}}{L^2} \\
0 & 0 & -\frac{12EI_{\bar{y}}}{L^3} & 0 & \frac{6EI_{\bar{y}}}{L^2} & 0 & 0 & 0 & \frac{12EI_{\bar{y}}}{L^3} & 0 & \frac{6EI_{\bar{y}}}{L^2} & 0 \\
0 & 0 & 0 & -k_2 & 0 & 0 & 0 & 0 & 0 & k_2 & 0 & 0 \\
0 & 0 & -\frac{6EI_{\bar{y}}}{L^2} & 0 & \frac{2EI_{\bar{y}}}{L} & 0 & 0 & 0 & \frac{6EI_{\bar{y}}}{L^2} & 0 & \frac{4EI_{\bar{y}}}{L} & 0 \\
0 & \frac{6EI_{\bar{z}}}{L^2} & 0 & 0 & 0 & \frac{2EI_{\bar{z}}}{L} & 0 & -\frac{6EI_{\bar{z}}}{L^2} & 0 & 0 & 0 & \frac{4EI_{\bar{z}}}{L}
\end{bmatrix}$$

in which $k_1 = \frac{EA}{L}$ and $k_2 = \frac{GK_v}{L}$, and where

$$\mathbf{G} = \begin{bmatrix}
n_{x\bar{x}} & n_{y\bar{x}} & n_{z\bar{x}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{x\bar{y}} & n_{y\bar{y}} & n_{z\bar{y}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{x\bar{z}} & n_{y\bar{z}} & n_{z\bar{z}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & n_{x\bar{x}} & n_{y\bar{x}} & n_{z\bar{x}} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & n_{x\bar{y}} & n_{y\bar{y}} & n_{z\bar{y}} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & n_{x\bar{z}} & n_{y\bar{z}} & n_{z\bar{z}} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & n_{x\bar{x}} & n_{y\bar{x}} & n_{z\bar{x}} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & n_{x\bar{y}} & n_{y\bar{y}} & n_{z\bar{y}} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & n_{x\bar{z}} & n_{y\bar{z}} & n_{z\bar{z}} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n_{x\bar{x}} & n_{y\bar{x}} & n_{z\bar{x}} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n_{x\bar{y}} & n_{y\bar{y}} & n_{z\bar{y}} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n_{x\bar{z}} & n_{y\bar{z}} & n_{z\bar{z}}
\end{bmatrix}$$

The element length L is computed according to

$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

In the transformation matrix $\mathbf{G}$, $n_{x\bar{x}}$ specifies the cosine of the angle between the x axis and $\bar{x}$ axis, and so on.

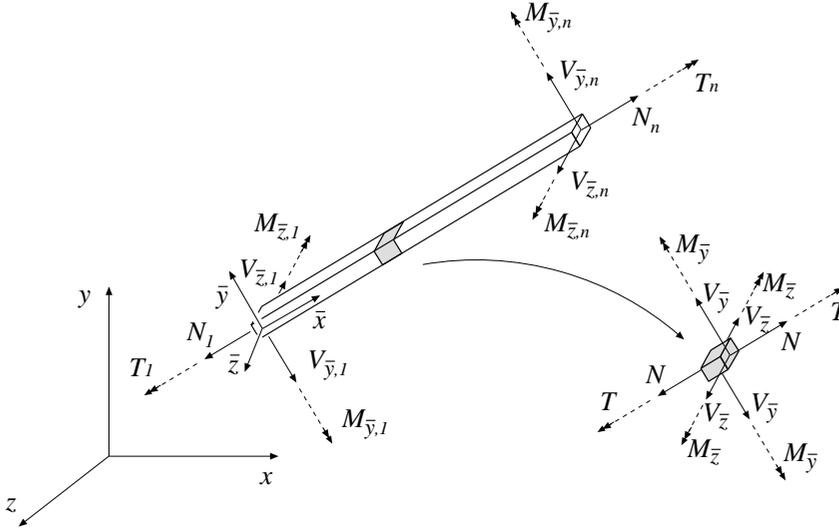The element load vector $\mathbf{f}_l^e$, stored in fe, is computed according to

$$\mathbf{f}_l^e = \mathbf{G}^T \bar{\mathbf{f}}_l^e$$

where

$$\bar{\mathbf{f}}_l^e = \begin{bmatrix} \dfrac{q_{\bar{x}}L}{2} \\[1.5ex] \dfrac{q_{\bar{y}}L}{2} \\[1.5ex] \dfrac{q_{\bar{z}}L}{2} \\[1.5ex] \dfrac{q_{\bar{\omega}}L}{2} \\[1.5ex] -\dfrac{q_{\bar{z}}L^2}{12} \\[1.5ex] \dfrac{q_{\bar{y}}L^2}{12} \\[1.5ex] \dfrac{q_{\bar{x}}L}{2} \\[1.5ex] \dfrac{q_{\bar{y}}L}{2} \\[1.5ex] \dfrac{q_{\bar{z}}L}{2} \\[1.5ex] \dfrac{q_{\bar{\omega}}L}{2} \\[1.5ex] \dfrac{q_{\bar{z}}L^2}{12} \\[1.5ex] -\dfrac{q_{\bar{y}}L^2}{12} \end{bmatrix}$$

Purpose:

Compute section forces in a three dimensional beam element.

Syntax:

es=beam3s(ex,ey,ez,eo,ep,ed)
es=beam3s(ex,ey,ez,eo,ep,ed,eq)
[es,edi,eci]=beam3s(ex,ey,ez,eo,ep,ed,eq,n)

Description:

beam3s computes the section forces and displacements in local directions along the beam element beam3e.

The input variables ex, ey, ez, eo, and ep are defined in beam3e, and the element displacements, stored in ed, are obtained by the function extract. If distributed loads are applied to the element, the variable eq must be included. The number of evaluation points for section forces and displacements are determined by n. If n is omitted, only the ends of the beam are evaluated.

The output variables

$$\mathsf{es} = [\, \mathbf{N}\ \ \mathbf{V}_{\bar{y}}\ \ \mathbf{V}_{\bar{z}}\ \ \mathbf{T}\ \ \mathbf{M}_{\bar{y}}\ \ \mathbf{M}_{\bar{z}} \,] \qquad \mathsf{edi} = [\, \bar{\mathbf{u}}\ \ \bar{\mathbf{v}}\ \ \bar{\mathbf{w}}\ \ \bar{\boldsymbol{\varphi}} \,] \qquad \mathsf{eci} = [\, \bar{\mathbf{x}} \,]$$

consist of column matrices that contain the section forces, the displacements, and the evaluation points on the local $\bar{x}$-axis. The explicit matrix expressions are

$$
\mathsf{es} = \begin{bmatrix}
N_1 & V_{\bar{y}1} & V_{\bar{z}1} & T & M_{\bar{y}1} & M_{\bar{z}1} \\
N_2 & V_{\bar{y}2} & V_{\bar{z}2} & T & M_{\bar{y}2} & M_{\bar{z}2} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
N_n & V_{\bar{y}n} & V_{\bar{z}n} & T & M_{\bar{y}n} & M_{\bar{z}n}
\end{bmatrix}
$$

$$
\mathsf{edi} = \begin{bmatrix} \bar{u}_1 & \bar{v}_1 & \bar{w}_1 & \bar{\varphi}_1 \\ \bar{u}_2 & \bar{v}_2 & \bar{w}_2 & \bar{\varphi}_2 \\ \vdots & \vdots & \vdots & \vdots \\ \bar{u}_n & \bar{v}_n & \bar{w}_n & \bar{\varphi}_n \end{bmatrix} \qquad \mathsf{eci} = \begin{bmatrix} 0 \\ \bar{x}_2 \\ \vdots \\ \bar{x}_{n-1} \\ L \end{bmatrix}
$$

where L is the length of the beam element.

Theory:

The evaluation of the section forces is based on the solutions of the basic equations

$$
EA\frac{d^2\bar{u}}{d\bar{x}^2} + q_{\bar{x}} = 0 \qquad\qquad EI_z\frac{d^4\bar{v}}{d\bar{x}^4} - q_{\bar{y}} = 0
$$

$$
EI_y\frac{d^4\bar{w}}{d\bar{x}^4} - q_{\bar{z}} = 0 \qquad\qquad GK_v\frac{d^2\bar{\varphi}}{d\bar{x}^2} + q_{\bar{\omega}} = 0
$$

From these equations, the displacements along the beam element are obtained as the sum of the homogeneous and the particular solutions

$$
\mathbf{u} = \begin{bmatrix} \bar{u}(\bar{x}) \\ \bar{v}(\bar{x}) \\ \bar{w}(\bar{x}) \\ \bar{\varphi}(\bar{x}) \end{bmatrix} = \mathbf{u}_h + \mathbf{u}_p
$$

where

$$
\mathbf{u}_h = \bar{\mathbf{N}}\,\mathbf{C}^{-1}\,\mathbf{G}\,\mathbf{a^e} \qquad\qquad \mathbf{u_p} = \begin{bmatrix} \bar{u}_p(\bar{x}) \\ \bar{v}_p(\bar{x}) \\ \bar{w}_p(\bar{x}) \\ \bar{\varphi}_p(\bar{x}) \end{bmatrix} = \begin{bmatrix} \dfrac{q_{\bar{x}}L\bar{x}}{2EA}(1 - \dfrac{\bar{x}}{L}) \\[2mm] \dfrac{q_{\bar{y}}L^2\bar{x}^2}{24EI_z}(1 - \dfrac{\bar{x}}{L})^2 \\[2mm] \dfrac{q_{\bar{z}}L^2\bar{x}^2}{24EI_y}(1 - \dfrac{\bar{x}}{L})^2 \\[2mm] \dfrac{q_{\bar{\omega}}L\bar{x}}{2GK_v}(1 - \dfrac{\bar{x}}{L}) \end{bmatrix}
$$

and

$$
\bar{\mathbf{N}} = \begin{bmatrix} 1 & \bar{x} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \bar{x} & \bar{x}^2 & \bar{x}^3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \bar{x} & \bar{x}^2 & \bar{x}^3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \bar{x} \end{bmatrix}
$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & L & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & L & L^2 & L^3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & L & L^2 & L^3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & L \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2L & 3L^2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2L & 3L^2 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{a}^e = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{12} \end{bmatrix}$$

The transformation matrix $\mathbf{G}^e$ and nodal displacements $\mathbf{a}^e$ are described in beam3e. Note that the transpose of $\mathbf{a}^e$ is stored in ed.

Finally the section forces are obtained from

$$N = EA\frac{d\bar{u}}{d\bar{x}} \qquad V_{\bar{y}} = -EI_z\frac{d^3\bar{v}}{d\bar{x}^3} \qquad V_{\bar{z}} = -EI_y\frac{d^3\bar{w}}{d\bar{x}^3}$$

$$T = GK_v\frac{d\bar{\varphi}}{d\bar{x}} \qquad M_{\bar{y}} = -EI_y\frac{d^2\bar{w}}{d\bar{x}^2} \qquad M_{\bar{z}} = EI_z\frac{d^2\bar{v}}{d\bar{x}^2}$$

Examples:

Section forces or element displacements can easily be plotted. The bending moment $M_{\bar{y}}$ along the beam is plotted by

```
>> plot(eci,es(:,5))
```

5.7 Plate element

Only one plate element is currently available, a rectangular 12 dof element. The element presumes a linear elastic material which can be isotropic or anisotropic.
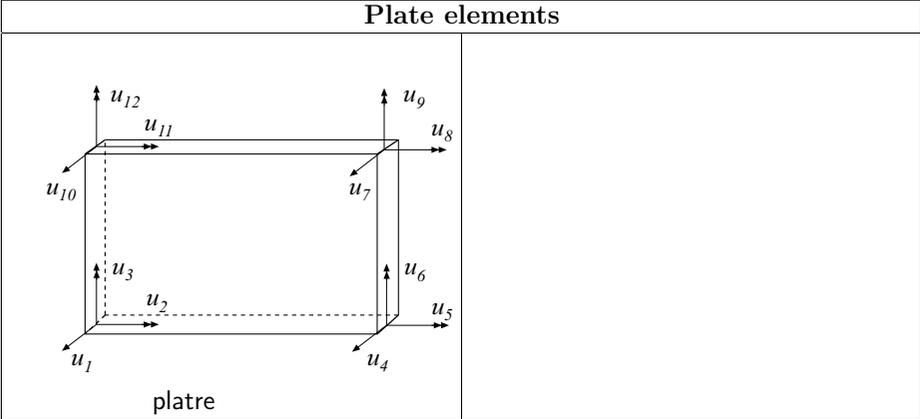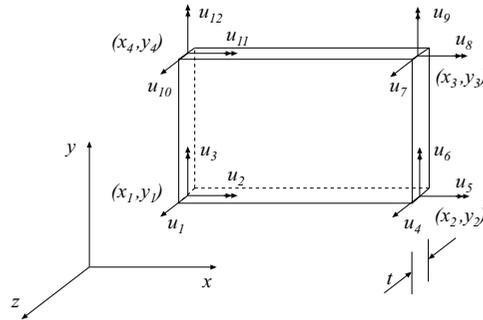
Plate elements	
platre	

Plate functions	
platre	Compute element matrices
platrs	Compute section forces

Purpose:

Compute element stiffness matrix for a rectangular plate element.

Syntax:

Ke=platre(ex,ey,ep,D)
[Ke,fe]=platre(ex,ey,ep,D,eq)

Description:

platre provides an element stiffness matrix Ke, and an element load vector fe, for a rectangular plate element. This element can only be used if the element edges are parallel to the coordinate axis.

The element nodal coordinates x_1, y_1, x_2 etc. are supplied to the function by ex and ey, the element thickness t by ep, and the material properties by the constitutive matrix D. Any arbitrary D-matrix with dimensions (3×3) and valid for plane stress may be given. For an isotropic elastic material the constitutive matrix can be formed by the function hooke, see Section 4.

$$
\begin{aligned}
\text{ex} &= \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} \\
\text{ey} &= \begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix}
\end{aligned}
\qquad
\text{ep} = \begin{bmatrix} t \end{bmatrix}
\qquad
\text{D} = \begin{bmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{bmatrix}
$$

If a uniformly distributed load is applied to the element, the element load vector fe is computed. The input variable

$$
\text{eq} = \begin{bmatrix} q_z \end{bmatrix}
$$

then contains the load q_z per unit area in the z-direction.

Theory:

The element stiffness matrix $\mathbf{K}^e$ and the element load vector $\mathbf{f}_l^e$, stored in Ke and fe respectively, are computed according to

$$\mathbf{K}^e = (\mathbf{C}^{-1})^T \int_A \bar{\mathbf{B}}^T \, \tilde{\mathbf{D}} \, \bar{\mathbf{B}} \, dA \, \mathbf{C}^{-1}$$

$$\mathbf{f}_l^e = (\mathbf{C}^{-1})^T \int_A \bar{\mathbf{N}}^T \, q_z \, dA$$

where the constitutive matrix

$$\tilde{\mathbf{D}} = \frac{t^3}{12} \, \mathbf{D}$$

and where $\mathbf{D}$ is defined by D.

The evaluation of the integrals for the rectangular plate element is based on the displacement approximation $w(x, y)$ and is expressed in terms of the nodal variables $u_1, u_2, \dots , u_{12}$ as

$$w(x, y) = \mathbf{N}^e \, \mathbf{a}^e = \bar{\mathbf{N}} \, \mathbf{C}^{-1} \, \mathbf{a}^e$$

where

$$\bar{\mathbf{N}} = \begin{bmatrix} 1 & x & y & x^2 & xy & y^2 & x^3 & x^2y & xy^2 & y^3 & x^3y & xy^3 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix}
1 & -a & -b & a^2 & ab & b^2 & -a^3 & -a^2b & -ab^2 & -b^3 & a^3b & ab^3 \\
0 & 0 & 1 & 0 & -a & -2b & 0 & a^2 & 2ab & 3b^2 & -a^3 & -3ab^2 \\
0 & -1 & 0 & 2a & b & 0 & -3a^2 & -2ab & -b^2 & 0 & 3a^2b & b^3 \\
1 & a & -b & a^2 & -ab & b^2 & a^3 & -a^2b & ab^2 & -b^3 & -a^3b & -ab^3 \\
0 & 0 & 1 & 0 & a & -2b & 0 & a^2 & -2ab & 3b^2 & a^3 & 3ab^2 \\
0 & -1 & 0 & -2a & b & 0 & -3a^2 & 2ab & -b^2 & 0 & 3a^2b & b^3 \\
1 & a & b & a^2 & ab & b^2 & a^3 & a^2b & ab^2 & b^3 & a^3b & ab^3 \\
0 & 0 & 1 & 0 & a & 2b & 0 & a^2 & 2ab & 3b^2 & a^3 & 3ab^2 \\
0 & -1 & 0 & -2a & -b & 0 & -3a^2 & -2ab & -b^2 & 0 & -3a^2b & -b^3 \\
1 & -a & b & a^2 & -ab & b^2 & -a^3 & a^2b & -ab^2 & b^3 & -a^3b & -ab^3 \\
0 & 0 & 1 & 0 & -a & 2b & 0 & a^2 & -2ab & 3b^2 & -a^3 & -3ab^2 \\
0 & -1 & 0 & 2a & -b & 0 & -3a^2 & 2ab & -b^2 & 0 & -3a^2b & -b^3
\end{bmatrix}$$

$$\mathbf{a}^e = \begin{bmatrix} u_1 & u_2 & \dots & u_{12} \end{bmatrix}^T$$

and where

$$a = \frac{1}{2}(x_3 - x_1) \quad \text{and} \quad b = \frac{1}{2}(y_3 - y_1)$$

The matrix $\bar{\mathbf{B}}$ is obtained as

$$\bar{\mathbf{B}} = \overset{*}{\nabla}\bar{\mathbf{N}} = \begin{bmatrix} 0 & 0 & 0 & 2 & 0 & 0 & 6x & 2y & 0 & 0 & 6xy & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 2x & 6y & 0 & 6xy \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 4x & 4y & 0 & 6x^2 & 6y^2 \end{bmatrix}$$

where

$$\overset{*}{\nabla} = \begin{bmatrix} \dfrac{\partial^2}{\partial x^2} \\[2ex] \dfrac{\partial^2}{\partial y^2} \\[2ex] 2\dfrac{\partial^2}{\partial x \partial y} \end{bmatrix}$$

Evaluation of the integrals for the rectangular plate element is done analytically. Computation of the integrals for the element load vector $\mathbf{f}_l^e$ yields
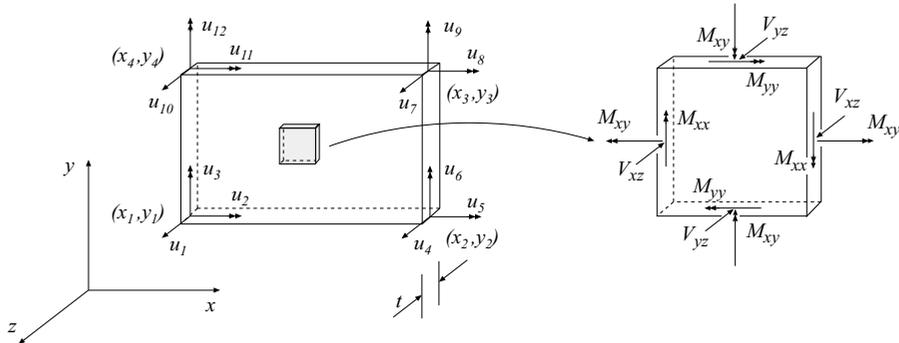
$$\mathbf{f}_l^e = q_z L_x L_y \left[\frac{1}{4} \ \frac{L_y}{24} \ -\frac{L_x}{24} \ \frac{1}{4} \ \frac{L_y}{24} \ \frac{L_x}{24} \ \frac{1}{4} \ -\frac{L_y}{24} \ \frac{L_x}{24} \ \frac{1}{4} \ -\frac{L_y}{24} \ -\frac{L_x}{24} \right]^T$$

where

$$L_x = x_3 - x_1 \quad \text{and} \quad L_y = y_3 - y_1$$

Purpose:

Compute section forces in a rectangular plate element.

Syntax:

[es,et]=platrs(ex,ey,ep,D,ed)

Description:

platrs computes the section forces es and the curvature matrix et in a rectangular plate element. The section forces and the curvatures are computed at the center of the element.

The input variables ex, ey, ep and D are defined in platre. The vector ed contains the nodal displacements $\mathbf{a}^e$ of the element and is obtained by the function extract as

$$\mathsf{ed} = (\mathbf{a}^e)^T = [\; u_1 \;\; u_2 \;\; ... \;\; u_{12} \;]$$

The output variables

$$\mathsf{es} = \left[\; \mathbf{M}^T \; \mathbf{V}^T \;\right] = [\; M_{xx} \; M_{yy} \; M_{xy} \; V_{xz} \; V_{yz} \;]$$

$$\mathsf{et} = \boldsymbol{\kappa}^T = [\; \kappa_{xx} \; \kappa_{yy} \; \kappa_{xy} \;]$$

contain the section forces and curvatures in global directions.

Theory:

The curvatures and the section forces are computed according to

$$\boldsymbol{\kappa} = \begin{bmatrix} \kappa_{xx} \\ \kappa_{yy} \\ \kappa_{xy} \end{bmatrix} = \bar{\mathbf{B}} \, \mathbf{C}^{-1} \, \mathbf{a}^e$$

$$\mathbf{M} = \begin{bmatrix} M_{xx} \\ M_{yy} \\ M_{xy} \end{bmatrix} = \tilde{\mathbf{D}} \, \boldsymbol{\kappa}$$

$$\mathbf{V} = \begin{bmatrix} V_{xz} \\ V_{yz} \end{bmatrix} = \tilde{\nabla} \, \mathbf{M}$$

where the matrices $\tilde{\mathbf{D}}$, $\bar{\mathbf{B}}$, $\mathbf{C}$ and $\mathbf{a}^e$ are described in platre, and where

$$\tilde{\nabla} = \begin{bmatrix} \dfrac{\partial}{\partial x} & 0 & \dfrac{\partial}{\partial y} \\ 0 & \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} \end{bmatrix}$$

6 System functions

6.1 Introduction

The group of system functions comprises functions for the setting up, solving, and elimination of systems of equations. The functions are separated in two groups:

Static system functions
Dynamic system functions

Static system functions concern the linear system of equations

$$\mathbf{K}\mathbf{a} = \mathbf{f}$$

where $\mathbf{K}$ is the global stiffness matrix and $\mathbf{f}$ is the global load vector. Often used static system functions are assem and solveq. The function assem assembles the global stiffness matrix and solveq computes the global displacement vector $\mathbf{a}$ considering the boundary conditions. It should be noted that $\mathbf{K}$, $\mathbf{f}$, and $\mathbf{a}$ also represent analogous quantities in systems others than structural mechanical systems. For example, in a heat flow problem $\mathbf{K}$ represents the conductivity matrix, $\mathbf{f}$ the heat flow, and $\mathbf{a}$ the temperature.

Dynamic system functions are related to different aspects of linear dynamic systems of coupled ordinary differential equations according to

$$\mathbf{C}\,\dot{\mathbf{d}} + \mathbf{K}\,\mathbf{d} = \mathbf{f}$$

for first-order systems and

$$\mathbf{M}\ddot{\mathbf{d}} + \mathbf{C}\dot{\mathbf{d}} + \mathbf{K}\mathbf{d} = \mathbf{f}$$

for second-order systems. First-order systems occur typically in transient heat conduction and second-order systems occur in structural dynamics.

6.2 Static system functions

The group of static system functions comprises functions for setting up and solving the global system of equations. It also contains a function for eigenvalue analysis, a function for static condensation, a function for extraction of element displacements from the global displacement vector and a function for extraction of element coordinates.

The following functions are available for static analysis:

Static system functions	
assem	Assemble element matrices
coordxtr	Extract element coordinates from a global coordinate matrix.
eigen	Solve a generalized eigenvalue problem
extract	Extract values from a global vector
insert	Assemble element internal force vector
solveq	Solve a system of equations
statcon	Perform static condensation

Purpose:

Assemble element matrices.

$$
\begin{array}{c}
\begin{array}{cc} i & j \end{array} \\
\left[\begin{array}{cc} k_{ii}^{e} & k_{ij}^{e} \\ k_{ji}^{e} & k_{jj}^{e} \end{array}\right]\begin{array}{c} i \\ j \end{array} \\
\mathbf{K}^{e} \\
i = dof_i \\
j = dof_j
\end{array}
\qquad \longrightarrow \qquad
\begin{array}{c}
\begin{array}{cc} \quad\; i & \quad\;\; j \end{array} \\
\left[\begin{array}{cccc}
k_{11} & k_{12} & \vdots & \vdots \\
k_{21} & & \vdots & \vdots \\
\cdots\cdots k_{ii}+k_{ii}^{e} & k_{ij}+k_{ij}^{e}\cdots\cdots & & \\
\cdots\cdots k_{ji}+k_{ji}^{e} & k_{jj}+k_{jj}^{e}\cdots\cdots & & \\
& & \vdots & \vdots \\
& & & k_{nn}
\end{array}\right]\begin{array}{c} \\ \\ i \\ j \\ \\ \\ \end{array} \\
\mathbf{K}
\end{array}
$$

Syntax:

K=assem(edof,K,Ke)
[K,f]=assem(edof,K,Ke,f,fe)

Description:

assem adds the element stiffness matrix $\mathbf{K}^e$, stored in Ke, to the structure stiffness matrix $\mathbf{K}$, stored in K, according to the topology matrix edof.

The element topology matrix edof is defined as

$$
\mathsf{edof} = [el \quad \underbrace{dof_1 \quad dof_2 \quad \ldots \quad dof_{ned}}_{\text{global dof.}}]
$$

where the first column contains the element number, and the columns 2 to $(ned + 1)$ contain the corresponding global degrees of freedom (ned = number of element degrees of freedom).
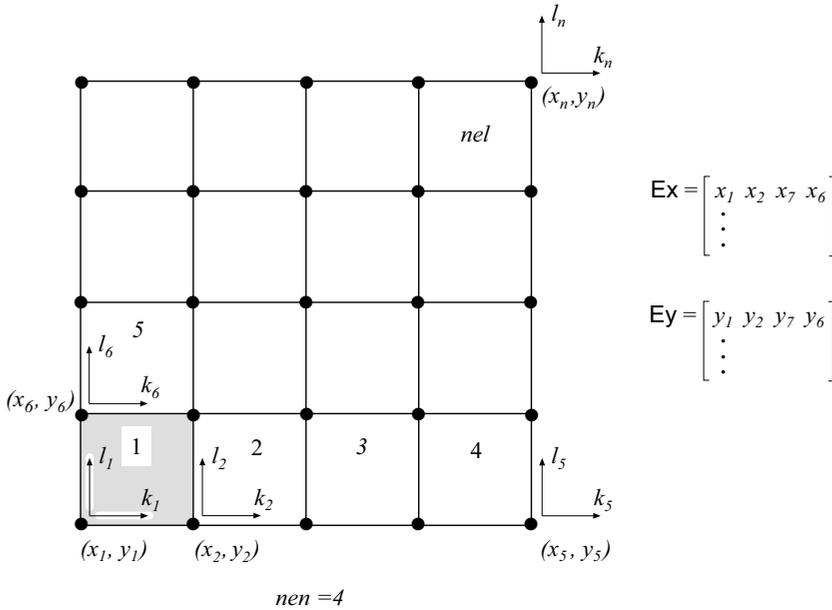
In the case where the matrix $\mathbf{K}^e$ is identical for several elements, assembling of these can be carried out simultaneously. Each row in Edof then represents one element, i.e. nel is the total number of considered elements.

$$
\mathsf{Edof} = \left.\left[\begin{array}{ccccccc}
el_1 & & dof_1 & dof_2 & . & . & . & dof_{ned} \\
el_2 & & dof_1 & dof_2 & . & . & . & dof_{ned} \\
\vdots & & \vdots & \vdots & & & & \vdots \\
el_{nel} & & dof_1 & dof_2 & . & . & . & dof_{ned}
\end{array}\right]\right\} one\ row\ for\ each\ element
$$

If fe and f are given in the function, the element load vector $\mathbf{f}^e$ is also added to the global load vector $\mathbf{f}$.

Purpose:

Extract element coordinates from a global coordinate matrix.

$$\mathsf{Ex} = \begin{bmatrix} x_1 & x_2 & x_7 & x_6 \\ & \vdots & \end{bmatrix}$$

$$\mathsf{Ey} = \begin{bmatrix} y_1 & y_2 & y_7 & y_6 \\ & \vdots & \end{bmatrix}$$

$$nen = 4$$

Syntax:

[Ex,Ey,Ez]=coordxtr(Edof,Coord,Dof,nen)

Description:

coordxtr extracts element nodal coordinates from the global coordinate matrix Coord for elements with equal numbers of element nodes and dof's.

Input variables are the element topology matrix Edof, defined in assem, the global coordinate matrix Coord, the global topology matrix Dof, and the number of element nodes nen in each element.

$$\mathsf{Coord} = \begin{bmatrix} x_1 & y_1 & [z_1] \\ x_2 & y_2 & [z_2] \\ x_3 & y_3 & [z_3] \\ \vdots & \vdots & \vdots \\ x_n & y_n & [z_n] \end{bmatrix} \qquad \mathsf{Dof} = \begin{bmatrix} k_1 & l_1 & \dots & m_1 \\ k_2 & l_2 & \dots & m_2 \\ k_3 & l_3 & \dots & m_3 \\ \vdots & \vdots & \dots & \vdots \\ k_n & l_n & \dots & m_n \end{bmatrix} \qquad \mathsf{nen} = [\, nen \,]$$

The nodal coordinates defined in row i of Coord correspond to the degrees of freedom of row i in Dof. The components k_i, l_i and m_i define the degrees of freedom of node i, and n is the number of global nodes for the considered part of the FE-model.

The output variables Ex, Ey, and Ez are matrices defined according to

$$
\mathsf{Ex} = \begin{bmatrix}
x_1{}^1 & x_2{}^1 & x_3{}^1 & ... & x_{nen}{}^1 \\
x_1{}^2 & x_2{}^2 & x_3{}^2 & ... & x_{nen}{}^2 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
x_1{}^{nel} & x_2{}^{nel} & x_3{}^{nel} & ... & x_{nen}{}^{nel}
\end{bmatrix}
$$

where row i gives the x-coordinates of the element defined in row i of Edof, and where nel is the number of considered elements.

The element coordinate data extracted by the function coordxtr can be used for plotting purposes and to create input data for the element stiffness functions.

Purpose:

Solve the generalized eigenvalue problem.

Syntax:

L=eigen(K,M)
L=eigen(K,M,b)
[L,X]=eigen(K,M)
[L,X]=eigen(K,M,b)

Description:

eigen solves the eigenvalue problem

$$\mid \mathsf{K} - \lambda \mathsf{M} \mid = 0$$

where K and M are square matrices. The eigenvalues λ are stored in the vector L and the corresponding eigenvectors in the matrix X.

If certain rows and columns in matrices K and M are to be eliminated in computing the eigenvalues, b must be given in the function. The rows (and columns) that are to be eliminated are described in the vector b defined as

$$\mathsf{b} = \left[\begin{array}{c} dof_1 \\ dof_2 \\ \vdots \\ dof_{nb} \end{array} \right]$$

The computed eigenvalues are given in order ranging from the smallest to the largest. The eigenvectors are normalized in order that

$$\mathsf{X}^T \mathsf{M} \mathsf{X} = \mathbf{I}$$

where $\mathbf{I}$ is the identity matrix.

Purpose:

Extract element nodal quantities from a global solution vector.

$$
\begin{bmatrix} \vdots \\ a_i \\ a_j \\ \vdots \\ a_m \\ a_n \\ \vdots \end{bmatrix}
\longrightarrow
\begin{bmatrix} a_i \\ a_j \\ a_m \\ a_n \end{bmatrix}
=
\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}
\qquad
\begin{aligned}
\mathsf{edof} &= [\,eln \quad i \quad j \quad m \quad n\,] \\
\mathsf{ed} &= [\,u_1 \quad u_2 \quad u_3 \quad u_4\,]
\end{aligned}
$$

Syntax:

ed=extract(edof,a)

Description:

extract extracts element displacements or corresponding quantities $\mathbf{a}^e$ from the global solution vector $\mathbf{a}$, stored in a.

Input variables are the element topology matrix edof, defined in assem, and the global solution vector a.

The output variable

$$\mathsf{ed} = (\mathbf{a}^e)^T$$

contains the element displacement vector.

If Edof contains more than one element, Ed will be a matrix

$$
\mathsf{Ed} =
\begin{bmatrix}
(\mathbf{a}^e)_1^T \\
(\mathbf{a}^e)_2^T \\
\vdots \\
(\mathbf{a}^e)_{nel}^T
\end{bmatrix}
$$

where row i gives the element displacements for the element defined in row i of Edof, and nel is the total number of considered elements.

Example:

For the two dimensional beam element, the extract function will extract six nodal displacements for each element given in Edof, and create a matrix Ed of size $(nel \times 6)$.

$$
\mathsf{Ed} = \begin{bmatrix}
u_1 & u_2 & u_3 & u_4 & u_5 & u_6 \\
u_1 & u_2 & u_3 & u_4 & u_5 & u_6 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
u_1 & u_2 & u_3 & u_4 & u_5 & u_6
\end{bmatrix}
$$

Purpose:

Assemble internal element forces in a global force vector.

$$
\begin{bmatrix} f_i^e \\ f_j^e \end{bmatrix} \longrightarrow
\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_i + f_i^e \\ f_j + f_j^e \\ \vdots \\ f_n \end{bmatrix}
$$

$$\mathbf{f}^e \qquad\qquad \mathbf{f}$$

$$i = dof_i$$
$$j = dof_j$$

Syntax:

f=insert(edof,f,ef)

Description:

insert adds the internal element load vector $\mathbf{f}_i^e$, stored in ef, to the global internal force vector $\mathbf{f}$, stored in f, according to the topology matrix edof. The function is for use in nonlinear analysis.

The element topology matrix edof is defined in assem. The vector f is the global internal force vector, and the vector ef is the internal element force vector computed from the element stresses, see for example plani4f.

Purpose:

Solve equation system.

Syntax:

a=solveq(K,f)
a=solveq(K,f,bc)
[a,r]=solveq(K,f,bc)

Description:

solveq solves the equation system

$$K\ a = f$$

where K is a matrix and a and f are vectors.

The matrix K and the vector f must be predefined. The solution of the system of equations is stored in a vector a which is created by the function.

If some values of a are to be prescribed, the row number and the corresponding values are given in the boundary condition matrix

$$bc = \begin{bmatrix} dof_1 & u_1 \\ dof_2 & u_2 \\ \vdots & \vdots \\ dof_{nbc} & u_{nbc} \end{bmatrix}$$

where the first column contains the row numbers and the second column the corresponding prescribed values.

If r is given in the function, support forces are computed according to

$$r = K\ a - f$$

Purpose:

Reduce system of equations by static condensation.

Syntax:

[K1,f1]=statcon(K,f,b)

Description:

statcon reduces a system of equations

$$K \, a = f$$

by static condensation.

The degrees of freedom to be eliminated are supplied to the function by the vector

$$b = \begin{bmatrix} dof_1 \\ dof_2 \\ \vdots \\ dof_{nb} \end{bmatrix}$$

where each row in b contains one degree of freedom to be eliminated.

The elimination gives the reduced system of equations

$$K_1 \, a_1 = f_1$$

where K_1 and f_1 are stored in K1 and f1 respectively.

6.3   Dynamic system functions

The group of system functions comprises functions for solving linear dynamic systems by time stepping or modal analysis, functions for frequency domain analysis, etc.

Dynamic system functions	
dyna2	Solve a set of uncoupled second-order differential equations
dyna2f	Solve a set of uncoupled second-order differential equations in the frequency domain
fft	Fast Fourier transform
freqresp	Compute frequency response
gfunc	Linear interpolation between equally spaced points
ifft	Inverse Fast Fourier transform
ritz	Compute approximative eigenvalues and eigenvectors by the Lanczos method
spectra	Compute seismic response spectra
step1	Carry out step-by-step integration in first-order systems
step2	Carry out step-by-step integration in second-order systems
sweep	Compute frequency response function

Note: Eigenvalue analysis is performed by using the function eigen; see static system functions.

Purpose:

Compute the dynamic solution to a set of uncoupled second-order differential equations.

Syntax:

X=dyna2(w2,xi,f,g,dt)

Description:

dyna2 computes the solution to the set

$$\ddot{x}_i + 2\xi_i\omega_i\dot{x}_i + \omega_i^2 x_i = f_i g(t), \qquad i = 1, ..., m$$

of differential equations, where $g(t)$ is a piecewise linear time function.

The set of vectors w2, xi and f contains the squared circular frequencies ω_i^2, the damping ratios ξ_i and the applied forces f_i, respectively. The vector g defines the load function in terms of straight line segments between equally spaced points in time. This function may have been formed by the command gfunc.

The dynamic solution is computed at equal time increments defined by dt. Including the initial zero vector as the first column vector, the result is stored in the m-by-n matrix X, $n - 1$ being the number of time steps.

Note:

The accuracy of the solution is *not* a function of the output time increment dt, since the command produces the exact solution for straight line segments in the loading time function.

See also:

gfunc

Purpose:

Compute the dynamic solution to a set of uncoupled second-order differential equations.

Syntax:

Y=dyna2f(w2,xi,f,p,dt)

Description:

dyna2f computes the solution to the set

$$\ddot{x}_i + 2\xi_i\omega_i\dot{x}_i + \omega_i^2 x_i = f_i g(t), \qquad i = 1, ..., m$$

of differential equations in the frequency domain.

The vectors w2, xi and f are the squared circular frequencies ω_i^2, the damping ratios ξ_i and the applied forces f_i, respectively. The force vector p contains the Fourier coefficients $p(k)$ formed by the command fft.

The solution in the frequency domain is computed at equal time increments defined by dt. The result is stored in the m-by-n matrix Y, where m is the number of equations and n is the number of frequencies resulting from the fft command. The dynamic solution in the time domain is achieved by the use of the command ifft.

Example:

The dynamic solution to a set of uncoupled second-order differential equations can be computed by the following sequence of commands:

```
>> g=gfunc(G,dt);
>> p=fft(g);
>> Y=dyna2f(w2,xi,f,p,dt);
>> X=(real(ifft(Y.')))';
```

where it is assumed that the input variables G, dt, w2, xi and f are properly defined. Note that the ifft command operates on column vectors if Y is a matrix; therefore use the transpose of Y. The output from the ifft command is complex. Therefore use Y.' to transpose rows and columns in Y in order to avoid the complex conjugate transpose of Y; see Section 3. The time response is represented by the real part of the output from the ifft command. If the transpose is used and the result is stored in a matrix X, each row will represent the time response for each equation as the output from the command dyna2.

See also:

gfunc, fft, ifft

Purpose:

Transform functions in time domain to frequency domain.

Syntax:

p=fft(g)
p=fft(g,N)

Description:

fft transforms a time dependent function to the frequency domain.

The function to be transformed is stored in the vector g. Each row in g contains the value of the function at equal time intervals. The function represents a span $-\infty \leq t \leq +\infty$; however, only the values within a typical period are specified by g.

The fft command can be used with one or two input arguments. If N is not specified, the numbers of frequencies used in the transformation is equal to the the numbers of points in the time domain, i.e. the length of the variable g, and the output will be a vector of the same size containing complex values representing the frequency content of the input signal.

The scalar variable N can be used to specify the numbers of frequencies used in the Fourier transform. The size of the output vector in this case will be equal to N. It should be remembered that the highest harmonic component in the time signal that can be identified by the Fourier transform corresponds to half the sampling frequency. The sampling frequency is equal to $1/dt$, where dt is the time increment of the time signal.

The complex Fourier coefficients $p(k)$ are stored in the vector p, and are computed according to

$$p(k) = \sum_{j=1}^{N} x(j)\omega_N^{(j-1)(k-1)},$$

where

$$\omega_N = e^{-2\pi i/N}.$$

Note:

This is a MATLAB built-in function.

Purpose:

Compute frequency response of a known discrete time response.

Syntax:

[Freq,Resp] = freqresp(D,dt)

Description:

freqresp computes the frequency response of a discrete dynamic system.

D is the time history function and dt is the sampling time increment, i.e. the time increment used in the time integration procedure.

Resp contains the computed response as a function of frequency. Freq contains the corresponding frequencies.

Example:

The result can be visualized by

```
>> plot(Freq,Resp)
>> xlabel('frequency (Hz)')
```

or

```
>> semilogy(Freq,Resp)
>> xlabel('frequency (Hz)')
```

The dimension of Resp is the same as that of the original time history function.

Note:

The time history function of a discrete system computed by direct integration behaves often in an unstructured manner. The reason for this is that the time history is a mixture of several participating eigenmodes at different eigenfrequencies. By using a Fourier transform, however, the response as a function of frequency can be computed efficiently. In particular it is possible to identify the participating frequencies.

Purpose:

Form vector with function values at equally spaced points by linear interpolation.
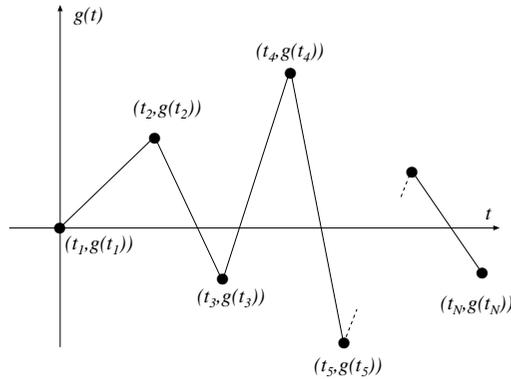
Syntax:

[t,g]=gfunc(G,dt)

Description:

gfunc uses linear interpolation to compute values at equally spaced points for a discrete function g given by

$$
\mathsf{G} = \left[\begin{array}{cc} t_1 & g(t_1) \\ t_2 & g(t_2) \\ \vdots & \\ t_N & g(t_N) \end{array} \right],
$$

as shown in the figure above.

Function values are computed in the range $t_1 \le t \le t_N$, at equal increments, dt being defined by the variable dt. The number of linear segments (steps) is $(t_N - t_1)/dt$. The corresponding vector t is also computed. The result can be plotted by using the command plot(t,g).

Purpose:

Transform function in frequency domain to time domain.

Syntax:

x=ifft(y)
x=ifft(y,N)

Description:

ifft transforms a function in the frequency domain to a function in the time domain.

The function to be transformed is given in the vector y. Each row in y contains Fourier terms in the interval $-\infty \leq \omega \leq +\infty$.

The fft command can be used with one or two input arguments. The scalar variable N can be used to specify the numbers of frequencies used in the Fourier transform. The size of the output vector in this case will be equal to N. See also the description of the command fft.

The inverse Fourier coefficients $x(j)$, stored in the variable x, are computed according to

$$x(j) = (1/N) \sum_{k=1}^{N} y(k)\omega_N^{-(j-1)(k-1)},$$

where

$$\omega_N = e^{-2\pi i/N}.$$

Note:

This is a MATLAB built-in function.

See also:

fft

Purpose:

Compute approximative eigenvalues and eigenvectors by the Lanczos method.

Syntax:

L=ritz(K,M,f,m)
L=ritz(K,M,f,m,b)
[L,X]=ritz(K,M,f,m)
[L,X]=ritz(K,M,f,m,b)

Description:

ritz computes, by the use of the Lanczos algorithm, m approximative eigenvalues and m corresponding eigenvectors for a given pair of n-by-n matrices K and M and a given non-zero starting vector f.

If certain rows and columns in matrices K and M are to be eliminated in computing the eigenvalues, b must be given in the command. The rows (and columns) to be eliminated are described in the vector b defined as

$$b = \begin{bmatrix} dof_1 \\ dof_2 \\ \vdots \\ dof_{nb} \end{bmatrix}.$$

Note:

If the number of vectors, m, is chosen less than the total number of degrees-of-freedom, n, only about the first $m/2$ Ritz vectors are good approximations of the true eigenvectors. Recall that the Ritz vectors satisfy the M-orthonormality condition

$$X^T M X = I,$$

where I is the identity matrix.

Purpose:

Compute seismic response spectra for elastic design.

Syntax:

s=spectra(a,xi,dt,f)

Description:

spectra computes the seismic response spectrum for a known acceleration history function.

The computation is based on the vector a, that contains an acceleration time history function defined at equal time steps. The time step is specified by the variable dt. The value of the damping ratio is given by the variable xi.

Output from the computation, stored in the vector s, is achieved at frequencies specified by the column vector f.

Example:

The following procedure can be used to produce a seismic response spectrum for a damping ratio $\xi = 0.05$, defined at 34 logarithmicly spaced frequency points. The acceleration time history a has been sampled at a frequency of 50 Hz, corresponding to a time increment $dt = 0.02$ between collected points:

```
>> freq=logspace(0,log10(2^(33/6)),34);
>> xi=0.05;
>> dt=0.02;
>> s=spectra(a,xi,dt,freq');
```

The resulting spectrum can be plotted by the command

```
>> loglog(freq,s,'*')
```

Purpose:

Compute the dynamic solution to a set of first order differential equations.

Syntax:

Tsnap=step1(K,C,d0,ip,f,pbound)
[Tsnap,D,V]=step1(K,C,d0,ip,f,pbound)

Description:

step1 computes at equal time steps the solution to a set of first order differential equations of the form

$$\mathbf{C}\dot{\mathbf{d}} + \mathbf{K}\mathbf{d} = \mathbf{f}(x,t),$$
$$\mathbf{d}(0) = \mathbf{d}_0.$$

The command solves transient field problems. In the case of heat conduction, K and C represent the $n \times n$ conductivity and capacity matrices, respectively.

The initial conditions are given by the vector d0 containing initial values of d. The time integration procedure is governed by the parameters given in the vector ip defined as

$$\mathsf{ip} = [dt\ T\ \alpha\ \ [nsnap\ nhist\ \ time_i\ ...\ \ dof_i\ ...\ \ ]],$$

list of	list of
nsnap	*nhist*
moments	dofs

where dt specifies the time increment in the time stepping scheme, T total time and α a time integration constant; see [1]. The parameter *nsnap* denotes the number of snapshots stored in Tsnap. The selected elapsed times are specified in $(time_i\ ...\ )$, whereas *nhist* is the number of time histories stored in D and V. The selected degrees-of-freedom are specified in $(dof_i\ ...\ )$. The following table lists frequently used values of α:

$\alpha = 0$ Forward difference; forward Euler,

$\alpha = \frac{1}{2}$ Trapezoidal rule; Crank-Nicholson,

$\alpha = 1$ Backward difference; backward Euler.

The matrix f contains the time-dependent load vectors. If no external loads are active, the matrix corresponding to f should be replaced by []. The matrix f contains the time-dependent prescribed values of the field variable. If no field variables are prescribed the matrix corresponding to pbound should be replaced by []. Matrix f is organized in the following manner:

$$\mathsf{f} = \begin{bmatrix} \text{time history of the load at } dof_1 \\ \text{time history of the load at } dof_2 \\ \vdots \\ \text{time history of the load at } dof_n \end{bmatrix}.$$

The dimension of f is

$$(\text{number of degrees-of-freedom}) \times (\text{number of timesteps} + 1).$$

The matrix pbound is organized in the following manner:

$$\mathsf{pbound} = \begin{bmatrix} dof_1 & \text{time history of the field variable} \\ dof_2 & \text{time history of the field variable} \\ \vdots & \vdots \\ dof_{m_2} & \text{time history of the field variable} \end{bmatrix}.$$

The dimension of pbound is

$$(\text{number of dofs with prescribed field values}) \times (\text{number of timesteps} + 2).$$

The time history functions can be generated using the command gfunc. If all the values of the time histories of f or pbound are kept constant, these values need to be stated only once. In this case the number of columns in f is one and in pbound two.

It is highly recommended to define f as sparse (a MATLAB built-in function). In most cases only a few degrees-of-freedom are affected by the exterior load, and hence the matrix contains only few non-zero entries.

The computed snapshots are stored in Tsnap, one column for each requested snapshot according to ip, i.e. the dimension of Tsnap is (number of dofs) $\times$ $nsnap$. The computed time histories of $\mathbf{d}$ and $\dot{\mathbf{d}}$ are stored in D and V, respectively, one line for each requested degree-of-freedom according to ip. The dimension of D is $nhist \times$ (number of timesteps $+ 1$).

Purpose:

Compute the dynamic solution to a set of second order differential equations.

Syntax:

Dsnap=step2(K,C,M,d0,v0,ip,f,pdisp)
[Dsnap,D,V,A]=step2(K,C,M,d0,v0,ip,f,pdisp)

Description:

step2 computes at equal time steps the solution to a second order differential equations of the form

$$\mathbf{M}\ddot{\mathbf{d}} + \mathbf{C}\dot{\mathbf{d}} + \mathbf{K}\mathbf{d} = \mathbf{f}(x, t),$$
$$\mathbf{d}(0) = \mathbf{d}_0,$$
$$\dot{\mathbf{d}}(0) = \mathbf{v}_0.$$

In structural mechanics problems, K , C and M represent the $n \times n$ stiffness, damping and mass matrices, respectively.

The initial conditions are given by the vectors d0 and v0, containing initial displacements and initial velocities. The time integration procedure is governed by the parameters given in the vector ip defined as

$$\mathsf{ip} = [dt \; T \; \alpha \; \delta \; [nsnap \; nhist \quad time_i \; ... \; dof_i \; ... \quad]]\,,$$

$$\underbrace{}_{\substack{\text{list of} \\ nsnap \\ \text{moments}}} \underbrace{}_{\substack{\text{list of} \\ nhist \\ \text{dofs}}}$$

where dt specifies the time increment in the time stepping scheme, T the total time and α and δ time integration constants for the Newmark family of methods; see [1]. The parameter $nsnap$ denotes the number of snapshots stored in Dsnap. The selected elapsed times are specified in $(time_i \; ... \;)$, whereas $nhist$ is the number of time histories stored in D, V and A. The selected degrees-of-freedom are specified in $(dof_i \; ... \;)$. The following table lists frequently used values of α and δ:

$\alpha = \frac{1}{4}$ $\delta = \frac{1}{2}$ Average acceleration (trapezoidal) rule,

$\alpha = \frac{1}{6}$ $\delta = \frac{1}{2}$ Linear acceleration,

$\alpha = 0$ $\delta = \frac{1}{2}$ Central difference.

The matrix f contains the time-dependent load vectors. If no external loads are active, the matrix corresponding to f should be replaced by []. The matrix pdisp contains the time-dependent prescribed displacement. If no displacements are prescribed the matrix corresponding to pdisp should be replaced by [].

The matrix f is organized in the following manner:

$$
\mathsf{f} = \begin{bmatrix}
\text{time history of the load at } dof_1 \\
\text{time history of the load at } dof_2 \\
\vdots \\
\text{time history of the load at } dof_n
\end{bmatrix}.
$$

The dimension of f is

$$(\text{number of degrees-of-freedom}) \times (\text{number of timesteps} + 1).$$

The matrix pdisp is organized in the following manner

$$
\mathsf{pdisp} = \begin{bmatrix}
dof_1 & \text{time history of the displacement} \\
dof_2 & \text{time history of the displacement} \\
\vdots & \vdots \\
dof_{m_2} & \text{time history of the displacement}
\end{bmatrix}.
$$

The dimension of pdisp is

$$(\text{number of dofs with prescribed displacement}) \times (\text{number of timesteps} + 2).$$

The time history functions can be generated using the command gfunc. If all the values of the time histories of f or pdisp are kept constant, these values need to be stated only once. In this case the number of columns in f is one and in pdisp two.

It is highly recommended to define f as sparse (a MATLAB built-in function). In most cases only a few degrees-of-freedom are affected by the exterior load, and hence the matrix contains only few non-zero entries.

The computed displacement snapshots are stored in Dsnap, one column for each requested snapshot according to ip, i.e. the dimension of Dsnap is (number of dofs) $\times$ $nsnap$. The computed time histories of $\mathbf{d}$, $\dot{\mathbf{d}}$ and $\ddot{\mathbf{d}}$ are stored in D, V and A, respectively, one line for each requested degree-of-freedom according to ip. The dimension of D is $nhist \times$ (number of timesteps $+ 1$).

Purpose:

Compute complex frequency response functions.

Syntax:

Y=sweep(K,C,M,p,w)

Description:

sweep computes the complex frequency response function for a system of the form

$$[\mathbf{K} + i\omega\mathbf{C} - \omega^2\mathbf{M}]\mathbf{y}(\omega) = \mathbf{p}.$$

Here K, C and M represent the m-by-m stiffness, damping and mass matrices, respectively. The vector p defines the amplitude of the force. The frequency response function is computed for the values of ω given by the vector w.

The complex frequency response function is stored in the matrix Y with dimension m-by-n, where n is equal to the number of circular frequencies defined in w.

Example:

The steady-state response can be computed by

```
>> X=real(Y*exp(i*w*t));
```

and the amplitude by

```
>> Z=abs(Y);
```

7 Statements and macros

Statements describe algorithmic actions that can be executed. There are two different types of control statements, conditional and repetitive. The first group defines conditional jumps whereas the latter one defines repetition until a conditional statement is fulfilled. Macros are used to define new functions to the MATLAB or CALFEM structure, or to store a sequence of statements in an .m-file.

Control statements	
if	Conditional jump
for	Initiate a loop
while	Define a conditional loop

Macros	
function	Define a new function
script	Store a sequence of statements

STATEMENTS

Purpose:

Conditional jump.

Syntax:

if *logical expression*

⋮

elseif *logical expression*

⋮

else

⋮

end

Description:

if initiates a conditional jump. If *logical expression* produces the value *True* the statements following if are executed. If *logical expression* produces the value *False* the next conditional statement elseif is checked.

elseif works like if. One or more of the conditional statement elseif can be added after the initial conditional statement if.

If else is present, the statements following else are executed if the *logical expressions* in all if and elseif statements produce the value *False*. The if loop is closed by end to define the loop sequence.

The following relation operators can be used

==	equal
>=	greater than or equal to
>	greater than
<=	less than or equal to
<	less than
∼=	not equal

Note:

This is MATLAB built-in language.

Purpose:

Initiate a loop.

Syntax:

for i = start : inc : stop
⋮
end

Description:

for initiates a loop which terminates when i>stop. The for loop is closed by end to define the loop sequence.

Examples:

for i = 1 : 10	i takes values from 1 to 10.
for i = 1 : 2 : 10	i equals 1, 3, 5, 7, 9.
for i = 20 : -1 : 1	i equals 20, 19 ... 2, 1.

Note:

This is MATLAB built-in language.

Purpose:

Define a conditional loop.

Syntax:

while *logical expression*
⋮
end

Description:

while initiates a conditional loop which terminates when *logical expression* equals *False.* The while loop is closed by end to define the loop sequence.

The different relation operators that can be used can be found under the if command.

Examples:

A loop continuing until a equals b

while a∼=b
⋮
end

Note:

This is MATLAB built-in language.

Purpose:

Define a new function.

Syntax:

function[out1 , out2 , ...]=*name*(in1 , in2 , ...)

Description:

name is replaced by the name of the function. The input variables in1, in2, ... can be scalars, vectors or matrices, and the same holds for the output variables out1, out2, ...

Example:

To define the CALFEM function spring1e a file named spring1e.m is created. The file contains the following statements:

```
function [Ke]=spring1e(k)
% Define the stiffness matrix
% for a one dimensional spring
% with spring stiffness k
Ke=[ k, -k; -k, k ]
```

i.e. the function spring1e is defined to return a stiffness matrix.

Note:

This is MATLAB built-in language.

Purpose:

Execute a stored sequence of statements.

Syntax:

name

Description:

name is replaced by the name of the script.

Example:

The statements below are stored in a file named spring.m and executed by typing spring in the MATLAB command window.

```
% Stiffness matrix for a one dimensional
% spring with stiffness k=10
k=10;
[Ke]=spring1e(k);
```

Note:

This is MATLAB built-in language.

8 Graphics functions

The group of graphics functions comprises functions for element based graphics. Mesh plots, displacements, section forces, flows, iso lines and principal stresses can be displayed. The functions are divided into two dimensional, and general graphics functions.

Two dimensional graphics functions	
plot	Plot lines and points in 2D space
fill	Draw filled 2D polygons
eldraw2	Draw undeformed finite element mesh
eldisp2	Draw deformed finite element mesh
eldia2	Draw section force diagram
elflux2	Plot flux vectors
eliso2	Draw isolines for nodal quantities
elprinc2	Plot principal stresses
scalfact2	Determine scale factor
pltscalb2	Draw scale bar

General graphics functions	
axis	Axis scaling and appearance
clf	Clear current figure
figure	Create figures
grid	Grid lines
hold	Hold current graph
print	Print graph or save graph to file
text	Add text to current plot
title	Titles for 2D and 3D plots
xlabel, ylabel, zlabel	Axis labels for 2D and 3D plots

Purpose:

Plot axis scaling and appearance.

Syntax:

axis([xmin xmax ymin ymax])
axis([xmin xmax ymin ymax zmin zmax])
axis auto
axis square
axis equal
axis off
axis on

Description:

axis([xmin xmax ymin ymax]) sets scaling for the x- and y-axes on the current 2D plot.

axis([xmin xmax ymin ymax zmin zmax]) sets the scaling for the x-, y- and z-axes on the current 3D plot.

axis auto returns the axis scaling to its default automatic mode where, for each plot, xmin = min(x), xmax = max(x), etc.

axis square makes the current axis box square in shape.

axis equal changes the current axis box size so that equal tick mark increments on the x- and y-axes are equal in size. This makes plot(sin(x),cos(x)) look like a circle, instead of an oval.

axis normal restores the current axis box to full size and removes any restrictions on the scaling of the units. This undoes the effects of axis square and axis equal.

axis off turns off all axis labeling and tick marks.

axis on turns axis labeling and tick marks back on.

Note:

This is a MATLAB built-in function. For more information about the axis function, type help axis.

Purpose:

Clear current figure (graph window).
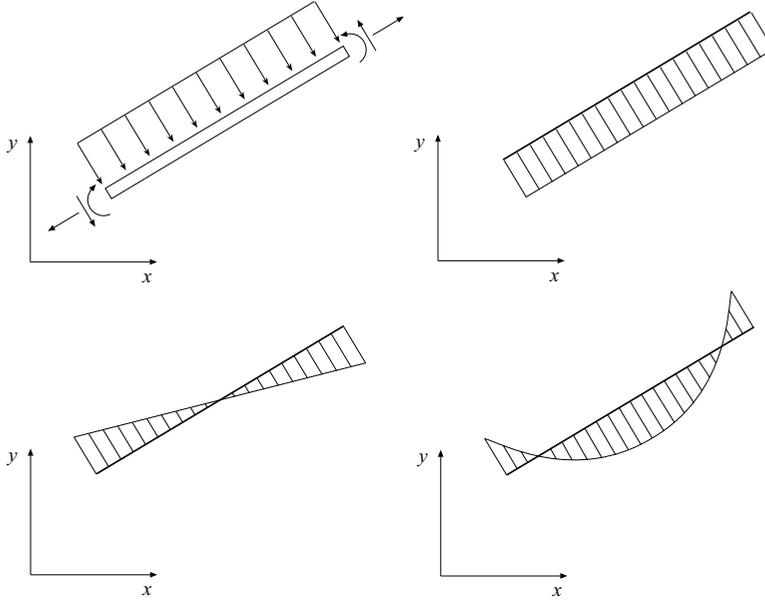
Syntax:

clf

Description:

clf deletes all objects (axes) from the current figure.

Note:

This is a MATLAB built-in function. For more information about the clf function, type help clf.

Purpose:

Draw the section force diagrams of a two dimensional beam element.

Syntax:

eldia2(ex,ey,es,plotpar,sfac)
eldia2(ex,ey,es,plotpar,sfac,eci)
[sfac]=eldia2(ex,ey,es)
[sfac]=eldia2(ex,ey,es,plotpar)

Description:

eldia2 plots a section force diagram of a two dimensional beam element in its global position.

The input variables ex and ey are defined in beam2e and the input variable

$$
\mathsf{es} = \left[\begin{array}{c} S_1 \\ S_2 \\ \vdots \\ S_n \end{array} \right]
$$

consists of a column matrix that contains section forces. The values in es are computed in beam2s. It should be noted, however, that whereas all three section forces are computed in beam2s only one of them shall be given as input to eldia2 by es.

The variable plotpar sets plot parameters for the diagram.

$$\text{plotpar} = [\; linecolor \quad elementcolor \;]$$

$$
\begin{array}{llll}
linecolor = 1 & \text{black} & elementcolor = 1 & \text{black} \\
2 & \text{blue} & 2 & \text{blue} \\
3 & \text{magenta} & 3 & \text{magenta} \\
4 & \text{red} & 4 & \text{red}
\end{array}
$$

The scale factor sfac is a scalar that the section forces are multiplied with to get a suitable graphical representation. sfac is set automatically if it is omitted in the input list.

The input variable

$$
\text{eci} =
\begin{bmatrix}
\bar{x}_1 \\
\bar{x}_2 \\
\vdots \\
\bar{x}_n
\end{bmatrix}
$$

specifies the local $\bar{x}$-coordinates of the quantities in es. If eci is not given, uniform distance is assumed.

Limitations:

Supported elements are two dimensional beam elements.

Purpose:

Draw the deformed mesh for a two dimensional structure.

Syntax:

[sfac]=eldisp2(Ex,Ey,Ed)
[sfac]=eldisp2(Ex,Ey,Ed,plotpar)
eldisp2(Ex,Ey,Ed,plotpar,sfac)

Description:

eldisp2 displays the deformed mesh for a two dimensional structure.

Input variables are the coordinate matrices Ex and Ey formed by the function co-ordxtr, and the element displacements Ed formed by the function extract.

The variable plotpar sets plot parameters for linetype, linecolor and node marker.

$$\text{plotpar}=[\ \textit{linetype}\quad \textit{linecolor}\quad \textit{nodemark}\ ]$$

$$
\begin{array}{llll}
\textit{linetype} = 1 & \text{solid line} & \textit{linecolor} = 1 & \text{black} \\
2 & \text{dashed line} & 2 & \text{blue} \\
3 & \text{dotted line} & 3 & \text{magenta} \\
& & 4 & \text{red}
\end{array}
$$

$$
\begin{array}{ll}
\textit{nodemark} = 1 & \text{circle} \\
2 & \text{star} \\
0 & \text{no mark}
\end{array}
$$

Default is dashed black lines with circles at nodes.

The scale factor sfac is a scalar that the element displacements are multiplied with to get a suitable geometrical representation. The scale factor is set automatically if it is omitted in the input list.

Limitations:

Supported elements are bar, beam, triangular three node, and quadrilateral four node elements.

Purpose:

Draw the undeformed mesh for a two dimensional structure.

Syntax:

eldraw2(Ex,Ey)
eldraw2(Ex,Ey,plotpar)
eldraw2(Ex,Ey,plotpar,elnum)

Description:

eldraw2 displays the undeformed mesh for a two dimensional structure.

Input variables are the coordinate matrices Ex and Ey formed by the function co-ordxtr.

The variable plotpar sets plot parameters for linetype, linecolor and node marker.

$$\mathsf{plotpar} = [\; linetype \quad linecolor \quad nodemark \;]$$

$$
\begin{array}{llll}
linetype = 1 & \text{solid line} & linecolor = 1 & \text{black} \\
2 & \text{dashed line} & 2 & \text{blue} \\
3 & \text{dotted line} & 3 & \text{magenta} \\
& & 4 & \text{red}
\end{array}
$$

$$
\begin{array}{ll}
nodemark = 1 & \text{circle} \\
2 & \text{star} \\
0 & \text{no mark}
\end{array}
$$

Default is solid black lines with circles at nodes.

Element numbers can be displayed at the center of the element if a column vector elnum with the element numbers is supplied. This column vector can be derived from the element topology matrix Edof,

elnum=Edof(:,1)

i.e. the first column of the topology matrix.

Limitations:

Supported elements are bar, beam, triangular three node, and quadrilateral four node elements.

Purpose:

Draw element flow arrows for two dimensional elements.

Syntax:

[sfac]=elflux2(Ex,Ey,Es)
[sfac]=elflux2(Ex,Ey,Es,plotpar)
elflux2(Ex,Ey,Es,plotpar,sfac)

Description:

elflux2 displays element heat flux vectors (or corresponding quantities) for a number of elements of the same type. The flux vectors are displayed as arrows at the element centroids. Note that only the flux vectors are displayed. To display the element mesh, use eldraw2.

Input variables are the coordinate matrices Ex and Ey, and the element flux matrix Es defined in flw2ts or flw2qs.

The variable plotpar sets plot parameters for the flux arrows.

plotpar=[*arrowtype   arrowcolor*]

$arrowtype = 1$	solid	$arrowcolor = 1$	black
2	dashed	2	blue
3	dotted	3	magenta
		4	red

Default, if plotpar is omitted, is solid black arrows.

The scale factor sfac is a scalar that the values are multipied with to get a suitable arrow size in relation to the element size. The scale factor is set automatically if it is omitted in the input list.

Limitations:

Supported elements are triangular 3 node and quadrilateral 4 node elements.

Purpose:

Display element iso lines for two dimensional elements.

Syntax:

eliso2(Ex,Ey,Ed,isov)
eliso2(Ex,Ey,Ed,isov,plotpar)

Description:

eliso2 displays element iso lines for a number of elements of the same type. Note that only the iso lines are displayed. To display the element mesh, use eldraw2.

Input variables are the coordinate matrices Ex and Ey formed by the function coordxtr, and the element nodal quantities (e.g displacement or energy potential) matrix Ed defined in extract.

If isov is a scalar it determines the number of iso lines to be displayed. If isov is a vector it determines the values of the iso lines to be displayed (number of iso lines equal to length of vector isov).

$$\text{isov} = [\, isolines \,]$$
$$\text{isov} = [\, isovalue(1) \, ... \, isovalue(n) \,]$$

The variable plotpar sets plot parameters for the iso lines.

$$\text{plotpar} = [\, linetype \quad linecolor \quad textfcn \,]$$

$arrowtype =$	1	solid	$arrowcolor =$	1	black
	2	dashed		2	blue
	3	dotted		3	magenta
				4	red

$textfcn =$	0	the iso values of the lines will not be printed
	1	the iso values of the lines will be printed at the iso lines
	2	the iso values of the lines will be printed where the cursor indicates

Default is solid, black lines and no iso values printed.

Limitations:

Supported elements are triangular 3 node and quadrilateral 4 node elements.

Purpose:

Draw element principal stresses as arrows for two dimensional elements.

Syntax:

[sfac]=elprinc2(Ex,Ey,Es)
[sfac]=elprinc2(Ex,Ey,Es,plotpar)
elprinc2(Ex,Ey,Es,plotpar,sfac)

Description:

elprinc2 displays element principal stresses for a number of elements of the same type. The principal stresses are displayed as arrows at the element centroids. Note that only the principal stresses are displayed. To display the element mesh, use eldraw2.

Input variables are the coordinate matrices Ex and Ey, and the element stresses matrix Es defined in plants or planqs

The variable plotpar sets plot parameters for the principal stress arrows.

plotpar=[*arrowtype*   *arrowcolor*]

$arrowtype = 1$	solid	$arrowcolor = 1$	black
2	dashed	2	blue
3	dotted	3	magenta
		4	red

Default, if plotpar is omitted, is solid black arrows.

The scale factor sfac is a scalar that values are multiplied with to get a suitable arrow size in relation to the element size. The scale factor is set automatically if it is omitted in the input list.

Limitations:

Supported elements are triangular 3 node and quadrilateral 4 node elements.

Purpose:

Create figures (graph windows).

Syntax:

figure(h)

Description:

figure(h) makes the h'th figure the current figure for subsequent plotting functions. If *figure h* does not exist, a new one is created using the first available figure handle.

Note:

This is a MATLAB built-in function. For more information about the figure function, type help figure.

Purpose:

Filled 2D polygons.

Syntax:

fill(x,y,c)
fill(X,Y,C)

Description:

fill(x,y,c) fills the 2D polygon defined by vectors x and y with the color specified by c. The vertices of the polygon are specified by pairs of components of x and y. If necessary, the polygon is closed by connecting the last vertex to the first.

If c is a vector of the same length as x and y, its elements are used to specify colors at the vertices. The color within the polygon is obtained by bilinear interpolation in the vertex colors.

If X, Y and C are matrices of the same size, fill(X,Y,C) draws one polygon per column with interpolated colors.

Example:

The solution of a heat conduction problem results in a vector d with nodal temperatures. The temperature distribution in a group of triangular 3 node (nen=3) or quadrilateral 4 node (nen=4) elements, with topology defined by edof, can be displayed by

```
[ex,ey]=coordxtr(edof,Coord,Dof,nen)
ed=extract(edof,d)
colormap(hot)
fill(ex',ey',ed')
```

Note:

This is a MATLAB built-in function. For more information about the fill function, type help fill.

Purpose:

Grid lines for 2D and 3D plots.

Syntax:

grid on
grid off
grid

Description:

grid on adds grid lines on the current axes.

grid off takes them off.

grid by itself, toggles the grid state.

Note:

This is a MATLAB built-in function. For more information about the grid function, type help grid.

Purpose:

Hold the current graph.

Syntax:

hold on
hold off
hold

Description:

hold on holds the current graph.

hold off returns to the default mode where plot functions erase previous plots.

hold by itself, toggles the hold state.

Note:

This is a MATLAB built-in function. For more information about the hold function, type help hold.

Purpose:

Linear two dimensional plot.

Syntax:

plot(x,y)
plot(x,y,'linetype')

Description:

plot(x,y) plots vector x versus vector y. Straight lines are drawn between each pair of values.

Various line types, plot symbols and colors may be obtained with plot(x,y,s) where s is a 1, 2, or 3 character string made from the following characters:

–	solid line	.	point	y	yellow
:	dotted line	o	circle	m	magenta
-.	dashdot line	x	x-mark	c	cyan
- -	dashed line	+	plus	r	red
		*	star	g	green
				b	blue
				w	white
				k	black

Default is solid blue lines.

Example:

The statement

```
plot(x,y,'-',x,y,'ro')
```

plots the data twice, giving a solid blue line with red circles at the data points.

Note:

This is a MATLAB built-in function. For more information about the plot function, type help plot.

Purpose:

Draw a scale bar.

Syntax:

pltscalb2(sfac,magnitude)
pltscalb2(sfac,magnitude,plotpar)

Description:

pltscalb2 draws a scale bar to visualize the magnitude of displayed computational results. The input variable sfact is a scale factor determined by the function scalfact2 and the variable

$$\text{magnitude} = [\, S \; x \; y \,]$$

specifies the value corresponding the length of the scale bar S, and (x, y) are the coordinates of the starting point. If no coordinates are given the starting point will be (0,-0.5).

The variable plotpar sets the the scale bar colour.

plotpar=[$color$]

$$color = \begin{array}{ll} 1 & \text{black} \\ 2 & \text{blue} \\ 3 & \text{magenta} \\ 4 & \text{red} \end{array}$$

Purpose:

Create hardcopy output of current figure window.

Syntax:

print [*filename*]

Description:

print with no arguments sends the contents of the current figure window to the default printer. print *filename* creates a PostScript file of the current figure window and writes it to the specified file.

Note:

This is a MATLAB built-in function. For more information about the print function, type help print.

Purpose:

Determine scale factor for drawing computational results.

Syntax:

[sfac]=scalfact2(ex,ey,ed)
[sfac]=scalfact2(ex,ey,ed,rat)

Description:

scalfact2 determines a scale factor sfac for drawing computational results, such as displacements, section forces or flux.

Input variables are the coordinate matrices ex and ey and the matrix ed containing the quantity to be displayed. The scalar rat defines the ratio between the geometric representation of the largest quantity to be displayed and the element size. If rat is not specified, 0.2 is used.

Purpose:

Add text to current plot.

Syntax:

text(x,y,'string')

Description:

text adds the text in the quotes to location (x,y) on the current axes, where (x,y) is in units from the current plot. If x and y are vectors, text writes the text at all locations given. If 'string' is an array with the same number of rows as the length of x and y, text marks each point with the corresponding row of the 'string' array.

Note:

This is a MATLAB built-in function. For more information about the text function, type help text.

Purpose:

Titles for 2D and 3D plots.

Syntax:

title('text')

Description:

title adds the text string 'text' at the top of the current plot.

Note:

This is a MATLAB built-in function. For more information about the title function, type help title.

Purpose:

x-, y-, and z-axis labels for 2D and 3D plots.

Syntax:

xlabel('text')
ylabel('text')
zlabel('text')

Description:

xlabel adds text beside the x-axis on the current plot.

ylabel adds text beside the y-axis on the current plot.

zlabel adds text beside the z-axis on the current plot.

Note:

This is a MATLAB built-in function. For more information about the functions, type help xlabel, help ylabel, or help zlabel.

9 User's Manual, examples

9.1 Introduction

This set of examples is defined with the ambition to serve as a User's Manual. The examples, except the introductory ones, are written as .m-files (script files) and supplied together with the CALFEM functions.

The User's Manual examples are separated into four groups:

MATLAB introduction
Static analysis
Dynamic analysis
Nonlinear analysis

The MATLAB introduction examples explain some basic concepts and introduce a set of standard MATLAB functions usable in the finite element context. The static linear examples illustrate finite element analysis of different structures loaded by stationary loads. The dynamic linear examples illustrate some basic features in dynamics, such as modal analysis and time stepping procedures. The examples of nonlinear analysis cover subjects such as second order theory and buckling.

9.2   MATLAB introduction

The examples in this section illustrate basic MATLAB concepts such as handling of workspace, variables and functions. The examples are:

MATLAB introduction	
exi1	Handling matrices
exi2	Matrix and array operations
exi3	Create and handle .m-files
exi4	Display formats
exi5	Create a session .log-file
exi6	Graphic display of vectors

Purpose:

Show how to create and handle matrices in MATLAB.

Description:

The following commands create a scalar x, two vectors u and v and two matrices A and B.

Lines starting with the MATLAB prompt >> are command lines while the other lines show the results from these commands.

```
>> x=7

x =
     7

>> u=[1 2 3 4]

u =
     1     2     3     4

>> v=[0:0.4:2]

v =

     0    0.4000    0.8000    1.2000    1.6000    2.0000

>> A=[1 2; 3 4]

A =
     1     2
     3     4

>> B=[5 6; 7 8]

B =
     5     6
     7     8
```

Both brief and detailed listing of variables is possible

```
>> who

Your variables are:

A         B         u         v         x
```

```
>> whos

  Name        Size          Bytes  Class

  A           2x2              32   double array
  B           2x2              32   double array
  u           1x4              32   double array
  v           1x6              48   double array
  x           1x1               8   double array

Grand total is 19 elements using 152 bytes
```

The value of a variable is displayed by writing the variable name,

```
>> u

u =
     1     2     3     4
```

and the dimension $(m \times n)$ of a variable is obtained by

```
>> size(u)

ans =
     1     4
```

where the answer is temporarily stored in the vector ans.

The variable x is removed from workspace by

```
>> clear x
```

To remove *all* variables from workspace, clear without argument is used.

Assignment of a value to an element in a matrix is made as

```
>> A(2,2)=9

A =
     1     2
     3     9
```

To select a complete row or column colon notation is used.

```
>> s=A(:,1)

s =
     1
     3

>> t=A(2,:)

t =
     3     9
```

A zero matrix K (4×4) is generated by

```
>> K=zeros(4,4)

K =
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

Similarly an ($m\times n$) matrix of all ones can be generated by ones(m,n).

Expand an already defined matrix

```
>> H=[A;B]

H =
     1     2
     3     9
     5     6
     7     8

>> J=[A B]

J =
     1     2     5     6
     3     9     7     8
```

Purpose:

Show some examples of basic matrix and element-by-element operations.

Description:

Consider the following matrices

$$a = \begin{bmatrix} 5 & 12 & 3 \end{bmatrix} \qquad b = \begin{bmatrix} 1 & 0 & 4 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 6 & 3 \\ 2 & 8 & 4 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 5 & 4 \\ 7 & 2 & 0 \end{bmatrix}$$

The transpose of a matrix is generated by

```
>> A'

ans =
        1        2
        6        8
        3        4

>> a'

ans =
        5
       12
        3
```

Addition and subtraction of matrices

```
>> A+B

ans =
        3       11        7
        9       10        4

>> A-B

ans =
       -1        1       -1
       -5        6        4
```

Note that if the result of an operation is not assigned to a specific variable, the answer is temporarily stored in the variable ans.

Multiplication of matrices

```
>> a*b'

ans =
      17

>> a'*b

ans =
      5      0     20
     12      0     48
      3      0     12

>> A'*B

ans =
     16      9      4
     68     46     24
     34     23     12

>> C=B*A'

C =
     44     60
     19     30
```

To perform arithmetic operations, matrix dimensions must agree

```
>> D=A*B
??? Error using ==> *
Inner matrix dimensions must agree.
```

The inverse of a square matrix

```
>> inv(C)

ans =
      0.1667    -0.3333
     -0.1056     0.2444
```

The determinant of a square matrix

```
>> det(C)

ans =
     180
```

An array or element-by-element arithmetic operation is denoted by a period (.) preceding an operator. Examples are element-by-element multiplication (.*), division (./), and powers (.^).

```
>> a.*b

ans =
     5      0     12

>> A.*B

ans =
     2     30     12
    14     16      0
```

Matrices in element-by-element operations must obviously have the same dimensions.

Mathematical functions applied to arrays are also evaluated element-by-element.

```
>> sin([0 1 2 3 4]*pi/2)

ans =
     0      1      0     -1      0
```

Purpose:

Show how to handle script files and function files.

Description:

When starting a MATLAB session the default working directory is according to initial settings, for example C:\USER. A new working directory can be chosen by typing for example

```
>> cd A:
```

which makes the root directory in drive A the working directory.

Files containing MATLAB and/or CALFEM statements are characterized by the suffix .m. For example the file bar2e.m contains statements to evaluate the element stiffness matrix for a two dimensional bar element. An .m-file is allowed to include references to other .m-files, and also to call itself recursively.

Two types of .m-files can be used, *script files* and *function files*. *Script files* collect a sequence of statements under one command name. *Function files* allow new functions with input and/or output variables to be defined. Both *script files* and *function files* are ordinary ASCII text files and can therefore be created in an arbitrary editor. In the MATLAB environment an .m-file editor can be activated from the pull down menu on top of the MATLAB window.

An example of a *script file* is given below. The following sequence of statements is typed in the .m-file editor, and saved as test.m.

```
% ----- Script file test.m -----
A=[0 4;2 8]
B=[3 9;5 7]
C=A*B
% ----------- end -------------
```

A line starting with an % is regarded as a comment line.

The statements are executed by writing the file name (without the suffix .m) in the command window

```
>> test

A =
     0     4
     2     8
```

```
B =
     3      9
     5      7

C =
    20     28
    46     74
```

The second type of .m-files is the *function file*. The first line of these files contains the word function. The example below is a function that computes the second and third power of a scalar.

```
function [b,c]=func1(a)
% ----- function file 'func1.m' -----
b=a*a;
c=a*a*a;
% ------------ end -----------------
```

The semi-colon prohibits the echo display of the variables on the screen.

The file can be created using an ordinary editor, and it must be saved as func1.m i.e. the file name without extension must be the same as the function name.

The second and third power of 2 are calculated by typing

```
>> [b,c]=func1(2)
```

producing

```
b =
     4

c =
     8
```

See also *function* and *script* in Section 7.

Purpose:

Show different display formats.

Description:

Consider the following matrix operation

```
>> A=[0 4;2 8];
>> B=[3 9;5 7];
>> C=A*B/2537

C =
    0.0079    0.0110
    0.0181    0.0292
```

The result from the computation of C above is shown in the default format, displaying four significant decimal digits.

Other formats can be defined by the command format

```
>> format long
>> C

C =
    0.00788332676389   0.01103665746945
    0.01813165155696   0.02916830902641

>> format short e
>> C

C =
    7.8833e-003   1.1037e-002
    1.8132e-002   2.9168e-002

>> format long e
>> C

C =
    7.883326763894364e-003    1.103665746945211e-002
    1.813165155695703e-002    2.916830902640915e-002
```

Purpose:

How to make a command window session .log-file.

Description:

The diary and echo commands are useful for presentation purposes since the complete set of statements can be saved together with some selected results.

The command

```
>> diary filename
```

saves all subsequent terminal input and resulting output in the file named *filename* on the default device. The file is closed using

```
>> diary off
```

Consider the script file test.m.

```
% ----- Script file test.m -----
diary testlog
echo on
A=[0 4;2 8];
B=[3 9;5 7];
C=A*B/2537
echo off
diary off
% ------------ end ------------
```

Normally, the statements in an .m-file do not display during execution. The commands echo on and echo off allow the statements to be viewed as they execute. Execution of test.m yields

```
>>test

A=[0 4;2 8];

B=[3 9;5 7];

C=A*B/2537

C =
    0.0079    0.0110
    0.0181    0.0292
```

in the command window and on the file testlog as well.

Purpose:

How to display vectors and handle the graphics window.

Description:

The contents of a vector versus the vector index or a vector versus another vector can be displayed in the graphics window. Consider the vectors

$$\mathsf{x} = \begin{bmatrix} 1 & 2 & 5 \end{bmatrix} \qquad \mathsf{y} = \begin{bmatrix} 5 & 22 & 16 \end{bmatrix}$$

The function

```
>> plot(y)
```

plots the contents of the vector y versus vector index and

```
>> plot(x,y)
```

plots the contents of the vector y versus the vector x.

The commands

title(*'text'*)
xlabel(*'xlabel'*)
ylabel(*'ylabel'*)

write *text* as a title of the current plot, and *xlabel* and *ylabel* as labels of the coordinate axis.

Grid lines are added with

grid

and

clf

clears the current figure.

9.3 Static analysis

This section illustrates some linear static finite element calculations. The examples deal with structural problems as well as field problems such as heat conduction.

Static analysis	
exs1	Linear spring system
exs2	One-dimensional heat flow
exs3	Plane truss
exs4	Plane truss analysed using loops
exs5	Simply supported beam
exs6	Plane frame
exs7	Plane frame stabilized with bars
exs8	Two dimensional diffusion

The introductory example exs1 illustrates the basic steps in the finite element method for a simple structure of one-dimensional linear springs. The linear spring or analogy element is also used in example exs2 to solve a problem of heat conduction in a wall. A plane truss consisting of three bars is analysed in exs3. In example exs4 a plane truss consisting of ten bars is analysed using loops. First the analysis is performed by defining coordinate data for each element directly, and then it is shown how this data can be obtained from global coordinate data. A simply supported beam is analysed in example exs5. Element forces and the support forces are calculated. A two dimensional plane frame is analysed in example exs6. A structure built up from both beams and bars is analysed in example exs7. Graphics facilities are also explained in examples exs6, exs7, and exs8.

Note: The examples listed above are supplied as .m-files under the directory examples. The example files are named according to the table.

Purpose:

Show the basic steps in a finite element calculation.

Description:

The general procedure in linear finite element calculations is carried out for a simple structure. The steps are

- define the model
- generate element matrices
- assemble element matrices into the global system of equations
- solve the global system of equations
- evaluate element forces

Consider the system of three linear elastic springs, and the corresponding finite element model. The system of springs is fixed in its ends and loaded by a single load F.
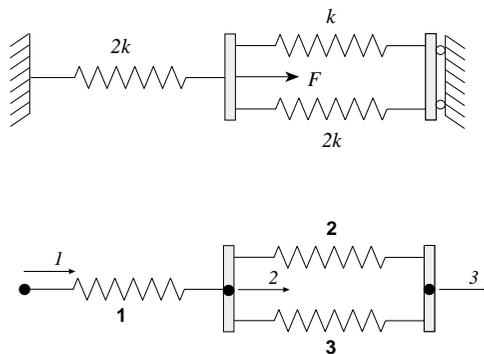
The computation is initialized by defining the topology matrix Edof, containing element numbers and global element degrees of freedom,

```
>> Edof=[1  1 2;
         2  2 3;
         3  2 3];
```

the global stiffness matrix K (3×3) of zeros,

```
>> K=zeros(3,3)

K =
     0     0     0
     0     0     0
     0     0     0
```

and the load vector f (3×1) with the load $F = 100$ in position 2.

```
>> f=zeros(3,1);  f(2)=100

f =
       0
     100
       0
```

Element stiffness matrices are generated by the function spring1e. The element property ep for the springs contains the spring stiffnesses k and $2k$ respectively, where $k = 1500$.

```
>> k=1500;  ep1=k;  ep2=2*k;
>> Ke1=spring1e(ep1)

Ke1 =
        1500         -1500
       -1500          1500

>> Ke2=spring1e(ep2)

Ke2 =
        3000         -3000
       -3000          3000
```

The element stiffness matrices are assembled into the global stiffness matrix K according to the topology.

```
>> K=assem(Edof(1,:),K,Ke2)

K =
        3000         -3000             0
       -3000          3000             0
           0             0             0

>> K=assem(Edof(2,:),K,Ke1)

K =
        3000         -3000             0
       -3000          4500         -1500
           0         -1500          1500
```

```
>> K=assem(Edof(3,:),K,Ke2)

K =
        3000        -3000            0
       -3000         7500        -4500
           0        -4500         4500
```

The global system of equations is solved considering the boundary conditions given in bc.

```
>> bc= [1 0; 3 0];
>> [a,r]=solveq(K,f,bc)

a =
           0
      0.0133
           0

r =
    -40.0000
           0
    -60.0000
```

Element forces are evaluated from the element displacements. These are obtained from the global displacements a using the function extract.

```
>> ed1=extract(Edof(1,:),a)

ed1 =
        0        0.0133

>> ed2=extract(Edof(2,:),a)

ed2 =
     0.0133        0

>> ed3=extract(Edof(3,:),a)

ed3 =
     0.0133        0
```

The spring forces are evaluated using the function spring1s.

```
>> es1=spring1s(ep2,ed1)

es1 =
      40

>> es2=spring1s(ep1,ed2)

es2 =
     -20

>> es3=spring1s(ep2,ed3)

es3 =
     -40
```

Purpose:

Analysis of one-dimensional heat flow.

Description:

Consider a wall built up of concrete and thermal insulation. The outdoor temperature is $-17°C$ and the temperature inside is $20°C$. At the inside of the theral insulation there is a heat source yielding $10\ W/m^2$.
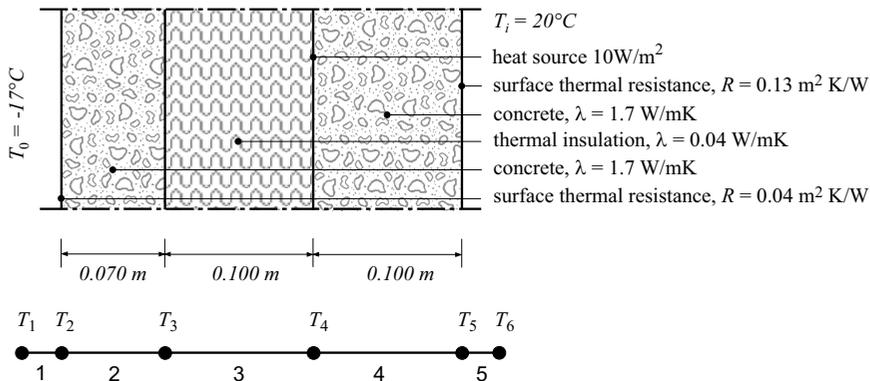
The wall is subdivided into five elements and the one-dimensional spring (analogy) element spring1e is used. Equivalent spring stiffnesses are $k_i = \lambda A/L$ for thermal conductivity and $k_i = A/R$ for thermal surface resistance. Corresponding spring stiffnesses per m^2 of the wall are:

$$
\begin{array}{lll}
k_1 = & 1/0.04 & = & 25.0 & \text{W/K} \\
k_2 = & 1.7/0.070 & = & 24.3 & \text{W/K} \\
k_3 = & 0.040/0.100 & = & 0.4 & \text{W/K} \\
k_4 = & 1.7/0.100 & = & 17.0 & \text{W/K} \\
k_5 = & 1/0.13 & = & 7.7 & \text{W/K}
\end{array}
$$

A global system matrix K and a heat flow vector f are defined. The heat source inside the wall is considered by setting $f_4 = 10$. The element matrices Ke are computed using spring1e, and the function assem assembles the global stiffness matrix.

The system of equations is solved using solveq with considerations to the boundary conditions in bc. The prescribed temperatures are $T_1 = -17°C$ and $T_6 = 20°C$.

```
>> Edof=[1  1 2
         2  2 3;
         3  3 4;
         4  4 5;
         5  5 6];
```

```
>> K=zeros(6);
>> f=zeros(6,1);  f(4)=10

f =

     0
     0
     0
    10
     0
     0

>> ep1=[25];  ep2=[24.3];
>> ep3=[0.4];  ep4=[17];
>> ep5=[7.7];

>> Ke1=spring1e(ep1);       Ke2=spring1e(ep2);
>> Ke3=spring1e(ep3);       Ke4=spring1e(ep4);
>> Ke5=spring1e(ep5);

>> K=assem(Edof(1,:),K,Ke1);   K=assem(Edof(2,:),K,Ke2);
>> K=assem(Edof(3,:),K,Ke3);   K=assem(Edof(4,:),K,Ke4);
>> K=assem(Edof(5,:),K,Ke5);

>> bc=[1 -17; 6 20];

>> [a,r]=solveq(K,f,bc)

a =

  -17.0000
  -16.4384
  -15.8607
   19.2378
   19.4754
   20.0000

r =

  -14.0394
    0.0000
   -0.0000
        0
    0.0000
    4.0394
```

The temperature values T_i in the node points are given in the vector a and the boundary flows in the vector r.

After solving the system of equations, the heat flow through the wall is computed using extract and spring1s.

```
>> ed1=extract(Edof(1,:),a);
>> ed2=extract(Edof(2,:),a);
>> ed3=extract(Edof(3,:),a);
>> ed4=extract(Edof(4,:),a);
>> ed5=extract(Edof(5,:),a);

>> q1=spring1s(ep1,ed1)

q1 =

    14.0394

>> q2=spring1s(ep2,ed2)

q2 =

    14.0394

>> q3=spring1s(ep3,ed3)

q3 =

    14.0394

>> q4=spring1s(ep4,ed4)

q4 =

     4.0394

>> q5=spring1s(ep5,ed5)

q5 =

     4.0394
```

The heat flow through the wall is $q = 14.0$ W/m^2 in the part of the wall to the left of the heat source, and $q = 4.0$ W/m^2 in the part to the right of the heat source.

Purpose:

Analysis of a plane truss.

Description:

Consider a plane truss consisting of tree bars with the properties $E = 200$ GPa, $A_1 = 6.0 \cdot 10^{-4}$ m^2, $A_2 = 3.0 \cdot 10^{-4}$ m^2 and $A_3 = 10.0 \cdot 10^{-4}$ m^2, and loaded by a single force $P = 80$ kN. The corresponding finite element model consists of three elements and eight degrees of freedom.
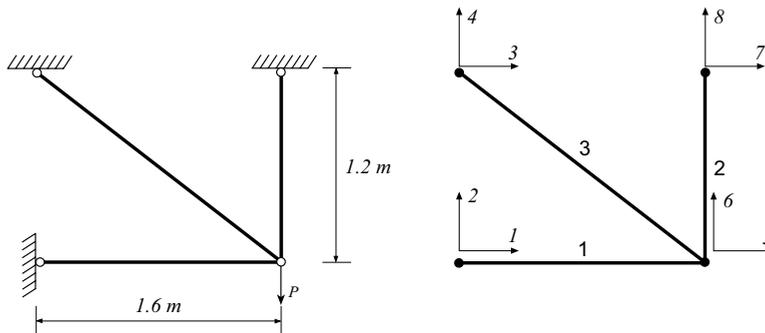
The topology is defined by the matrix

```
>> Edof=[1   1  2  5  6;
         2   5  6  7  8;
         3   3  4  5  6];
```

The stiffness matrix K and the load vector f, are defined by

```
>> K=zeros(8);
   f=zeros(8,1); f(6)=-80e3;
```

The element property vectors ep1, ep2 and ep3 are defined by

```
>> E=2.0e11;
>> A1=6.0e-4;     A2=3.0e-4;     A3=10.0e-4;
>> ep1=[E A1];    ep2=[E A2];    ep3=[E A3];
```

and the element coordinate vectors ex1, ex2, ex3, ey1, ey2 and ey3 by

```
>> ex1=[0 1.6];   ex2=[1.6 1.6];   ex3=[0 1.6];
>> ey1=[0 0];     ey2=[0 1.2];     ey3=[1.2 0];
```

The element stiffness matrices Ke1, Ke2 and Ke3 are computed using bar2e.

```
>> Ke1=bar2e(ex1,ey1,ep1)

Ke1 =

  1.0e+007 *

    7.5000          0   -7.5000          0
         0          0         0          0
   -7.5000          0    7.5000          0
         0          0         0          0

>> Ke2=bar2e(ex2,ey2,ep2)

Ke2 =

  1.0e+007 *

         0          0         0          0
         0     5.0000         0    -5.0000
         0          0         0          0
         0    -5.0000         0     5.0000

>> Ke3=bar2e(ex3,ey3,ep3)

Ke3 =

  1.0e+007 *

    6.4000    -4.8000   -6.4000    4.8000
   -4.8000     3.6000    4.8000   -3.6000
   -6.4000     4.8000    6.4000   -4.8000
    4.8000    -3.6000   -4.8000    3.6000
```

Based on the topology information, the global stiffness matrix can be generated by assembling the element stiffness matrices

```
>> K=assem(Edof(1,:),K,Ke1);
>> K=assem(Edof(2,:),K,Ke2);
>> K=assem(Edof(3,:),K,Ke3)
```

```
K =

  1.0e+008 *

  Columns 1 through 7

    0.7500          0          0          0   -0.7500          0          0
         0          0          0          0          0          0          0
         0          0     0.6400    -0.4800    -0.6400     0.4800          0
         0          0    -0.4800     0.3600     0.4800    -0.3600          0
   -0.7500          0    -0.6400     0.4800     1.3900    -0.4800          0
         0          0     0.4800    -0.3600    -0.4800     0.8600          0
         0          0          0          0          0          0          0
         0          0          0          0          0    -0.5000          0

  Column 8

         0
         0
         0
         0
         0
   -0.5000
         0
    0.5000
```

Considering the prescribed displacements in bc, the system of equations is solved using the function solveq, yielding displacements a and support forces r.

```
>> bc= [1 0;2 0;3 0;4 0;7 0;8 0];
>> [a,r]=solveq(K,f,bc)

a =

  1.0e-002 *

         0
         0
         0
         0
   -0.0398
   -0.1152
         0
         0
```

```
r =

  1.0e+004 *

    2.9845
         0
   -2.9845
    2.2383
    0.0000
    0.0000
         0
    5.7617
```

The vertical displacement at the point of loading is 1.15 mm. The section forces es1, es2 and es3 are calculated using bar2s from element displacements ed1, ed2 and ed3 obtained using extract.

```
>> ed1=extract(Edof(1,:),a);
>> N1=bar2s(ex1,ey1,ep1,ed1)

N1 =

-2.9845e+004

>> ed2=extract(Edof(2,:),a);
>> N2=bar2s(ex2,ey2,ep2,ed2)

N2 =

  5.7617e+004

>> ed3=extract(Edof(3,:),a);
>> N3=bar2s(ex3,ey3,ep3,ed3)

N3 =

  3.7306e+004
```

i.e., the normal forces are $N_1 = -29.84$ kN, $N_2 = 57.62$ kN and $N_3 = 37.31$ kN.

Purpose:

Analysis of a plane truss.

Description:

Consider a plane truss, loaded by a single force $P = 0.5$ MN.

$A = 25.0 \cdot 10^{-4} \text{ m}^2$
$E = 2.10 \cdot 10^5 \text{ MPa}$
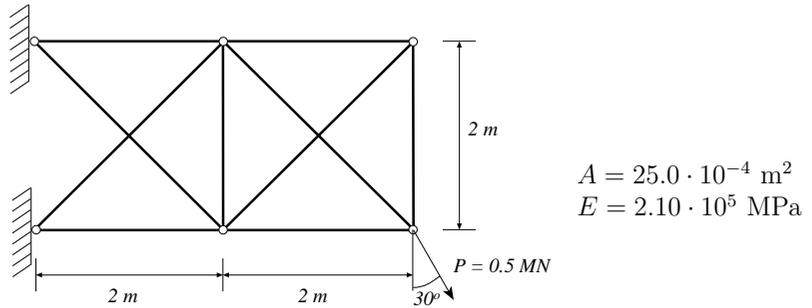
The corresponding finite element model consists of ten elements and twelve degrees of freedom.
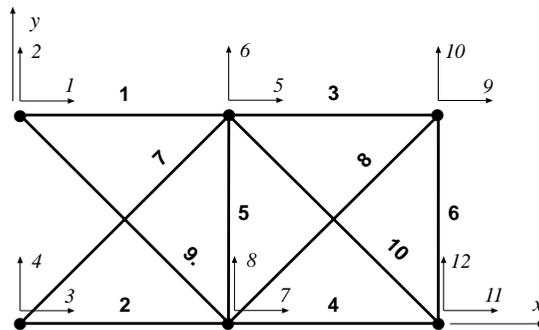
The topology is defined by the matrix

```
>> Edof=[1    1   2   5   6;
         2    3   4   7   8;
         3    5   6   9  10;
         4    7   8  11  12;
         5    7   8   5   6;
         6   11  12   9  10;
         7    3   4   5   6;
         8    7   8   9  10;
         9    1   2   7   8;
        10    5   6  11  12];
```

A global stiffness matrix K and a load vector f are defined. The load P is divided into x and y components and inserted in the load vector f.

```
>> K=zeros(12);
>> f=zeros(12,1);  f(11)=0.5e6*sin(pi/6);  f(12)=-0.5e6*cos(pi/6);
```

The element matrices Ke are computed by the function bar2e. These matrices are then assembled in the global stiffness matrix using the function assem.

```
>> A=25.0e-4;    E=2.1e11;   ep=[E A];

>> Ex=[0 2;
       0 2;
       2 4;
       2 4;
       2 2;
       4 4;
       0 2;
       2 4;
       0 2;
       2 4];

>> Ey=[2 2;
       0 0;
       2 2;
       0 0;
       0 2;
       0 2;
       0 2;
       0 2;
       2 0;
       2 0];
```

All the element matrices are computed and assembled in the loop

```
>> for i=1:10
       Ke=bar2e(Ex(i,:),Ey(i,:),ep);
       K=assem(Edof(i,:),K,Ke);
    end;
```

The displacements in a and the support forces in r are computed by solving the system of equations considering the boundary conditions in bc.

```
>> bc=[1 0;2 0;3 0;4 0];
>> [a,r]=solveq(K,f,bc)
```

```
a =

         0
         0
         0
         0
    0.0024
   -0.0045
   -0.0016
   -0.0042
    0.0030
   -0.0107
   -0.0017
   -0.0113


r =

  1.0e+005 *

   -8.6603
    2.4009
    6.1603
    1.9293
    0.0000
   -0.0000
   -0.0000
   -0.0000
    0.0000
    0.0000
    0.0000
    0.0000
```

The displacement at the point of loading is $-1.7 \cdot 10^{-3}$ m in the x-direction and $-11.3 \cdot 10^{-3}$ m in the y-direction. At the upper support the horizontal force is -0.866 MN and the vertical 0.240 MN. At the lower support the forces are 0.616 MN and 0.193 MN, respectively.

Normal forces are evaluated from element displacements. These are obtained from the global displacements a using the function extract. The normal forces are evaluated using the function bar2s.

```
 ed=extract(Edof,a);

>> for i=1:10
      N(i,:)=bar2s(Ex(i,:),Ey(i,:),ep,ed(i,:));
    end
```

The obtained normal forces are

```
>> N

N =

  1.0e+005 *

    6.2594
   -4.2310
    1.7064
   -0.1237
   -0.6945
    1.7064
   -2.7284
   -2.4132
    3.3953
    3.7105
```

The largest normal force $N = 0.626$ MN is obtained in element 1 and is equivalent to a normal stress $\sigma = 250$ MPa.

To reduce the quantity of input data, the element coordinate matrices Ex and Ey can alternatively be created from a global coordinate matrix Coord and a global topology matrix Coord using the function coordxtr, i.e.

```
>> Coord=[0 2;
          0 0;
          2 2;
          2 0;
          4 2;
          4 0];

>> Dof=[ 1  2;
         3  4;
         5  6;
         7  8;
         9 10;
        11 12];

>> [ex,ey]=coordxtr(Edof,Coord,Dof,2);
```

Purpose:

Analysis of a simply supported beam.

Description:

Consider the simply supported beam loaded by a single load $f = 10000$ N, applied at a point 1 meter from the left support. The corresponding finite element mesh is also shown. The following data apply to the beam

Young's modulus	E	$=$	$2.10 \cdot 10^{11}$	Pa
Cross section area	A	$=$	$45.3 \cdot 10^{-4}$	m^2
Moment of inertia	I	$=$	$2510 \cdot 10^{-8}$	m^4
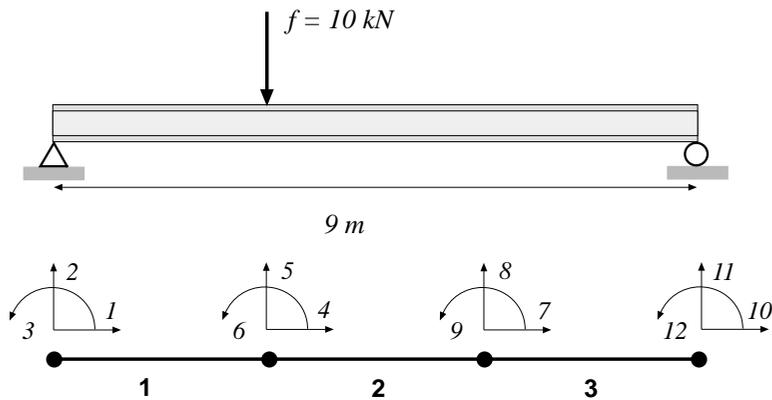
The element topology is defined by the topology matrix

```
>> Edof=[1   1  2  3  4  5  6
         2   4  5  6  7  8  9
         3   7  8  9 10 11 12];
```

The system matrices, i.e. the stiffness matrix K and the load vector f, are defined by

```
>> K=zeros(12);   f=zeros(12,1);   f(5)=-10000;
```

The element property vector ep, the element coordinate vectors ex and ey, and the element stiffness matrix Ke, are generated. Note that the same coordinate vectors are applicable for all elements because they are identical.

```
>> E=2.1e11;     A=45.3e-4;      I=2510e-8;        ep=[E A I];

>> ex=[0 3];     ey=[0 0];

>> Ke=beam2e(ex,ey,ep)

Ke =

  1.0e+008 *

    3.1710         0         0   -3.1710         0         0
         0    0.0234    0.0351         0   -0.0234    0.0351
         0    0.0351    0.0703         0   -0.0351    0.0351
   -3.1710         0         0    3.1710         0         0
         0   -0.0234   -0.0351         0    0.0234   -0.0351
         0    0.0351    0.0351         0   -0.0351    0.0703
```

Based on the topology information, the global stiffness matrix can be generated by assembling the element stiffness matrices

```
>> K=assem(Edof,K,Ke);
```

Finally, the solution can be calculated by defining the boundary conditions in bc and solving the system of equations. Displacements a and support forces r are computed by the function solveq.

```
>> bc=[1 0; 2 0; 11 0];
[a,r]=solveq(K,f,bc);
```

The section forces es are calculated from element displacements Ed

```
>> Ed=extract(Edof,a);
```

```
>> es1=beam2s(ex,ey,ep,Ed(1,:));
```

```
>> es2=beam2s(ex,ey,ep,Ed(2,:));
```

```
>> es3=beam2s(ex,ey,ep,Ed(3,:));
```

Results

```
a =                     r =
        0                       1.0e+003 *
        0
  -0.0095                               0
        0                          6.6667
  -0.0228                         -0.0000
  -0.0038                               0
        0                          0.0000
  -0.0199                         -0.0000
   0.0047                               0
        0                          0.0000
        0                          0.0000
   0.0076                               0
                                   3.3333
                                  -0.0000
```

```
es1 =

  1.0e+004 *

        0   -0.6667    0.0000
        0   -0.6667    2.0000

es2 =

  1.0e+004 *

        0    0.3333    2.0000
        0    0.3333    1.0000

es3 =

  1.0e+004 *

        0    0.3333    1.0000
        0    0.3333   -0.0000
```

Purpose:

Analysis of a plane frame.

Description:

A frame consists of one horizontal and two vertical beams according to the figure.
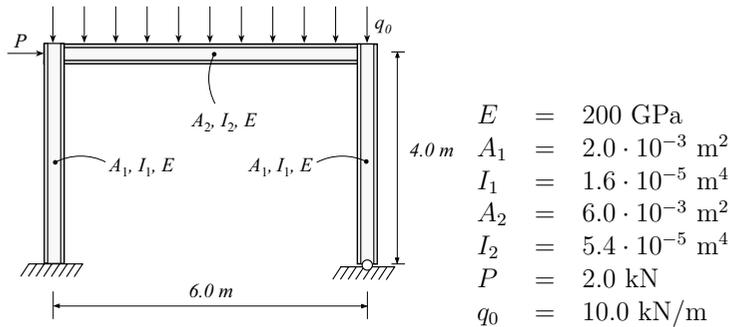
$$
\begin{array}{rcl}
E &=& 200 \text{ GPa} \\
A_1 &=& 2.0 \cdot 10^{-3} \text{ m}^2 \\
I_1 &=& 1.6 \cdot 10^{-5} \text{ m}^4 \\
A_2 &=& 6.0 \cdot 10^{-3} \text{ m}^2 \\
I_2 &=& 5.4 \cdot 10^{-5} \text{ m}^4 \\
P &=& 2.0 \text{ kN} \\
q_0 &=& 10.0 \text{ kN/m}
\end{array}
$$

The corresponding finite element model consists of three beam elements and twelve degrees of freedom.
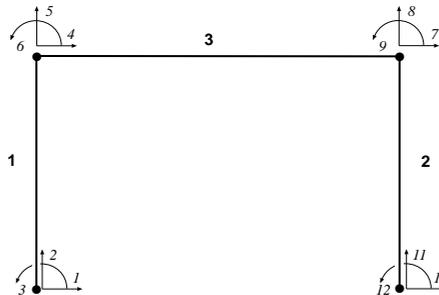
A topology matrix Edof, a global stiffness matrix K and load vector f are defined. The element matrices Ke and fe are computed by the function beam2e. These matrices are then assembled in the global matrices using the function assem.

```
>>  Edof=[1  4  5  6  1  2  3;
          2  7  8  9 10 11 12;
          3  4  5  6  7  8  9];
```

```
>>  K=zeros(12);  f=zeros(12,1);  f(4)=2e+3;
```

```
>>  E=200e9;
>>  A1=2e-3;    A2=6e-3;
>>  I1=1.6e-5;  I2=5.4e-5;
>>  ep1=[E A1 I1];  ep3=[E A2 I2];
```

```
>> ex1=[0 0];  ex2=[6 6];  ex3=[0 6];
>> ey1=[0 4];  ey2=[0 4];  ey3=[4 4];
>> eq1=[0 0];  eq2=[0 0];  eq3=[0 -10e+3];

>> Ke1=beam2e(ex1,ey1,ep1);
>> Ke2=beam2e(ex2,ey2,ep1);
>> [Ke3,fe3]=beam2e(ex3,ey3,ep3,eq3);

>> K=assem(Edof(1,:),K,Ke1);
>> K=assem(Edof(2,:),K,Ke2);
>> [K,f]=assem(Edof(3,:),K,Ke3,f,fe3);
```

The system of equations are solved considering the boundary conditions in bc.

```
>> bc=[1 0; 2 0; 3 0; 10 0; 11 0];
>> [a,r]=solveq(K,f,bc)

a =                          r =

         0                      1.0e+004 *
         0
         0                        0.1927
    0.0075                        2.8741
   -0.0003                        0.0445
   -0.0054                             0
    0.0075                        0.0000
   -0.0003                       -0.0000
    0.0047                       -0.0000
         0                             0
         0                        0.0000
   -0.0052                       -0.3927
                                  3.1259
                                       0
```

The element displacements are obtained from the function extract, and the function beam2s computes the section forces.

```
>> Ed=extract(Edof,a);
>> es1=beam2s(ex1,ey1,ep1,Ed(1,:),eq1,21)
es1 =

  1.0e+004 *

   -2.8741    0.1927    0.8152
   -2.8741    0.1927    0.7767
      .         .         .
   -2.8741    0.1927    0.0445
```

```
>> es2=beam2s(ex2,ey2,ep1,Ed(2,:),eq2,21)

es2 =

  1.0e+004 *

   -3.1259    -0.3927    -1.5707
   -3.1259    -0.3927    -1.4922
       .          .          .
   -3.1259    -0.3927    -0.0000

>> es3=beam2s(ex3,ey3,ep3,Ed(3,:),eq3,21)

es3 =

  1.0e+004 *

   -0.3927    -2.8741    -0.8152
   -0.3927    -2.5741     0.0020
       .          .          .
   -0.3927     3.1259    -1.5707
```

A displacement diagram is displayed using the function eldisp2 and section force
diagrams using the function eldia2.

```
>> figure(1)
>> plotpar=[2 1 0];
>> eldraw2(ex1,ey1,plotpar);
>> eldraw2(ex2,ey2,plotpar);
>> eldraw2(ex3,ey3,plotpar);
>> sfac=scalfact2(ex3,ey3,Ed(3,:),0.1);
>> plotpar=[1 2 1];
>> eldisp2(ex1,ey1,Ed(1,:),plotpar,sfac);
>> eldisp2(ex2,ey2,Ed(2,:),plotpar,sfac);
>> eldisp2(ex3,ey3,Ed(3,:),plotpar,sfac);
>> axis([-1.5 7.5 -0.5 5.5]);
>> pltscalb2(sfac,[1e-2 0.5 0]);
>> axis([-1.5 7.5 -0.5 5.5]);
>> title('displacements')
```

```
>> figure(2)
>> plotpar=[2 1];
>> sfac=scalfact2(ex1,ey1,es1(:,1),0.2);
>> eldia2(ex1,ey1,es1(:,1),plotpar,sfac);
>> eldia2(ex2,ey2,es2(:,1),plotpar,sfac);
>> eldia2(ex3,ey3,es3(:,1),plotpar,sfac);
>> axis([-1.5 7.5 -0.5 5.5]);
>> pltscalb2(sfac,[3e4 1.5 0]);
>> title('normal force')

>> figure(3)
>> sfac=scalfact2(ex3,ey3,es3(:,2),0.2);
>> eldia2(ex1,ey1,es1(:,2),plotpar,sfac);
>> eldia2(ex2,ey2,es2(:,2),plotpar,sfac);
>> eldia2(ex3,ey3,es3(:,2),plotpar,sfac);
>> axis([-1.5 7.5 -0.5 5.5]);
>> pltscalb2(sfac,[3e4 0.5 0]);
>> title('shear force')

>> figure(4)
>> sfac=scalfact2(ex3,ey3,es3(:,3),0.2);
>> eldia2(ex1,ey1,es1(:,3),plotpar,sfac);
>> eldia2(ex2,ey2,es2(:,3),plotpar,sfac);
>> eldia2(ex3,ey3,es3(:,3),plotpar,sfac);
>> axis([-1.5 7.5 -0.5 5.5]);
>> pltscalb2(sfac,[3e4 0.5 0]);
>> title('moment')
```
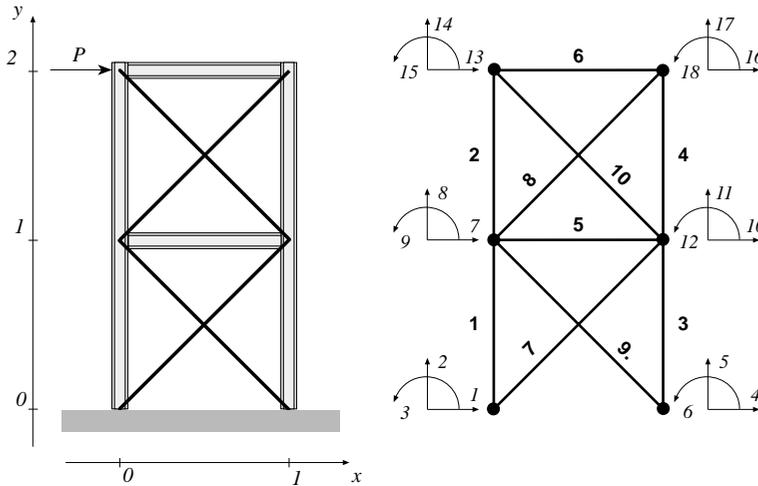
normal force

shear force

moment
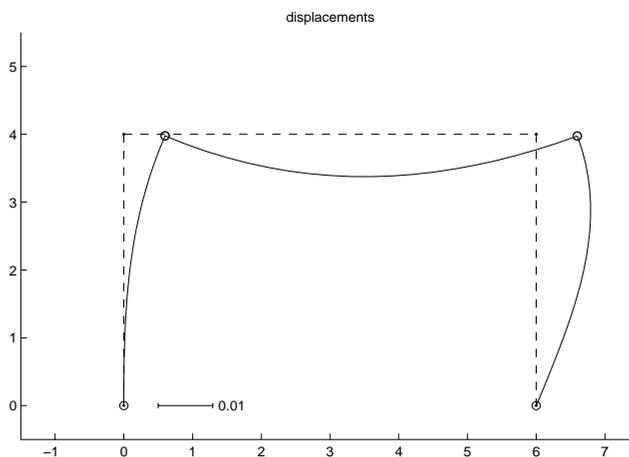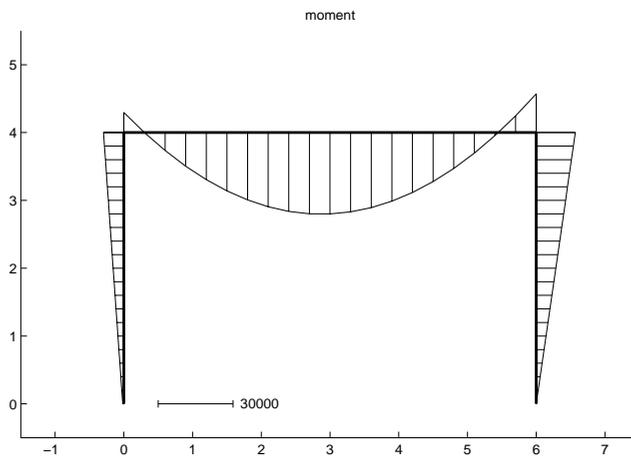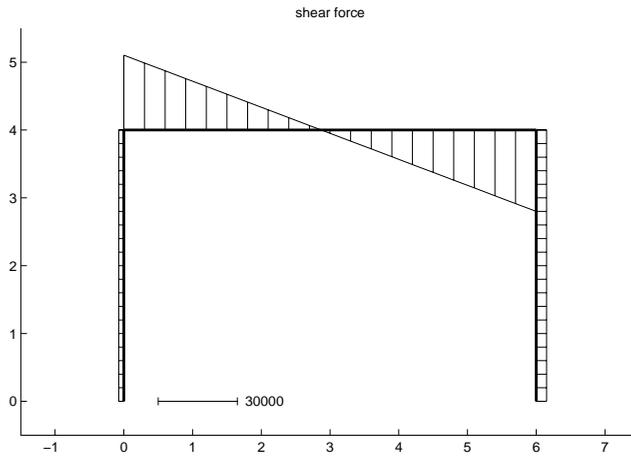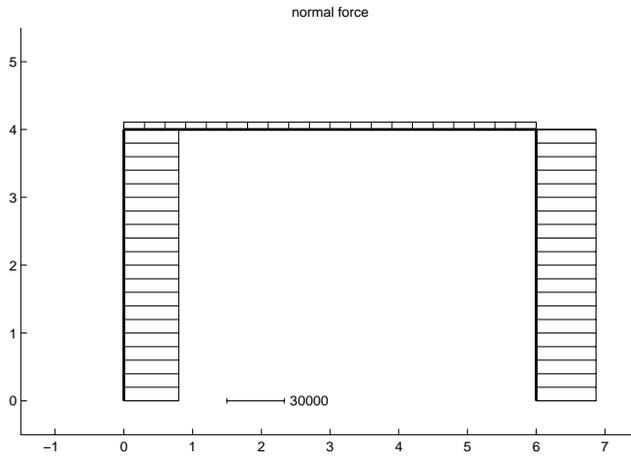
Purpose:

Set up a frame, consisting of both beams and bars, and illustrate the calculations by use of graphics functions.

Description:

A frame consists of horizontal and vertical beams, and is stabilized with diagonal bars.

The frame with its coordinates and loading is shown in the left figure, and the finite element model in the right. In the following, the statements for analysing the frame are given as an .m-file.

The matrices of the global system i.e. the stiffness matrix K, the load vector f, the coordinate matrix Coord, and the corresponding degrees of freedom matrix Dof are defined by

```
% ----- System matrices -----

K=zeros(18,18); f=zeros(18,1);  f(13)=1;

Coord=[0 0;
       1 0;
       0 1;
       1 1;
       0 2;
       1 2];
```

```
Dof=[1  2  3;
     4  5  6;
     7  8  9;
     10 11 12;
     13 14 15;
     16 17 18];
```

The material properties, the topology, and the element coordinates for the beams
and bars respectively, are defined by

```
% ----- Element properties, topology and coordinates -----

ep1=[1 1 1];
Edof1=[1   1    2   3   7    8    9;
       2   7    8   9   13   14   15;
       3   4    5   6   10   11   12;
       4   10   11  12  16   17   18;
       5   7    8   9   10   11   12;
       6   13   14  15  16   17   18];
[Ex1,Ey1]=coordxtr(Edof1,Coord,Dof,2);

ep2=[1 1];
Edof2=[7    1    2   10   11;
       8    7    8   16   17;
       9    7    8    4    5;
      10   13   14   10   11];
[Ex2,Ey2]=coordxtr(Edof2,Coord,Dof,2);
```

To check the model, the finite element mesh can be drawn.

```
eldraw2(Ex1,Ey1,[1 3 1]);
eldraw2(Ex2,Ey2,[1 2 1]);
```

The element stiffness matrices are generated and assembled in two loops, one for the
beams and one for the bars. The element stiffness functions beam2e and bar2e use
the element coordinate matrices ex and ey. These matrices are extracted from the
global coordinates Coord by the function coordxtr above.

```
% ----- Create and assemble element matrices -----

for i=1:6
  Ke=beam2e(Ex1(i,:),Ey1(i,:),ep1);
  K=assem(Edof1(i,:),K,Ke);
end

for i=1:4
  Ke=bar2e(Ex2(i,:),Ey2(i,:),ep2);
  K=assem(Edof2(i,:),K,Ke);
end
```

The global system of equations is solved considering the boundary conditions in bc,

```
% ----- Solve equation system -----

bc= [1 0; 2 0; 3 0; 4 0; 5 0; 6 0]; [a,r]=solveq(K,f,bc);
```

and the deformed frame is displayed by the function eldisp2, where the displacements are scaled by the variable sfac.

```
Ed1=extract(Edof1,a);
Ed2=extract(Edof2,a);

[sfac]=scalfact2(Ex1,Ey1,Ed1,0.1);
eldisp2(Ex1,Ey1,Ed1,[2 1 1],sfac);
eldisp2(Ex2,Ey2,Ed2,[2 1 1],sfac);
```
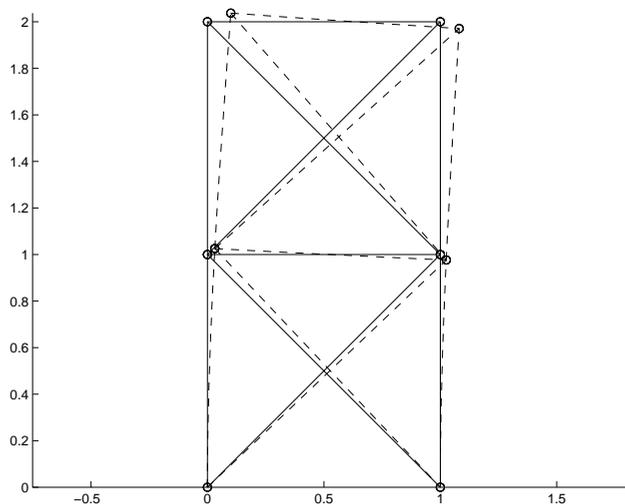
Purpose:

Analysis of two dimensional diffusion.

Description:
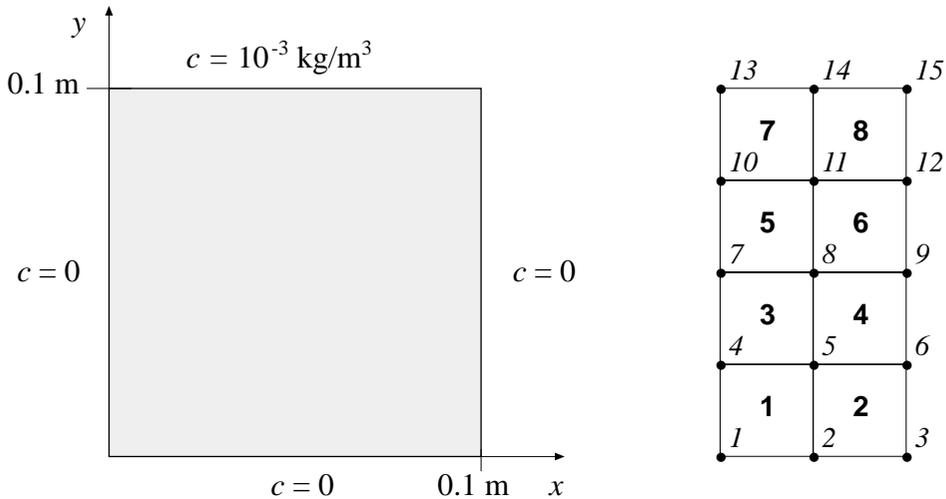
Description:

Consider a filter paper of square shape. Three sides are in contact with pure water and the fourth side is in contact with a solution of concentration $c = 1.0 \cdot 10^{-3}$ kg/m^3. The length of each side is 0.100 m. Using symmetry, only half of the paper has to be analyzed. The paper and the corresponding finite element mesh are shown. The following boundary conditions are applied

$c(0, y) = c(x, 0) = c(0.1, y) = 0$
$c(x, 0.1) = 10^{-3}$

The element topology is defined by the topology matrix

```
>> Edof=[1    1  2  5  4
         2    2  3  6  5
         3    4  5  8  7
         4    5  6  9  8
         5    7  8 11 10
         6    8  9 12 11
         7   10 11 14 13
         8   11 12 15 14];
```

The system matrices, i.e. the stiffness matrix K and the load vector f, are defined by

```
>> K=zeros(15);    f=zeros(15,1);
```

Because of the same geometry, orientation, and constitutive matrix for all elements, only one element stiffness matrix Ke has to be computed. This is done by the function flw2qe.

```
>> ep=1;     D=[1 0; 0 1];

>> ex=[0 0.025 0.025 0];     ey=[0 0 0.025 0.025];

>> Ke=flw2qe(ex,ey,ep,D)

>> Ke =

    0.7500   -0.2500   -0.2500   -0.2500
   -0.2500    0.7500   -0.2500   -0.2500
   -0.2500   -0.2500    0.7500   -0.2500
   -0.2500   -0.2500   -0.2500    0.7500
```

Based on the topology information, the global stiffness matrix is generated by assembling this element stiffness matrix Ke in the global stiffness matrix K

```
>> K=assem(Edof,K,Ke);
```

Finally, the solution is calculated by defining the boundary conditions bc and solving the system of equations. The boundary condition at dof 13 is set to $0.5 \cdot 10^{-3}$ as an average of the concentrations at the neighbouring boundaries. Concentrations a and unknown boundary flows r are computed by the function solveq.

```
>> bc=[1 0;2 0;3 0;4 0;7 0;10 0;13 0.5e-3;14 1e-3;15 1e-3];

>> [a,r]=solveq(K,f,bc);
```

The element flows q are calculated from element concentration Ed

```
>> Ed=extract(Edof,a);

>> for i=1:8
     Es=flw2qs(ex,ey,ep,D,Ed(i,:));
   end
```

Results

```
a=                      r=
  1.0e-003 *              1.0e-003 *

        0                 -0.0165
        0                 -0.0565
        0                 -0.0399
        0                 -0.0777
   0.0662                  0.0000
   0.0935                       0
        0                 -0.2143
   0.1786                  0.0000
   0.2500                  0.0000
        0                 -0.6366
   0.4338                  0.0000
   0.5494                 -0.0000
   0.5000                  0.0165
   1.0000                  0.7707
   1.0000                  0.2542


  Es =
     -0.0013   -0.0013
     -0.0005   -0.0032
     -0.0049   -0.0022
     -0.0020   -0.0054
     -0.0122   -0.0051
     -0.0037   -0.0111
     -0.0187   -0.0213
     -0.0023   -0.0203
```

The following .m-file shows an alternative set of commands to perform the diffusion analysis of exs8. By use of global coordinates, an FE-mesh is generated. Also plots with flux-vectors and contour lines are created.

```
% ----- System matrices -----

K=zeros(15);  f=zeros(15,1);
Coord=[0     0    ; 0.025 0     ; 0.05  0
       0     0.025; 0.025 0.025; 0.05  0.025
       0     0.05 ; 0.025 0.05 ; 0.05  0.05
       0     0.075; 0.025 0.075; 0.05  0.075
       0     0.1  ; 0.025 0.1  ; 0.05  0.1  ];
Dof=[1; 2; 3
     4; 5; 6
     7; 8; 9
    10;11;12
    13;14;15];

% ----- Element properties, topology and coordinates -----

ep=1; D=[1 0;0 1];
Edof=[1    1  2  5  4
      2    2  3  6  5
      3    4  5  8  7
      4    5  6  9  8
      5    7  8 11 10
      6    8  9 12 11
      7   10 11 14 13
      8   11 12 15 14];
[Ex,Ey]=coordxtr(Edof,Coord,Dof,4);

% ----- Generate FE-mesh -----

eldraw2(Ex,Ey,[1 3 0],Edof(:,1));
pause; clf;

% ----- Create and assemble element matrices -----

for i=1:8
  Ke=flw2qe(Ex(i,:),Ey(i,:),ep,D);
  K=assem(Edof(i,:),K,Ke);
end;

% ----- Solve equation system -----

bc=[1 0;2 0;3 0;4 0;7 0;10 0;13 0.5e-3;14 1e-3;15 1e-3];
```

```
    [a,r]=solveq(K,f,bc)

% ----- Compute element flux vectors -----

  Ed=extract(Edof,a);
  for i=1:8
    Es(i,:)=flw2qs(Ex(i,:),Ey(i,:),ep,D,Ed(i,:))
  end

% ----- Draw flux vectors and contour lines -----
  sfac=scalfact2(Ex,Ey,Es,0.5);
  eldraw2(Ex,Ey,[1,3,0]);
  elflux2(Ex,Ey,Es,[1,4],sfac);
  pltscalb2(sfac,[2e-2 0.06 0.01],4);
  pause; clf;
  eldraw2(Ex,Ey,[1,3,0]);
  eliso2(Ex,Ey,Ed,5,[1,4]);
```
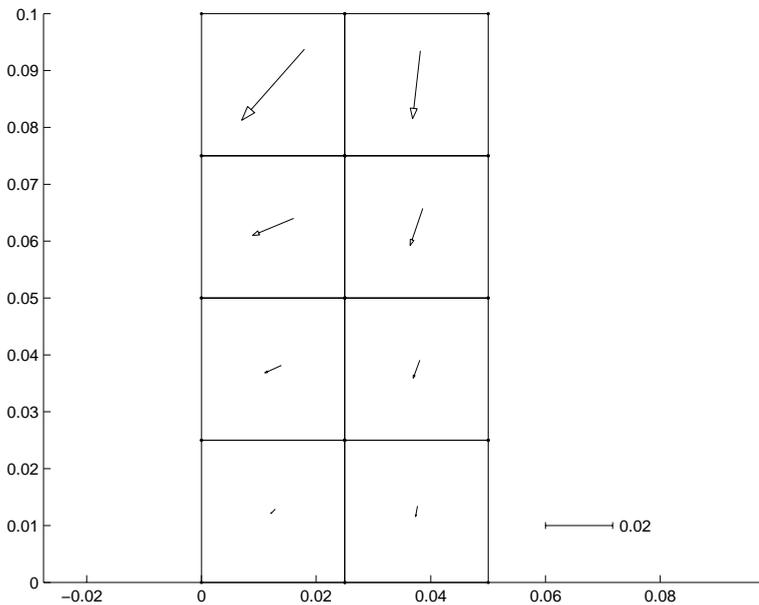

Flux vectors
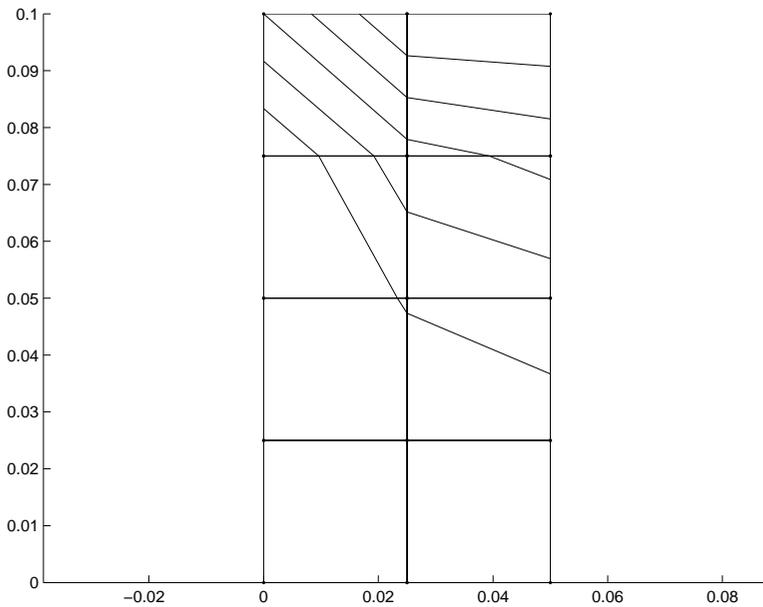
Contour lines

Two comments concerning the contour lines:

In the upper left corner, the contour lines should physically have met at the corner point. However, the drawing of the contour lines for the planqe element follows the numerical approximation along the element boundaries, i.e. a linear variation. A finer element mesh will bring the contour lines closer to the corner point.

Along the symmetry line, the contour lines should physically be perpendicular to the boundary. This will also be improved with a finer element mesh.

With the MATLAB functions colormap and fill a color plot of the concentrations can be obtained.

```
colormap('jet')
fill(Ex',Ey',Ed')
axis equal
```

9.4 Dynamic analysis

This section concerns linear dynamic finite element calculations. The examples illustrate some basic features in dynamics such as modal analysis and time stepping procedures.

Dynamic analysis	
exd1	Modal analysis of frame
exd2	Transient analysis
exd3	Reduced system transient analysis
exd4	Time varying boundary condition

Note: The examples listed above are supplied as .m-files under the directory examples. The example files are named according to the table.

Purpose:

Set up the finite element model and perform eigenvalue analysis for a simple frame structure.

Description:

Consider the two dimensional frame shown below. A vertical beam is fixed at its lower end, and connected to a horizontal beam at its upper end. The horizontal beam is simply supported at the right end. The length of the vertical beam is 3 m and of the horizontal beam 2 m. The following data apply to the beams

	vertical beam	horizontal beam
Young's modulus (N/m^2)	$3 \cdot 10^{10}$	$3 \cdot 10^{10}$
Cross section area (m^2)	$0.1030 \cdot 10^{-2}$	$0.0764 \cdot 10^{-2}$
Moment of inertia (m^4)	$0.171 \cdot 10^{-5}$	$0.0801 \cdot 10^{-5}$
Density (kg/m^3)	2500	2500

a)   b)
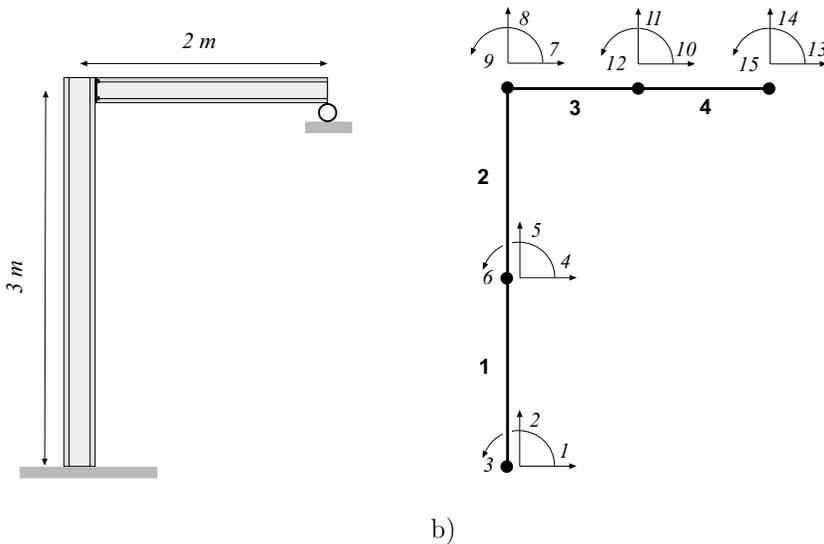
The structure is divided into 4 elements. The numbering of elements and degrees-of-freedom are apparent from the figure. The following .m-file defines the finite element model.

```
% --- material data ----------------------------------------
E=3e10;               rho=2500;
Av=0.1030e-2;         Iv=0.0171e-4;                 % IPE100
Ah=0.0764e-2;         Ih=0.00801e-4;                % IPE80
epv=[E Av Iv rho*Av]; eph=[E Ah Ih rho*Ah];
```

```
% ---  topology -------------------------------------------
Edof=[1    1  2  3  4  5  6
      2    4  5  6  7  8  9
      3    7  8  9 10 11 12
      4   10 11 12 13 14 15];
% --- list of coordinates  -------------------------------
Coord=[0 0; 0 1.5; 0 3; 1 3; 2 3];
% --- list of degrees-of-freedom  -----------------------
Dof=[1 2 3;  4 5 6;  7 8 9;  10 11 12;  13 14 15];
% --- generate element matrices, assemble in global matrices -
K=zeros(15);   M=zeros(15);
[Ex,Ey]=coordxtr(Edof,Coord,Dof,2);
for i=1:2
  [k,m,c]=beam2d(Ex(i,:),Ey(i,:),epv);
  K=assem(Edof(i,:),K,k);    M=assem(Edof(i,:),M,m);
end
for i=3:4
  [k,m,c]=beam2d(Ex(i,:),Ey(i,:),eph);
  K=assem(Edof(i,:),K,k);    M=assem(Edof(i,:),M,m);
end
```

The finite element mesh is plotted, using the following commands

```
clf;
eldraw2(Ex,Ey,[1 2 2],Edof);
grid; title('2D Frame Structure');
pause;
```
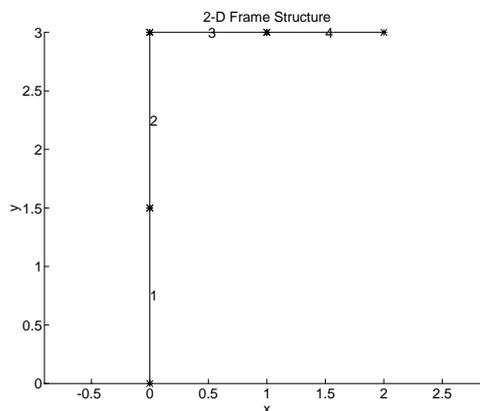

Finite element mesh

A standard procedure in dynamic analysis is eigenvalue analysis. This is accomplished by the following set of commands.

```
b=[1 2 3 14]';
[La,Egv]=eigen(K,M,b);
Freq=sqrt(La)/(2*pi);
```

Note that the boundary condition matrix, b, only lists the degrees-of-freedom that are zero. The results of these commands are the eigenvalues, stored in La, and the eigenvectors, stored in Egv. The corresponding frequencies in Hz are calculated and stored in the column matrix Freq.

$$\mathsf{Freq} = [6.9826 \quad 43.0756 \quad 66.5772 \quad 162.7453 \quad 230.2709 \quad 295.6136$$
$$426.2271 \quad 697.7628 \quad 877.2765 \quad 955.9809 \quad 1751.3]^T$$

The eigenvectors can be plotted by entering the commands below.

```
figure(1),    clf,     grid,    title('The first eigenmode'),
eldraw2(Ex,Ey,[2 3 1]);
Edb=extract(Edof,Egv(:,1));    eldisp2(Ex,Ey,Edb,[1 2 2]);
FreqText=num2str(Freq(1));    text(.5,1.75,FreqText);
pause;
```
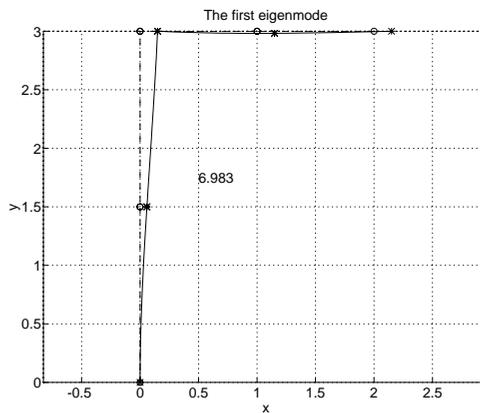

The first eigenmode, 6.98 Hz

An attractive way of displaying the eigenmodes is shown in the figure below. The result is accomplished by translating the different eigenmodes in the x-direction, see the Ext-matrix defined below, and in the y-direction, see the Eyt-matrix.

```
clf, axis('equal'), hold on, axis off
sfac=0.5;
title('The first eight eigenmodes (Hz)' )
for i=1:4;
  Ext=Ex+(i-1)*3;            eldraw2(Ext,Ey,[2 3 1]);
  Edb=extract(Edof,Egv(:,i));
  eldisp2(Ext,Ey,Edb,[1 2 2],sfac);
  FreqText=num2str(Freq(i));  text(3*(i-1)+.5,1.5,FreqText);
end;
Eyt=Ey-4;
for i=5:8;
  Ext=Ex+(i-5)*3;            eldraw2(Ext,Eyt,[2 3 1]);
  Edb=extract(Edof,Egv(:,i));
  eldisp2(Ext,Eyt,Edb,[1 2 2],sfac);
  FreqText=num2str(Freq(i));  text(3*(i-5)+.5,-2.5,FreqText);
end
```

The first eight eigenmodes (Hz)
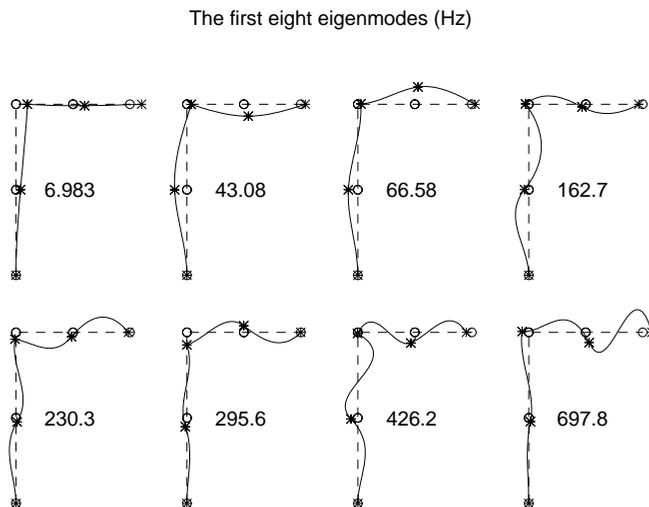
The first eight eigenmodes. Frequencies are given in Hz.

Purpose:

The frame structure defined in exd1 is exposed in this example to a transient load. The structural response is determined by a time stepping procedure.

Description:

The structure is exposed to a transient load, impacting on the center of the vertical beam in horizontal direction, i.e. at the 4th degree-of-freedom. The time history of the load is shown below. The result shall be displayed as time history plots of the 4th degree-of-freedom and the 11th degree-of-freedom. At time $t = 0$ the frame is at rest. The timestep is chosen as $\Delta t = 0.001$ seconds and the integration is performed for $T = 1.0$ second. At every 0.1 second the deformed shape of the whole structure shall be displayed.
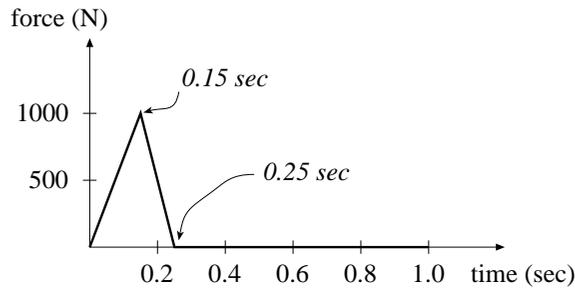
Time history of the impact load

The load is generated using the **gfunc**-function. The time integration is performed by the **step2**-function. Because there is no damping present, the C-matrix is entered as [ ].

```
dt=0.005;    T=1;
% --- the load ----------------------------------------------
G=[0 0; 0.15 1; 0.25 0; T 0];   [t,g]=gfunc(G,dt);
f=zeros(15, length(g));      f(4,:)=1000*g;
% --- boundary condition, initial condition ------------------
bc=[1 0; 2 0; 3 0; 14 0];
d0=zeros(15,1);              v0=zeros(15,1);
% --- output parameters --------------------------------------
ntimes=[0.1:0.1:1];          nhist=[4 11];
% --- time integration parameters ----------------------------
ip=[dt T 0.25 0.5 10 2 ntimes nhist];
% --- time integration ---------------------------------------
k=sparse(K);                 m=sparse(M);
[Dsnap,D,V,A]=step2(k,[],m,d0,v0,ip,f,bc);
```

The requested time history plots are generated by the following commands

```
figure(1), plot(t,D(1,:),'-',t,D(2,:),'--')
grid, xlabel('time (sec)'), ylabel('displacement (m)')
title('Displacement(time) for the 4th and 11th'...
      ' degree-of-freedom')
text(0.3,0.009,'solid line = impact point, x-direction')
text(0.3,0.007,'dashed line = center, horizontal beam,'...
                ' y-direction')
```
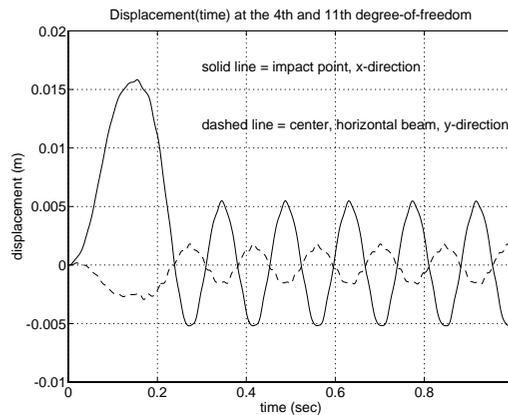

Time history at DOF 4 and DOF 11.

The deformed shapes at time increment 0.1 sec are stored in Dsnap. They are visualized by the following commands:

```
figure(2),clf, axis('equal'), hold on, axis off
sfac=25;
title('Snapshots (sec), magnification = 25');
for i=1:5;
  Ext=Ex+(i-1)*3;           eldraw2(Ext,Ey,[2 3 0]);
  Edb=extract(Edof,Dsnap(:,i));
  eldisp2(Ext,Ey,Edb,[1 2 2],sfac);
  Time=num2str(ntimes(i));   text(3*(i-1)+.5,1.5,Time);
end;


Eyt=Ey-4;
for i=6:10;
  Ext=Ex+(i-6)*3;           eldraw2(Ext,Eyt,[2 3 0]);
  Edb=extract(Edof,Dsnap(:,i));
  eldisp2(Ext,Eyt,Edb,[1 2 2],sfac);
  Time=num2str(ntimes(i));   text(3*(i-6)+.5,-2.5,Time);
end
```

Snapshots (sec), magnification = 25
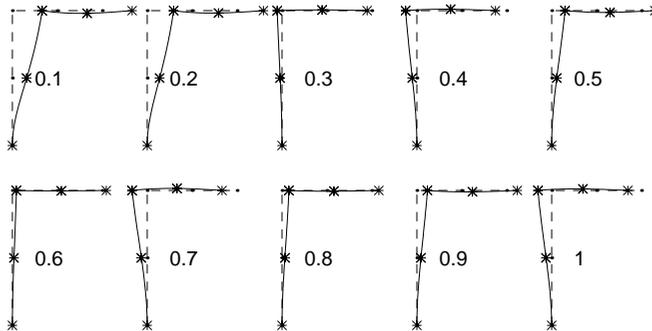
Snapshots of the deformed geometry for every 0.1 sec.

Purpose:

This example concerns reduced system analysis for the frame structure defined in exd1. Transient analysis on modal coordinates is performed for the reduced system.

Description:

In the previous example the transient analysis was based on the original finite element model. Transient analysis can also be employed on some type of reduced system, commonly a subset of the eigenvectors. The commands below pick out the first two eigenvectors for a subsequent time integration, see constant nev. The result in the figure below shall be compared to the result in exd2.
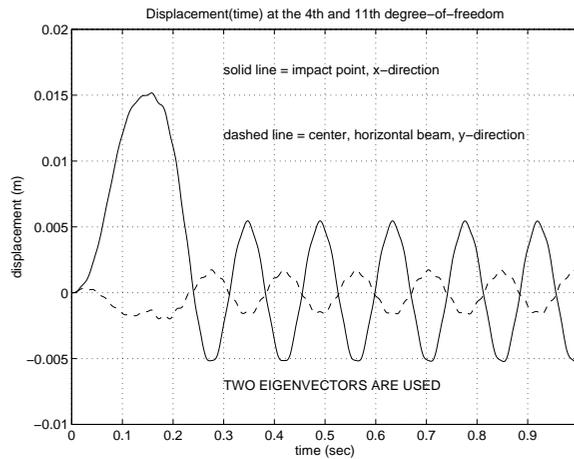
Time history at DOF 4 and DOF 11 using two eigenvectors.

```
dt=0.002;    T=1;    nev=2;
% --- the load ------------------------------------------------
G=[0 0; 0.15 1; 0.25 0; T 0];         [t,g]=gfunc(G,dt);
f=zeros(15, length(g));               f(4,:)=9000*g;
fr=sparse([[1:1:nev]' Egv(:,1:nev)'*f]);
% --- reduced system matrices ---------------------------------
kr=sparse(diag(diag(Egv(:,1:nev)'*K*Egv(:,1:nev))));
mr=sparse(diag(diag(Egv(:,1:nev)'*M*Egv(:,1:nev))));
% --- initial condition ---------------------------------------
dr0=zeros(nev,1);                     vr0=zeros(nev,1);
% --- output parameters ---------------------------------------
ntimes=[0.1:0.1:1];                   nhistr=[1:1:nev];
% --- time integration parameters -----------------------------
ip=[dt T 0.25 0.5 10 nev ntimes nhistr];
% --- time integration ----------------------------------------
[Dsnapr,Dr,Vr,Ar]=step2(kr,[],mr,dr0,vr0,ip,fr,[]);
% --- mapping back to original coordinate system -------------
```

```
DsnapR=Egv(:,1:nev)*Dsnapr;          DR=Egv(nhist,1:nev)*Dr;
% --- plot time history for two DOF:s ----------------------
figure(1), plot(t,DR(1,:),'-',t,DR(2,:),'--')
axis([0    1.0000   -0.0100    0.0200])
grid, xlabel('time (sec)'), ylabel('displacement (m)')
title('Displacement(time) at the 4th and 11th'...
      ' degree-of-freedom')
text(0.3,0.017,'solid line = impact point, x-direction')
text(0.3,0.012,'dashed line = center, horizontal beam,'...
                ' y-direction')
text(0.3,-0.007,'2 EIGENVECTORS ARE USED')
```

Purpose:

This example deals with a time varying boundary condition and time integration for the frame structure defined in exd1.

Description:

Suppose that the support of the vertical beam is moving in the horizontal direction. The commands below prepare the model for time integration. Note that the structure of the boundary condition matrix bc differs from the structure of the load matrix f defined in exd2.
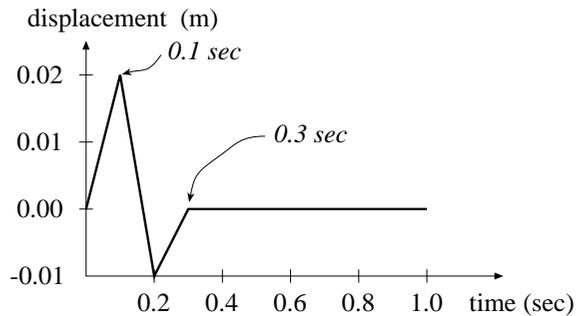
Time dependent boundary condition at the support, DOF 1.

```
dt=0.002;    T=1;
% --- boundary condition, initial condition -----------------
G=[0 0; 0.1 0.02; 0.2 -0.01; 0.3 0.0; T 0]; [t,g]=gfunc(G,dt);
bc=zeros(4, 1 + length(g));
bc(1,:)=[1 g];  bc(2,1)=2;  bc(3,1)=3;  bc(4,1)=14;
d0=zeros(15,1);              v0=zeros(15,1);
% --- output parameters ------------------------------------
ntimes=[0.1:0.1:1];          nhist=[1 4 11];
% --- time integration parameters --------------------------
ip=[dt T 0.25 0.5 10 3 ntimes nhist];
% --- time integration -------------------------------------
k=sparse(K);                 m=sparse(M);
[Dsnap,D,V,A]=step2(k,[],m,d0,v0,ip,[],bc);
% --- plot time history for two DOF:s ----------------------
figure(1), plot(t,D(1,:),'-',t,D(2,:),'--',t,D(3,:),'-.')
grid, xlabel('time (sec)'), ylabel('displacement (m)')
title('Displacement(time) at the 1st, 4th and 11th'...
      ' degree-of-freedom')
text(0.2,0.022,'solid line = bottom, vertical beam,'...
                ' x-direction')
text(0.2,0.017,'dashed line = center, vertical beam,'...
                ' x-direction')
```

```
text(0.2,0.012,'dashed-dotted line = center,'...
                 ' horizontal beam, y-direction')
% --- plot displacement for some time increments ------------
figure(2),clf, axis('equal'), hold on, axis off
sfac=20;
title('Snapshots (sec), magnification = 20'); for i=1:5;
  Ext=Ex+(i-1)*3;                eldraw2(Ext,Ey,[2 3 0]);
  Edb=extract(Edof,Dsnap(:,i));
  eldisp2(Ext,Ey,Edb,[1 2 2],sfac);
  Time=num2str(ntimes(i));   text(3*(i-1)+.5,1.5,Time);
end;
Eyt=Ey-4;
for i=6:10;
  Ext=Ex+(i-6)*3;                eldraw2(Ext,Eyt,[2 3 0]);
  Edb=extract(Edof,Dsnap(:,i));
  eldisp2(Ext,Eyt,Edb,[1 2 2],sfac);
  Time=num2str(ntimes(i));   text(3*(i-6)+.5,-2.5,Time);
end
```
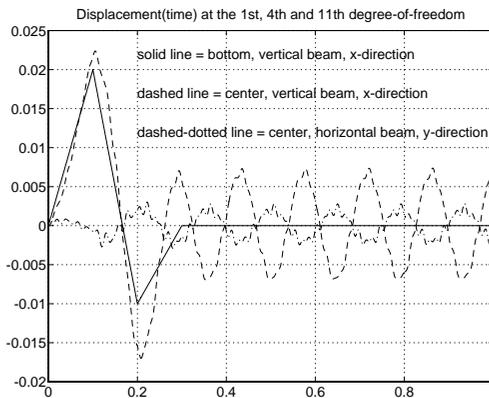

Time history at DOF 1, DOF 4 and DOF 11.

Snapshots (sec), magnification = 20
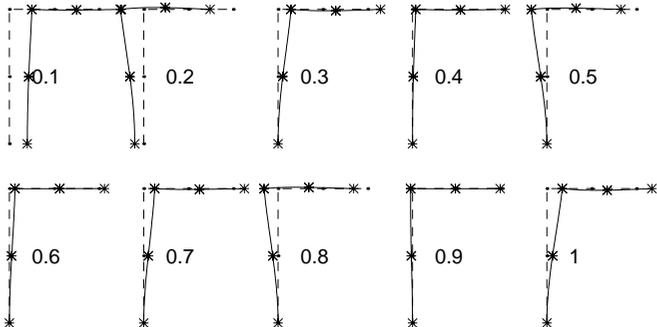
Snapshots of the deformed geometry for every 0.1 sec.

9.5 Nonlinear analysis

This section illustrates some nonlinear finite element calculations.

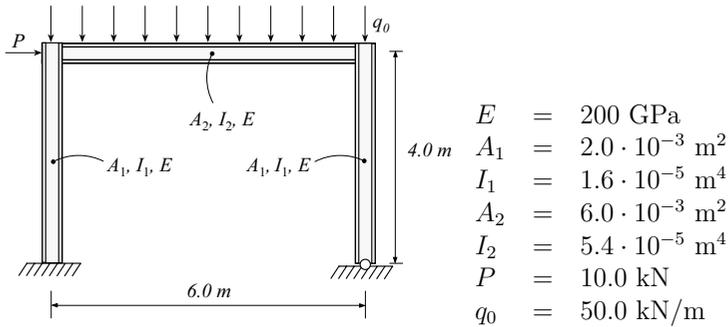| Nonlinear analysis	
exN1	Second order theory analysis of a frame
exN2	Buckling analysis of a frame

Note: The examples listed above are supplied as .m-files under the directory examples. The example files are named according to the table.

Purpose:

    Analysis of a plane frame using second order theory.

Description:

    The frame of exs6 is analysed again, but it is now subjected to a load five times larger than in exs6. Second order theory is used.

$$
\begin{array}{rcl}
E & = & 200 \text{ GPa} \\
A_1 & = & 2.0 \cdot 10^{-3} \text{ m}^2 \\
I_1 & = & 1.6 \cdot 10^{-5} \text{ m}^4 \\
A_2 & = & 6.0 \cdot 10^{-3} \text{ m}^2 \\
I_2 & = & 5.4 \cdot 10^{-5} \text{ m}^4 \\
P & = & 10.0 \text{ kN} \\
q_0 & = & 50.0 \text{ kN/m}
\end{array}
$$

The element model consisting of three beam elements and twelve degrees of freedom is repeated here.
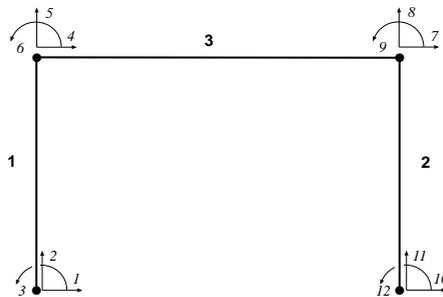
The following .m-file defines the finite element model.

```
% ----- Topology -----

 Edof=[1  4  5  6  1  2  3 ;
       2  7  8  9 10 11 12;
       3  4  5  6  7  8  9];
```

```
% ----- Element properties and global coordinates -----

 E=200e9;
 A1=2e-3;        A2=6e-3;
 I1=1.6e-5;      I2=5.4e-5;
 ep1=[E A1 I1];  ep3=[E A2 I2];
 eq3=[-50e3];

 Ex=[0 0;6 6;0 6];   Ey=[4 0;4 0;4 4];
```

The beam element function of the second order theory **beam2g** requires a normal
force as input variable. In the first iteration this normal force is chosen to zero.
This means that the first iteration is equivalent to a linear first order analysis using
beam2e. Since the normal forces are not known initially, an iterative procedure has
to be applied, where the normal forces **N** are updated according to the results of
the former iteration. The iterations continue until the difference in normal force of
the two last iteration steps is less than an accepted error **eps**, $(N-N0)/N0 <$ **eps**.
The small value given to the initial normal force $N(1)$ is to avoid division by zero
in the second convergence check. If **N** does not converge in 20 steps the analysis is
interrupted.

```
% ----- Initial values for the iteration -----

eps=0.0001;        % Error norm
N=[0.01 0 0];      % Initial normal forces
N0=[1 1 1];        % Normal forces of the initial former iteration
n=0;               % Iteration counter

% ----- Iteration procedure -----

while(abs((N(1)-N0(1))/N0(1)) > eps)
  n=n+1;

  K=zeros(12,12);
  f=zeros(12,1);
  f(4)=10e3;

  Ke1=beam2g(Ex(1,:),Ey(1,:),ep1,N(1));
  Ke2=beam2g(Ex(2,:),Ey(2,:),ep1,N(2));
  [Ke3,fe3]=beam2g(Ex(3,:),Ey(3,:),ep3,N(3),eq3);

  K=assem(Edof(1,:),K,Ke1);
  K=assem(Edof(2,:),K,Ke2);
  [K,f]=assem(Edof(3,:),K,Ke3,f,fe3);
```

```
    bc=[1 0;2 0;3 0;10 0;11 0];
    [a,r]=solveq(K,f,bc)

    Ed=extract(Edof,a);

    es1=beam2gs(Ex(1,:),Ey(1,:),ep1,Ed(1,:),N(1))
    es2=beam2gs(Ex(2,:),Ey(2,:),ep1,Ed(2,:),N(2))
    es3=beam2gs(Ex(3,:),Ey(3,:),ep3,Ed(3,:),N(3),eq3)

    N0=N;
    N=[es1(1,1) es2(1,1) es3(1,1)];

    if (n>20)
      disp('The solution doesn''t converge')
      return
    end
  end
end
```

Displacements and element forces from the linear elastic analysis and from the second order theory analysis respectively:

```
a =                                         a =

        0                                           0
        0                                           0
        0                                           0
   0.0377                                      0.0452
  -0.0014                                     -0.0014
  -0.0269                                     -0.0281
   0.0376                                      0.0451
  -0.0016                                     -0.0016
   0.0233                                      0.0239
        0                                           0
        0                                           0
  -0.0258                                     -0.0296
```

```
es1 =                                 es1 =

  1.0e+005 *                            1.0e+005 *

  -1.4370     0.0963     0.4076         -1.4241     0.0817     0.3428
  -1.4370     0.0963     0.0223         -1.4241     0.0817     0.0803


es2 =                                 es2 =

  1.0e+005 *                            1.0e+005 *

  -1.5630    -0.1963    -0.7854         -1.5759    -0.1817    -0.7980
  -1.5630    -0.1963     0.0000         -1.5759    -0.1817    -0.0000


es3 =                                 es3 =

  1.0e+005 *                            1.0e+005 *

  -0.1963    -1.4370    -0.4076         -0.1817    -1.4241    -0.3428
  -0.1963     1.5630    -0.7854         -0.1817     1.5759    -0.7980
```

Using the second order theory, the horizontal displacement of the upper left corner of the frame increases from 37.7 to 45.2 mm. The moment in the lower left corner increases from 2.2 kNm to 8.0 kNm.

Purpose:

Buckling analysis of a plane frame.

Description:

The frame of exn1 is in this example analysed with respect to security against buckling for a case when all loads are increased proportionally. The initial load distribution is increased by a loading factor alpha until buckling occurs, i.e. the determinant of the stiffness matrix K passes zero, or the solution does not converge. For each value of alpha a second order theory calculation of type exn1 is performed. The horizontal displacement a_4 and the moment M_A are plotted against alpha. The shape of the buckling mode is also plotted using the last computed displacement vector before buckling occurs.

$$
\begin{aligned}
E &= 200 \text{ GPa} \\
A_1 &= 2.0 \cdot 10^{-3} \text{ m}^2 \\
I_1 &= 1.6 \cdot 10^{-5} \text{ m}^4 \\
A_2 &= 6.0 \cdot 10^{-3} \text{ m}^2 \\
I_2 &= 5.4 \cdot 10^{-5} \text{ m}^4 \\
H &= 1.0 \text{ kN} \\
P &= 150.0 \text{ kN}
\end{aligned}
$$
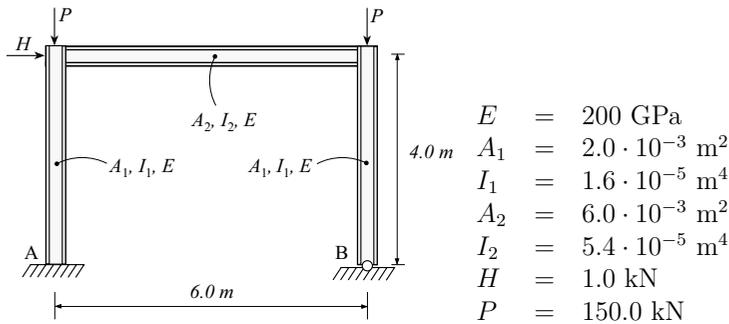
The element model consists of three beam elements and twelve degrees of freedom.

The following .m-file defines the finite element model.

```
% ----- Topology -----

 Edof=[1  4  5  6  1  2  3 ;
       2  7  8  9 10 11 12;
       3  4  5  6  7  8  9];
```
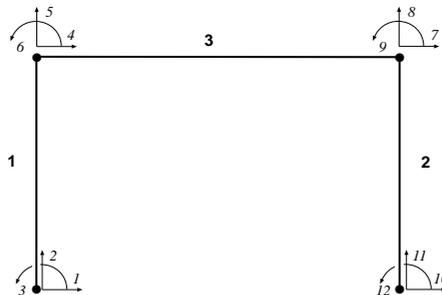
```
% ----- Element properties and global coordinates -----

 E=200e9;
 A1=2e-3;    A2=6e-3;
 I1=1.6e-5;  I2=5.4e-5;
 ep1=[E A1 I1];  ep3=[E A2 I2];

 Ex=[0 0;6 6;0 6];   Ey=[4 0;4 0;4 4];

% ----- Initial loads -----

 f0=zeros(12,1);
 f0(4)=1e3;   f0(5)=-150e3;   f0(8)=-150e3;

% ----- Increase loads until det(K)=0 -----

 j=0;
 for alpha=1:0.2:10
   j=j+1;
   N=[0.01 0 0];
   N0=[1 1 1];

% ----- Iteration for convergence -----

   eps=0.0001;
   n=0;
   while(abs((N(1)-N0(1))/N0(1))>eps)
     n=n+1;

     K=zeros(12,12);
     f=f0*alpha;
     Ke1=beam2g(Ex(1,:),Ey(1,:),ep1,N(1));
     Ke2=beam2g(Ex(2,:),Ey(2,:),ep1,N(2));
     Ke3=beam2g(Ex(3,:),Ey(3,:),ep3,N(3));

     K=assem(Edof(1,:),K,Ke1);
     K=assem(Edof(2,:),K,Ke2);
     K=assem(Edof(3,:),K,Ke3);

     bc=[1 0;2 0;3 0;10 0;11 0];
     [a,r]=solveq(K,f,bc);

     Ed=extract(Edof,a);
```

```
      es1=beam2gs(Ex(1,:),Ey(1,:),ep1,Ed(1,:),N(1));
      es2=beam2gs(Ex(2,:),Ey(2,:),ep1,Ed(2,:),N(2));
      es3=beam2gs(Ex(3,:),Ey(3,:),ep3,Ed(3,:),N(3));

      N0=N;
      N=[es1(1,1),es2(1,1),es3(1,1)];

      if(n>20)
        disp(['Alpha= ',num2str(alpha), ...
        ': The solution doesn''t converge.'])
        break
      end
    end

% ----- Check determinant for buckling -----

    Kred=red(K,bc(:,1));
    if (det(Kred)<=0)
      disp(['Alpha= ',num2str(alpha), ...
      ': Determinant <= 0, buckling load passed.'])
      break
    end
    if(n>20)
      break
    end
    disp(['Alpha= ',num2str(alpha),' is OK! ', int2str(n), ...
    ' iterations are performed.'])
    disp(['  '])

% ----- Save values for plotting of results -----

    deform(j)=a(4);
    M(j)=r(3);
    loadfact(j)=alpha;
    bmode=a;
  end
```

The following text strings are produced by the .m-file.

```
Alpha= 1 is OK! 3 iterations are performed.
Alpha= 1.2 is OK! 3 iterations are performed.
Alpha= 1.4 is OK! 3 iterations are performed.
Alpha= 1.6 is OK! 3 iterations are performed.
 .
 .
  .
Alpha= 6 is OK! 3 iterations are performed.
Alpha= 6.2 is OK! 4 iterations are performed.
Alpha= 6.4 is OK! 4 iterations are performed.
Alpha= 6.6 is OK! 5 iterations are performed.
Alpha= 6.8: The solution doesn't converge.
Alpha= 6.8: Determinant <= 0, buckling load passed
```

The requested plots of the horizontal displacement, the moment M_A, and the shape of the buckling mode are generated by the following commands

```
% ----- Plot results -----

figure(1), clf
plot(deform(:),loadfact(:),'+',deform(:),loadfact(:),'--')
axis([0 0.4 0 7]), grid
xlabel('Horizontal displacement (m)'), ylabel('alpha')
title('Displacement(alpha) for the upper left corner')

figure(2), clf
plot(M(:),loadfact(:),'+',M(:),loadfact(:),'--')
axis([0 0.4e6 0 7]), grid
xlabel('Moment in A (Nm)'), ylabel('alpha')
title('Supporting moment M-A(alpha)')

figure(3), clf, axis off
eldraw2(Ex,Ey,[2,3,0]);
Ed1=extract(Edof,bmode);
sfac=eldisp2(Ex,Ey,Ed1);
eldisp2(Ex,Ey,Ed1,[1,1,1],sfac);
title('Shape of buckling mode')
```

Displacement(alpha) for the upper left corner

Supporting moment M−A(alpha)

Shape of buckling mode

Index

Index