User Manual

Mefisto 2.8

Volume 1: Solid Mechanics

Contents

1 Introduction....2
2 Model Description....3
2.1 Description of the Input Data....3
2.1.1 Nodes....4
2.1.2 Elements....5
2.1.3 Sets....6
2.1.4 Constraints....6
2.1.5 Loads....9
2.1.6 Damping....11
2.2 Description of the Translation Data....12
2.2.1 Translation Data for the Gmsh MSH File Format....12
3 Functions....15
3.1 Model Definition....15
3.2 Analysis....21
3.3 Output....30
3.4 Utilities....51
4 Elements....55
4.1 Elements for 2-Dimensional Problems....55
4.1.1 Elements....55
4.1.2 Element Results....60
4.2 Elements for 3-Dimensional Problems....61
4.2.1 Elements....61
4.2.2 Element Results....70
5 List of Functions....74

1 Introduction

This volume of the manual describes how to use Mefisto to solve problems in solid mechanics. If you are new to Mefisto, please read first the Getting Started Manual.

In solid mechanics, you can use Mefisto to solve problems of linear elasticity. It supports the Finite Element Method with the following functionality:

Elements:	2-dim.:	rod, beam, membrane, point mass, rigid body mass
	3-dim.:	rod, beam, membrane, shell, solid, point mass, rigid body mass
Analysis:	Linear static analysis	
	Normal modes analysis	
	Frequency response analysis	
	Transient response analysis	

2 Model Description

The model is described by a GNU Octave structure that is input to function **mfs_new**. In case of simple models, this structure is usually defined by hand in the GNU Octave file running the analysis. For more complex models, it can be created from the output file of a preprocessor, using function **mfs_import**. Function **mfs_import** needs as additional input a GNU Octave structure with data controlling the translation.

In the following, ***variables*** written in italics can have any name or value. The names of **variables** that are not written in italics must not be changed.

2.1 Description of the Input Data

The model is defined by the following GNU Octave structure which is either defined by the user or output from function **mfs_import**:

```
model.type
     .subtype
     .nodes(:).id
              .coor(:)
     .elements(:).id
                 .type
                 .nodes(:)
                 .geom
                 .mat
     .nset
     .eset
     .constraints.prescribed(:).id
                                .dofs(:)
                 .connect(:).dofs(:)
                            .noda(:)
                            .nodd(:)
                  rigbdy(:).dofs(:)
                            .noda
                            .nodd(:)
                  rigfit(:).nodd
                            .dofa{:, 2}
     .loads.point(:).id
                    .data(:)
                    .lc
           .disp(:).id
                   .data(:)
                   .lc
           .velo(:).id
                   .data(:)
```

```
                    .lc
           .acce(:).id
                    .data(:)
                    .lc
           .inertia(:).data(:)
                       .lc
    .damping.type
             .data
```

type	**String**	Model type: For solid mechanics, the model type is **"solid"**.
subtype	**String**	The subtype further specifies the model type: **"2d"** 2-dimensional problem **"3d"** 3-dimensional problem
nodes(:)	**Structure**	Structure array with the nodal point data (see page 4)
elements(:)	**Structure**	Structure array with the element data (see page 5)
nset	**Structure**	Structure with the definition of the nodal point sets (see page 6)
eset	**Structure**	Structure with the definition of the element sets (see page 6)
constraints(:)	**Structure**	Structure array defining the constraints (see page 6)
loads	**Structure**	Structure with the load data (see page 9)
damping	**Structure**	Structure with the damping data (see page 11)

2.1.1 Nodes

```
nodes.id
nodes.coor(:)
```

id	**Integer**	Nodal point identifier

coor(:)	**Real Array**	Nodal point coordinates: 2-dimensional: $[x, y]$ 3-dimensional: $[x, y, z]$

Remarks

1. Nodal point identifiers are arbitrary positive integer numbers. They have to be unique.

2.1.2 Elements

```
elements.id
        .type
        .nodes(:)
        .geom
        .mat
```

id	**Integer**	Element identifier
type	**String**	Element type (see Sections 4.1.1 and 4.2.1)
nodes(:)	**Integer**	List of element nodal points
geom	**Structure**	Structure with geometrical data
mat	**Structure**	Structure with material data (see page 5)

Remarks

1. Element identifiers are arbitrary positive integer numbers. They have to be unique.
2. The list of element nodal points contains the identifiers of the nodal points the element is attached to.
3. The geometrical data depend on the element type (see Sections 4.1.1 and 4.2.1).

Material data

mat.type Material type (String)

The remaining fields depend on the material type:

"iso"	Isotropic material:		
	mat.E	**Real**	Young's modulus
	mat.ny	**Real**	Poisson's ratio
	mat.rho	**Real**	Mass density

2.1.3 Sets

Structure **nset** defines the nodal point sets and structure **eset** the element sets:

- The names of the fields correspond to the names of the sets.
- Each field contains an array with the identifiers of the items in the set.

Examples

nset.right_wing = 1 : 100;

Set **right_wing** contains the nodal points with identifiers 1 to 100.

eset.skin = 10 : 20;

Set **skin** contains the elements with identifiers 10 to 20.

2.1.4 Constraints

The structure contains one structure array for each type of constraint.

```
constraints.prescribed(:).id
                          .dofs(:)
            .connect(:).dofs(:)
                       .noda(:)
                       .nodd(:)
            .rigbdy(:).dofs(:)
                      .noda
                      .nodd(:)
            .rigfit(:).nodd
                      .dofa{:, 2}
```

prescribed(:)	Prescribed displacements
connect(:)	Linear constraint connecting degrees of freedom
rigbdy(:)	Linear constraint imposing a linearized rigid body motion on the dependent degrees of free-

		dom
rigfit(:)		Linear constraint determining a rigid body motion that best fits the motion of the autonomous degrees of freedom

Prescribed displacements

id	**Integer**	Identifier of the nodal point with prescribed displacements
dofs(:)	**Integer**	Identifiers of the degrees of freedom with prescribed displacements

Connected degrees of freedom

dofs(:)	**Integer**	Identifiers of the degrees of freedom which are connected
noda(:)	**Integer**	List of nodal points with autonomous degrees of freedom
nodd(:)	**Integer**	List of nodal points with dependent degrees of freedom

The displacements at the specified degrees of freedom of the dependent nodal points are identical to those of the corresponding autonomous nodal points. The number of nodal points with dependent degrees of freedom must match the number of nodal points with autonomous degrees of freedom.

Degrees of freedom following a rigid body motion

dofs(:)	**Integer**	Identifiers of the degrees of freedom which are connected to the rigid body
noda	**Integer**	Identifier of nodal point with autonomous degrees of freedom
nodd(:)	**Integer**	List of nodal points with dependent degrees of freedom

The displacements at the specified degrees of freedom of the dependent nodal points follow a rigid body motion defined by the translations and rota-

tions of the autonomous nodal point.

Rigid motion that best fits displacements

nodd	**Integer**	Identifier of the nodal point with the dependent degrees of freedom
dofa{:, 2}	**Cell Array**	Definition of the autonomus degrees of freedom: **{:, 1}** List of nodal point identifiers **{:, 2}** List of degree of freedom identifiers

The displacements of the nodal point with dependent degrees of freedom define a rigid body motion that best fits the displacements at the autonomous degrees of freedom. The autonomous degrees of freedom must define at least statically determinate supports of the rigid body.

This constraint does not stiffen the solid. It is useful to distribute concentrated loads to different nodal points or to attach concentrated masses.

Degree of freedom identifiers

2-dim.: 1 = u_x, 2 = u_y, 3 = r_z

3-dim.: 1 = u_x, 2 = u_y, 3 = u_z, 4 = r_x, 5 = r_y, 6 = r_z

Remarks

1. u_x, u_y and u_z are translations in *x*-, *y*- and *z*-direction, r_x, r_y and r_z are rotations about the *x*-, *y*- and *z*-axis.
2. If no values are assigned to the prescribed displacements (see Section 2.1.5), then their values are zero.
3. Autonomous degrees of freedom are independent degrees of freedom involved in a linear constraint.

Examples

```
rigfit.nodd = 12;
rigfit.dofa = {1, 1 : 4; 3, 2 : 3};
```

The rigid body motion with respect to nodal point 12 is determined such that it best fits the displacements u_x, u_y, u_z and r_x at nodal point 1 and u_y and u_z at nodal point 3.

```
rigfit = struct("nodd", nsets.nodd,
                "dofa", {{nsets.noda, 1 : 3}});
```

The rigid body motion with respect to the nodal point which is defined by **nsets.nodd** is determined such that it best fits the displacements u_x, u_y and u_z at the nodal points defined by **nsets.noda**. The extra braces around the cell array are needed to prevent GNU Octave from creating a structure array.

2.1.5 Loads

The structure contains one structure array for each load type.

```
loads.point(:).id
               .data(:)
               .lc
     .disp(:).id
              .data(:)
              .lc
     .velo(:).id
              .data(:)
              .lc
     .acce(:).id
              .data(:)
              .lc
     .inertia(:).data(:)
                .lc
```

point(:)	**Structure**	Point loads
disp(:)	**Structure**	Prescribed displacements
velo(:)	**Structure**	Prescribed velocities
acce(:)	**Structure**	Prescribed accelerations
inertia(:)	**Structure**	Inertia loads
id	**Integer**	Identifier of the nodal point the load is applied to
data(:)	**Real**	Array with load data
lc	**Integer**	Load case number

Point loads

data(:)	**Real**	Load vector: 2-dim.: [*Fx, Fy, Mz*] 3-dim.: [*Fx, Fy, Fz, Mx, My, Mz*]

Prescribed displacements

data(:)	**Real**	Displacement vector: 2-dim.: [*ux, uy, rz*] 3-dim.: [*ux, uy, uz, rx, ry, rz*]

Prescribed velocities

data(:)	**Real**	Velocity vector: 2-dim.: [*vx, vy, wz*] 3-dim.: [*vx, vy, vz, wx, wy, wz*]

Prescribed accelerations

data(:)	**Real**	Acceleration vector: 2-dim.: [*ax, ay, dwz*] 3-dim.: [*ax, ay, az, dwx, dwy, dwz*]

Inertia loads

data(:)	**Real**	Acceleration vector: 2-dim.: [*ax, ay*] 3-dim.: [*ax, ay, az*]

Remarks

1. Load case numbers are positive integer numbers. There should be no gaps between the load case numbers.
2. If no load case number is defined, the default **`lc = 1`** is used.
3. Displacements, velocities or accelerations that are prescribed at degrees of freedom that are not defined as prescribed (see Section 2.1.4) are ignored.

4. Velocities and accelerations can be prescribed in a dynamic analysis only.
5. *ux, uy, uz* are displacements, *vx, vy, vz* are velocities and *ax, ay, az* are accelerations.
6. *rx, ry, rz* are rotations, *wx, wy, wz* are angular velocities and *dwx, dwy, dwz* are angular accelerations. Rotations have to be defined in radians.

2.1.6 Damping

```
damping.type
       .data
```

type	**String**	Damping type
data		Damping data

The damping data depend on the damping type:

type = "Rayleigh": Rayleigh-Damping

The damping matrix is given by

$$[D]=\alpha_K[K]+\alpha_M[M].$$

Damping data: **data(1)** α_K

data(2) α_M

type = "ratios": Modal damping ratios

There are three methods to define the damping ratios:

1. If only one value is defined, it applies to all normal modes.
2. The damping ratios of the first n normal modes are defined in an array. For all higher modes, the last value defined is used.
3. The damping ratios are defined in a cell array. The cells either contain real numbers or cell arrays containing a positive integer number followed by a real number. The integer number defines the number of normal modes the damping ratio defined by the real number applies to. The last value defined is also used for all higher normal modes.

Examples

```
damping = struct("type","Rayleigh", "data",[1e-4, 0.2]);
```

defines Raleigh damping with $\alpha_K = 10^{-4}$ s und $\alpha_M = 0{,}2$ s^{-1}

```
damping = struct("type","ratios", "data",0.04);
```

defines a damping ratio of 4 % for all normal modes

```
damping.type = "ratios";
damping.data = [0.05, 0.05, 0.04, 0.03, 0.02];
```

defines a damping ratio of 5 % for the first two normal modes, of 4 % for the third normal mode, of 3 % for the the fourth normal mode and of 2 % for the fifth and all higher normal modes

```
damping.type = "ratios";
damping.data = {{5, 0.05}, {10, 0.04}, 0.03, 0.02};
```

defines a damping ratio of 5 % for the first five normal modes, of 4 % for the next ten normal modes, of 3 % for the 16th normal mode and of 2 % for all higher normal modes

2.2 Description of the Translation Data

Translation data are needed to generate the GNU Octave structure with the model definition from the output of a preprocessor. The translation data are defined by a GNU Octave structure that depends on the preprocessor used. This structure is needed as input to function **mfs_import**.

2.2.1 Translation Data for the Gmsh MSH File Format

The translation data define the model type, the subtype as well as the geometrical data, the material data, the constraints and the loads. The definition of the geometrical data, the material data, the constraints and the loads uses the physical groups defined in Gmsh.

The physical groups can also be used to define node sets. This is especially useful in combination with function **mfs_getresp** (see page 39).

```
translation_data.type
                .subtype
                .group
                .damping
```

type	**String**	Model type (see page 4)
subtype	**String**	Model subtype (see page 4)
group	**Structure**	Structure with translation data for the physical group named ***group***
damping	**Structure**	Structure with the damping definition (see Section 2.1.6)

Remarks

1. **"type"**, **"subtype"** and **"damping"** cannot be used as names of physical groups.

Element data

```
group.type
     .name
     .geom
     .mat
```

type	**String**	**"elements"**
name	**String**	Name of the Mefisto element type
	Cell Array	Translation list for element types
geom	**Structure**	Structure with geometrical data (see Sections 4.1.1 and 4.2.1)
mat	**Structure**	Structure with material data (see page 5)

If all elements of the physical group have the same type, it is sufficient to supply the name of the corresponding Mefisto element, e. g. ***group*.name = "b2"**.

If the phsical group contains elements of different types, e. g. elements with three and four nodes, then the name of the corresponding Mefisto element has to be defined for each element type in Gmsh. This is done by using a two-dimensional cell array. The first column of the cell array contains the Gmsh element types and the second the names of the corresponding Mefisto elements.

Example: ***group*.name = {2, "t3"; 3, "q4"}**

The old method of using a cell array of strings of the form ***Gmsh_type* = *Mefisto_type*** is still supported.

The Gmsh element types can be found in Sections 4.1.1 and 4.2.1 or in the Gmsh Reference Manual.[1]

Loads

```
group.type
     .name
     .data(:)
     .lc
```

1 http://gmsh.info//doc/texinfo/gmsh.pdf

type	**String**	**"loads"**
name	**String**	Load type: The name is identical to the name of the corresponding field of the **loads** structure of the model description (see page 9)
data(:)	**Array**	Array with load data
lc	**Integer**	Load case number

The load data and the load case number are identical to the corresponding fields of the **loads** structure of the model definition (see page 9).

Point loads can be applied to physical points only.

Constraints

type	**String**	**"constraints"**
name	**String**	Constraint type: The name is identical to the name of the corresponding field of the **constraints** structure of the model description (see page 6).

The remaining fields depend on the constraint type. They are identical to the corresponding fields of the **constraints** structure of the model description (see page 6).

Constraints can be applied to physical points, physical lines and physical surfaces.

Node Sets

type	**String**	**"nodeset"**

Nodes of this physical group are added to a set whose name equals the name of the physical group.

Remarks

1. Physical groups in the input file for which no translation data are provided are ignored.

3 Functions

In GNU Octave, a list of all functions can be obtained by typing

```
help mefisto
```

or just

```
mefisto
```

Information on a specific function is obtained by typing

```
help function
```

where ***function*** has to be replaced by the name of the specific function.

3.1 Model Definition

```
[model, nset, eset] = mfs_import(fid, fname, format,
                                 tdata)
```

fid	**File Handle**	File handle of the output file
fname	**String**	Name of the input file
format	**String**	Format of the input file
tdata	**Structure**	Structure with translation data (see Section 2.2)
model	**Structure**	Structure with the model description (see Section 2.1)
nset	**Structure**	Structure containing the node sets (optional)
eset	**Structure**	Structure containing the element sets (optional)

The function generates a model description from the data in the input file and the translation data. The following file formats are supported:

"msh"	Gmsh MSH ASCII file format (Both file version 2.2 and 4.1 are supported)

Error messages and information on the model generated are written to the output file.

Structures **nset** and **eset** are copies of the corresponding fields of structure **model** (see Section 2.1.3). The field names match the names of the corresponding physical groups in the Gmsh file. Physical groups defining node sets must also be defined as such in the translation data (see Section 2.2.1, page 14). Element sets are automatically built from the physical groups that define

elements (see Section 2.2.1, page 13).

```
nodes = mfs_midnodes(nodes, midnodes)
```

nodes(:)	**Structure**	Structure array with nodal point data
midnodes(:,3)	**Integer**	Definition of the mid nodes: (1) Identifier (2) Index of first node (3) Index of second node

The function generates nodes in the middle between two existing nodes. The existing nodes are referenced by their index in the structure array **nodes**. The nodes created are added to the nodes already existing.

```
nodes = mfs_linenodes(nodes, id1, id2, idnew, bias)
```

nodes(:)	**Structure**	Structure array with nodal point data
id1	**Integer**	Identifier of the first nodal point
id2	**Integer**	Identifier of the second nodal point
idnew(:)	**Integer**	Array of identifiers for the nodes to be generated
bias	**Real**	Bias factor (optional; default: 1)

The function generates nodal points along the line connecting two existing nodal points. The bias factor is the ratio of the distance between the last two nodal points to the distance between the first two nodal points.

```
[nodes, elements] = mfs_line(nodes, id1, id2, idnew,
                             idelt, type, geom, mat, bias)
```

nodes(:)	**Structure**	Structure array with nodal point data
id1	**Integer**	Identifier of the first nodal point
id2	**Integer**	Identifier of the second nodal point
idnew(:)	**Integer**	Array of identifiers for the nodes to be generated
idelt(:)	**Integer**	Array of identifiers for the elements to be generated
type	**String**	Element type
geom	**Structure**	Structure with the geometrical data (optional)
mat	**Structure**	Structure with the material data (optional)
bias	**Real**	Bias factor (optional; default: 1)

elements(:)	**Structure**	Structure array with element data

First, the function generates nodal points along the line connecting two existing nodal points. The bias factor is the ratio of the distance between the last two nodal points to the distance between the first two nodal points. Subsequently, elements between these nodal points are generated.

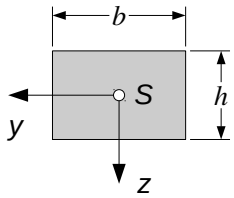
The position of the arguments must not be changed. Thus, if you want to use the default of an optional argument that is followed by an optional argument you want to define, you have to specify brackets (**"[]"**) for the undefined argument.

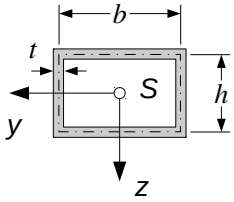
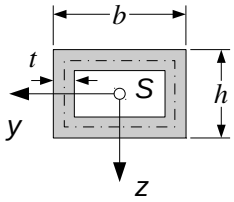
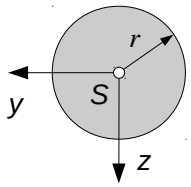
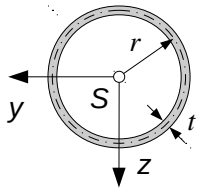
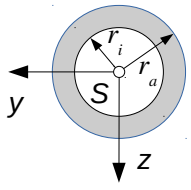
```
[geom, eyz] = mfs_beamsection(type, arg1, arg2,…)
```

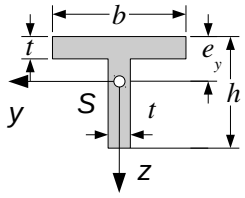
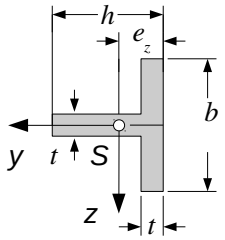
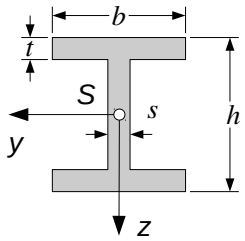
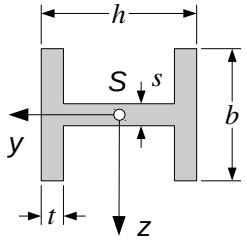
type	**String**	Cross section type
arg1, arg2, …		Arguments that depend on the cross section type
geom	**Structure**	Structure with geometrical data of the cross section
eyz(2)	**Real**	Parameters e_y and e_z defining the location of the centre of area (optional)

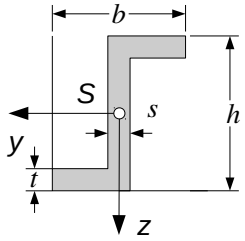
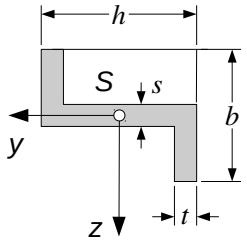
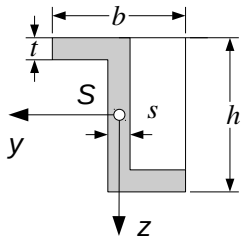
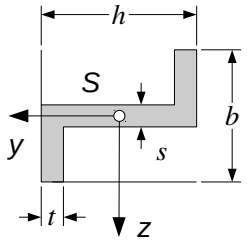
The function computes the cross section data of a 3-dimensional beam. The following fields of structure **geom** are computed: **A, Iy, Iz, Iyz, IT, CE(2)** (cf. Section 4.2.1).

Cross section types

Type	Arguments	
"bar"	**b, h**	

Type	Arguments	
`"box"`	`"thin", b, h, t`	
`"box"`	`"thick", b, h, t`	
`"circle"`	`r`	
`"ring"`	`"thin", r, t`	
`"ring"`	`"thick", ra, ri`	

Type	Arguments	
"T"	**b, h, t**	
"T"	**b, h, t, "rot"**	
"I"	**b, h, t, s**	
"I"	**b, h, t, s, "rot"**	

Type	Arguments	
"Z"	**"right", b, h, t, s**	
"Z"	**"right", b, h, t, s, "rot"**	
"Z"	**"left", b, h, t, s**	
"Z"	**"left", b, h, t, s, "rot"**	

3.2 Analysis

```
cmp = mfs_new(fid, model, opts)
```

fid	**File Handle**	File handle of the output file
model	**Structure**	Structure with the model description (see Section 2.1)
opts	**Structure**	Structure with options (optional)
cmp	**Structure**	Structure with the component data

The function generates a component from the model description. Error messages and information on the component generated are written to the output file.

List of options

Options are defined with structure **opts**. The structure has the following fields:

Name	Value	Default	Description
chkelts	**0/1**	**1**	1 = perform element checks 0 = skip element checks

Element checks should only be skipped if the model is known to be correct.

```
cmp = mfs_stiff(cmp)
```

cmp	**Structure**	Structure with the component data

The function computes the stiffness matrix.

```
cmp = mfs_mass(cmp, type)
```

cmp	**Structure**	Structure with the component data
type	**String**	Type of mass matrix (optional):
	"lumped"	lumped mass matrix (default)
	"consistent"	consistent mass matrix

The function computes the mass matrix. In case of serendipity elements, the lumped mass matrix has negative values at the corner nodes.

```
cmp = mfs_statresp(cmp)
```

cmp	**Structure**	Structure with the component data

The function computes the static displacements and the reaction loads.

The function needs the stiffness matrix. If there are inertia loads, also the mass matrix is needed.

```
mp = mfs_massproperties(fid, cmp, refpnt)
```

fid	**File Handle**	File handle of output file
cmp	**Structure**	Structure with the component data
refpnt(:)	**Real**	Coordinates of the reference point (optional)
mp	**Structure**	Structure with mass properties (optional)

The function computes the mass properties and writes them to the output file. The moments of inertia are computed with respect to the reference point defined and with respect to the centre of mass. The default reference point is the origin of the structural coordinate system.

Structure **mp** has the following fields:

Field	Value	Description
mrr(6, 6)	**Real**	Rigid body mass matrix with respect to the reference point: $$[m_{RR}]=[X_R]^T[M][X_R]$$
m	**Real**	Mass
J(3, 3)	**Real**	Inertia tensor with respect to the reference point and the structural coordinate system
cm(3)	**Real**	Coordinates of the centre of mass in the structural coordinate system

```
cmp = mfs_freevib(cmp, nofmod, opts)
```

cmp	**Structure**	Structure with the component data
nofmod	**Integer**	Number of normal modes
opts	**Structure**	Structure with options (optional)

The function computes the first **nofmod** normal modes. It can also handle systems with rigid body modes.

By default, the function tries to detect the rigid degrees of freedom automatically. The automatic detection is controlled by parameter **rbc**. A manual definition of the rigid degrees of freedom is only necessary if the automatic detec-

tion fails. If the rigid degrees of freedom are defined manually, they must define statically determinate supports.

The value of parameter **rbc** as well as the rigid body degrees of freedom can be defined with structure **opts**.

List of options

Field	Value	Default	Description
rbc	**Real > 0**	**6**	Criterion to detect rigid body modes: Increase this value if not all rigid body modes have been detected, and decrease it if too many rigid body modes have been detected
rdofs(:, 2)	**Integer**		Definition of rigid degrees of freedom: (1) Nodal point identifier (2) Degree of freedom identifier
disp	**0/1/2**	**0**	Level of diagnostic printout, see GNU Octave function **eigs**
maxit	**Integer**	**300**	Maximum number of iterations
p	**Integer**	**2*nofmod**	Number of Lanczos vectors to use, see GNU Octave function **eigs**
tol	**Real > 0**	**eps**	Convergence tolerance, see GNU Octave function **eigs**

rederr = mfs_reductionerror(fid, cmp, f, lcs)

fid	**File Handle**	File handle of the output file
cmp	**Structure**	Structure with the component data
f(:)	**Real**	List of excitation frequencies (optional)
lcs(:)	**Integer**	List of load case numbers (optional; Default: all load cases)
rederr(:)	**Structure**	Structure array with reduction errors (optional)

For each load case selected, the function computes the relative modal strain energies and their sum and writes the results to the output file.

When output arguments are specified, the list of excitation frequencies must be defined. The errors are computed for each load case in the list and each excitation frequency.

The elements of structure array **rederr** correspond to load cases. They

have the following fields:

Field	Value	Description
f(:)	**Real**	Frequencies in ascending order
abs(:)	**Real**	Absolute error in strain energy without static correction
eabs(:)	**Real**	Absolute error in strain energy with static correction
rel(:)	**Real**	Relative error in strain energy without static correction
erel(:)	**Real**	Relative error in strain energy with static correction

Examples

mfs_reductionerror(fid, frame)

computes the relative modal strain energies for all load cases and writes the results to the output file

mfs_reductionerror(fid, frame, [], [1, 3, 4])

computes the relative modal strain energies for load cases 1, 3 und 4 and writes the results to the output file

rederr = mfs_reductionerror(fid, frame, f)

computes the relative modal strain energies for all load cases and writes the results to the output file. In addition, the errors in strain energy for the excitation frequencies specified are returned in structure array **rederr** where each element of the array contains the results of one load case.

mem = mfs_meffmass(fid, cmp, refpnt)

fid	**File Handle**	File handle of output file
cmp	**Structure**	Structure with the component data
refpnt(:)	**Real**	Coordinates of the reference point (optional)
mem	**Structure**	Structure with modal effective masses (optional)

The function computes the modal effective masses and writes them to the output file. Rotations are computed with respect to the reference point. If no reference point is defined, the origin of the structural coordinate system is used as reference point.

Structure **mem** has the following fields:

Field	Value	Description
refpnt(ndim)	**Real**	Coordinates of reference point
abs(nofmod, ncol)	**Real**	Absolute modal effective masses
rel(nofmod, ncol)	**Real**	Relative modal effective masses

ndim is the problem dimension (2 or 3), **nofmod** the number of normal modes and **ncol** the number of basic rigid body modes (3 for 2-dimensional problems and 6 for 3-dimensional problems).

```
[cmp, freq, ES, WR] = ...
     mfs_freqresp(cmp, f, parameter, value, …)
```

cmp	**Structure**	Structure with the component data
f(:)	**Real**	List of excitation frequencies
freq(:)	**Real**	List of excitation frequencies (optional)
ES(:)	**Real**	Array with strain energy (optional)
WR(:)	**Real**	Array with work of the residual load (optional)

The function performs a frequency response analysis.

Parameters

Name	Value	Default	Meaning	
"method"	**String**	**"enhanced"**	Method:	
			"direct"	direct
			"modal"	modal
			"enhanced"	enhanced
"nband"	**Integer**	**5** or **0**	Number of additional excitation frequencies per halfpower bandwidth	
"loadcase"	**Integer**	**1**	Load case number	

The default of **nband** is 5 in case of a modal frequency response analysis and 0 in case of a direct frequency response analysis. Additional excitation frequencies can only be generated if normal modes have been computed.

The work of the residual load is computed only in case of a direct frequency response analysis. For a modal frequency response analysis it is always zero.

Examples

```
frame = mfs_freqresp(frame, f, "method","direct",
                     "loadcase",2);
```

performs a direct frequency response analysis using the excitation defined in load case 2

```
[frame, f] = mfs_freqresp(frame, f, "nband", 9);
```

performs a modal frequency response analysis with static correction using the excitation defined in load case 1. For each resonance frequency, 9 additional excitation frequencies are defined within the half-power bandwidth.

```
[cmp, t] = ...
     mfs_transresp(cmp, tdef, parameter, value, …)
```

cmp	**Structure**	Structure with the component data
tdef(2)	**Real**	Definition of time steps: (1) Time step Δt (2) Simulation time
t(:)	**Real**	List of time steps (optional)

The function performs a transient response analysis.

Parameters

Name	Value	Meaning
"method"	**String**	Method (optional): **"direct"** direct **"modal"** modal **"enhanced"** enhanced (default)
"load"	**Structure**	Structure array with definition of excitation (see page 27)
"rcase"	**Integer**	Result case number (optional; default: 1)
"statresp"	**Integer**	Number of a static load case defining the initial deformation
"transresp"	**Integer**	Array defining the result case and the time step of a transient response analysis defining the initial conditions: (1) Result case number (2) Time step number

Name	Value	Meaning
"param"	**Real**	Array with parameters for the Newmark algorithm (optional): (1) Newmark parameter α (Default: 0.25) (2) Newmark parameter β (Default: 0.5) (3) Parameter γ used to compute initial accelerations (Default: 0.1)

Structure array **load**

Transient excitations are defined by load patterns multiplied by time dependent functions. The structure array has the following fields:

Field	Value	Description
lc	**Integer**	Number of a load case defining the load pattern
func	**String**	Name of a function defining the time dependency
params	**Real**	Array with parameters to be supplied to function **func**

User-defined functions

The user-defined functions defining the time dependency have the following arguments:

y = func(t, params)

t(:)	**Real**	Array with time steps
params(:)	**Real**	Array with user-supplied parameters
y(3, :)	**Real**	Row 1: Function values Row 2: Values of first derivative Row 3: Values of second derivative

First and second derivatives need only be calculated if the function is associated with prescribed displacements.

Remarks

1. Both transient loads and initial conditions can be defined.
2. Either transient loads or initial conditions must be defined.
3. Transient loads can be applied loads or prescribed displacements.

4. Applied loads and prescribed displacements can be mixed.
5. Parameter γ is used in a direct transient response analysis only to compute the initial accelerations (see the Theory Manual for details).

Examples

```
load(1) = struct("lc", 1, "func", "pulse",
                 "params", [T0, 0]);
load(2) = struct("lc", 2, "func", "pulse",
                 "params", [T0, delay]);

plate = mfs_transresp(plate, [dt, T], "load", load(1),
                      "rcase", 1);
plate = mfs_transresp(plate, [dt, T], "load", load(2),
                      "rcase", 2);
plate = mfs_transresp(plate, [dt, T], "load", load,
                     "rcase", 3);
```

This code first defines two dynamic excitations. The load pattern of the first excitation is defined by load case 1 and that of the second excitation by load case 2. The time-dependency is defined by function **pulse**. Parameters **T0** and **delay** define the duration of the pulse and the delay.

The first analysis computes the response to the first excitation, the second analysis computes the response to the second excitation, and the third analysis computes the response to a combination of both excitations. The method used is the enhanced modal transient response analysis.

```
beam = mfs_transresp(beam, [dt, T], "statresp", 2,
                     "rcase", 2);
```

performs an enhanced modal transient response analysis with initial conditions defined by static load case 2. There is no additional excitation. Results are stored in result case 2.

```
[plate, t] = mfs_transresp(plate, [dt, T],"rcase", 4,
                           "transresp", [1, stepno],
                           "method", "direct");
```

performs a direct transient response analysis with initial conditions defined by time step **stepno** of result case 1 of a preceeding transient response analysis. Results are stored in result case 4.

```
[cmp, out1] = mfs_back(cmp, class, item, arg1, arg2,…)
```

cmp **Structure** Structure with the component data

class	**String**	Class
item	**String**	Item
arg1, arg2, …		Additional input arguments
out1		Additional output argument

The function performs the back transformation from modal to physical coordinates.

List of items

class = "freqresp": Frequency response analysis

Item	Description	
"disp"	Displacements	
	arg1 = ldc	Load case number
	arg2 = freq(:)	List of excitation frequencies for which displacements are requested
	out1 = freq(:)	List of actual excitation frequencies for which displacements have been computed (optional)

class = "transresp": Transient response analysis

Item	Description	
"disp"	Displacements	
	arg1 = ldc	Result case number
	arg2 = t(:)	List of time steps for which displacements are requested (optional; default is all time steps)
	out1 = t(:)	List of actual time steps for which displacements have been computed (optional)

Remarks

1. Back transformation is performed for those excitation frequencies or time steps that are closest to the excitation frequencies or time steps in the list.
2. For a transient response analysis, the velocities are also calculated.

```
cmp = mfs_results(cmp, class, item1, …)
```

cmp	**Structure**	Structure with the component data
class	**String**	Class
item1, …	**String**	Item

The function computes secondary results that can be computed from the primary results.

List of items

class = "statresp": Static response

Item	Description
"element"	Element results: see Sections 4.1.2 and 4.2.2

class = "freqresp": Frequency response

Item	Description
"element"	Element results: see Sections 4.1.2 and 4.2.2

class = "transresp": Transient response

Item	Description
"element"	Element results: see Sections 4.1.2 and 4.2.2

class = "trim": Aeroelastic trim analysis

Item	Description
"element"	Element results: see Sections 4.1.2 and 4.2.2

Remarks

1. Results of a modal frequency response analysis are computed for those excitation frequencies only for which a back transformation has been performed (see function **mfs_back**).
2. Results of a modal transient response analysis are computed for those time steps only for which a back transformation has been performed (see function **mfs_back**).

3.3 Output

```
fgh = mfs_plot(cmp, parameter, value, …)
```

cmp	**Structure**	Structure with the component data
fgh	**Figure Handle**	Handle of the figure created (optional)

The function creates a plot.

List of parameters

Name	Value	Default	Description
"nodid"	**0/1**	**0**	1 = print node identifiers
"eltid"	**0/1**	**0**	1 = print element identifiers
"deform"	**0/1/2**	**0**	Plot the deformed mesh: 0 = undeformed 1 = deformed 2 = both
"force"	**0/1/2**	**0**	Plot applied forces: 0 = don't plot forces 1 = force arrows 2 = force arrows with labels
"reac"	**0/1/2**	**0**	Plot reaction forces: 0 = don't plot forces 1 = force arrows 2 = force arrows with labels
"loadcase"	**Integer**	**1**	List of load cases to be plotted (see below)
"modes"	**Integer**	**0**	List of normal modes to be plotted (see below)
"beamaxis"	**0/1**	**0**	Plot the local *z*-axis of beam elements: 0 = don't plot the *z*-axis 1 = plot the *z*-axis
"scald"	**Real**	**0.1**	Displacement scale factor
"scalf"	**Real**	**0.1**	Force arrow scale factor
"scalax"	**Real**	**0.2**	Beam axis scale factor
"figure"	**Figure handle**	**0**	Handle of an existing figure to which the plot is to be added
"position"			Position and size of the figure on the screen (cf. the **figure** GNU

Name	Value	Default	Description
			Octave function)
"paperposition"			Position and size of the figure on the paper (cf. the **figure** GNU Octave function)
"outerposition"			Position and size of the plot (cf. GNU Octave axes properties)
"visible"	**"on"** **"off"**	**"off"**	controls visibility of the axes
"fontsize"	**Integer**	**14**	Font size
"fontname"	**String**	**"*"**	Font type
"view"	**[theta, alpha]**	**[30, 60]**	View direction: angles θ and α (see below)

The load case list and the list of normal modes are one-dimensional arrays containing positive integers.

The angles defining the view direction are defined as follows:

- θ Angle between the *z*-axis and the view direction (90° minus height)
- α Angle between the *xz*-plane and the plane containing the *z*-axis and the view direction (azimuth)

These definitions agree with the values of the angles that are indicated on the plot. They differ from the angles of the GNU Octave **view** function.

Examples

```
mfs_plot(truss, "nodid", 1, "force", 2, "fontsize", 20);
```

generates a plot of component **truss** with node identifiers and labelled force arrows using a font size of 20 pt

```
mfs_plot(truss, "nodid", 1, "force", 1, "scalf", 0.2,
         "view", [60, 140], "fontname", "Times");
```

plots the component **truss** with node identifiers and force arrows. Force arrows are scaled with a factor of 0.2. The view direction is defined by the angles $\theta = 60°$ and $\alpha = 140°$. Font type Times is used for labels and annotations.

```
mfs_print(fid, cmp, class, item1, …)
```

fid	**File**	File handle of the output file

	Handle	
cmp	**Structure**	Structure with the component data
class	**String**	Class
item1	**String**	Item

The function writes the selected result items to the output file.

List of items

class = "load": Loads

Item	Description
"point"	Point loads: forces and moments
{"point", "res"}	Resultants of point loads
"disp"	Prescribed displacements
"velo"	Prescribed velocities
"acce"	Prescribed accelerations

class = "statresp": Static response

Item	Description
"disp"	Displacements
{"disp", nsel}	Displacements at selected nodal points
"reac"	Reaction loads
{"reac", "res"}	Resultants of reaction loads
"icsl"	Internal constraint loads
{"icsl", "res"}	Resultants of internal constraint loads
"stress"	Element stresses
{"stress", esel}	Element stresses of selected elements
"pstress"	Principal element stresses
{"pstress", esel}	Principal element stresses of selected elements
"strain"	Element strains
{"strain", esel}	Element strains of selected elements
"resultant"	Element stress resultants
{"resultant", esel}	Element stress resultants of selected elements

class = "modes": Normal modes

Item	Description
"disp"	Displacements

Item	Description
{"disp", nsel}	Displacements at selected nodal points
"freq"	Natural frequencies
"reac"	Reaction loads
{"reac", "res"}	Resultants of reaction loads
"icsl"	Internal constraint loads
{"icsl", "res"}	Resultants of internal constraint loads

class = "freqresp": Frequency response

Item	Description
"disp"	Displacements
{"disp", nsel}	Displacements at selected nodal points
"velo"	Velocities
{"velo", nsel}	Velocities at selected nodal points
"acce"	Accelerations
{"acce", nsel}	Accelerations at selected nodal points
"stress"	Element stresses
{"stress", esel}	Element stresses of selected elements
"strain"	Element strains
{"strain", esel}	Element strains of selected elements
"resultant"	Element stress resultants
{"resultant", esel}	Element stress resultants of selected elements

class = "transresp": Transient response

Item	Description
"disp"	Displacements
{"disp", nsel}	Displacements at selected nodal points
"velo"	Velocities
{"velo", nsel}	Velocities at selected nodal points
"stress"	Element stresses
{"stress", esel}	Element stresses of selected elements
"strain"	Element strains
{"strain", esel}	Element strains of selected elements
"resultant"	Element stress resultants
{"resultant", esel}	Element stress resultants of selected elements

class = "diverg": Aeroelastic divergence analysis

Item	Description
"disp"	Displacements
{"disp", nsel}	Displacements at selected nodal points
"qdyn"	Dynamic pressure at divergence

class = "trim": Aeroelastic trim analysis

Item	Description
"disp"	Displacements
{"disp", nsel}	Displacements at selected nodal points
"stress"	Element stresses
{"stress", esel}	Element stresses of selected elements
"strain"	Element strains
{"strain", esel}	Element strains of selected elements
"resultant"	Element stress resultants
{"resultant", esel}	Element stress resultants of selected elements

class = "flutter": Aeroelastic flutter analysis

Item	Description
"disp"	Displacements
{"disp", nsel}	Displacements at selected nodal points

Remarks

1. Internal constraint loads are loads due to linear constraints.
2. To print results from a modal response analysis, first a back transformation has to be performed (see function **mfs_back**).
3. **nsel** is either a list of nodal point identifiers or the name of a node set.
4. **esel** is either a list of element identifiers or the name of an element set or the name of an element type.

Examples

```
mfs_print(fid, truss, "modes", "freq", "disp");
```

writes frequencies and displacements of the normal modes of component **truss** to the output file

`mfs_print(fid, cmp, "load", "point");`

writes point loads of component **`cmp`** to the output file

`mfs_print(fid, cmp, "statresp", {"disp", "edge"});`

writes the displacements at the nodal points in set **`edge`** of a static analysis of component **`cmp`** to the output file

`sig = mfs_beamstress(fid, cmp, class, eltsel, coor, lcs)`

`fid`	**`File Handle`**	File handle of the output file
`cmp`	**`Structure`**	Structure with the component data
`class`	**`String`**	Class
`eltsel`		Element selector (see page 46)
`coor(:, nc)`	**`Real`**	List of stress point coordinates in the element coordinate system: • 2-dimensional problems: **`nc`** = 1 • 3-dimensional problems: **`nc`** = 2
`lcs(:)`	**`Integer`**	List of load cases (optional; default is all load cases)
`sig(:,:,:)`	**`Real`**	Array with bending stresses (optional): • 1st index: elements • 2nd index: stress points • 3rd index: load cases

The function computes the bending stresses and writes them to the output file. The stresses are computed at the midpoints of the elements. Currently, only classes **`"statresp"`** and **`"trim"`** are supported.

`mfs_export(fname, format, cmp, class, item1, …)`

`fname`	**`String`**	Name of the output file
`format`	**`String`**	Format of the output file
`cmp`	**`Structure`**	Structure with the component data
`class`	**`String`**	Class
`item1`	**`String`**	Item

The function writes the selected items to the output file. The following file formats are supported:

- **`"msh"`**: Gmsh MSH ASCII file Version 4.1
- **`"msh41"`**: Gmsh MSH ASCII file Version 4.1

- **"msh22"**: Gmsh MSH ASCII file Version 2.2

List of items

class = "mesh": Mesh

Item	Description
	When no item is specified, the nodal points and the elements are written to the output file.
"mesh"	Nodal points and elements
"axes"	Unit vectors of the axes of the element coordinate system of 3-dimensional beam, membrane and shell elements

class = "load": Loads

Item	Description
"point"	Point loads

class = "statresp": Static results

Item	Description
"disp"	Displacements
"rot"	Rotations
"reac"	Reaction loads
"stress"	Element stresses
"pstress"	Principal element stresses
"resultant"	Element stress resultants

class = "modes": Normal modes

Item	Description
"disp"	Displacements
"rot"	Rotations

class = "freqresp": Frequency response analysis

Item	Description
"disp"	Complex displacements
"stress"	Element stresses
"resultant"	Element stress resultants

class = "transresp": Transient response analysis

Item	Description
"disp"	Displacements
"stress"	Element stresses
"resultant"	Element stress resultants

class = "diverg": Aeroelastic divergence analysis

Item	Description
"disp"	Displacements

class = "trim": Aeroelastic trim analysis

Item	Description
"disp"	Displacements
"stress"	Element stresses
"resultant"	Element stress resultants

class = "flutter": Aeroelastic flutter analysis

Item	Description
"disp"	Complex displacements

Correspondence between results and Gmsh fields

Stresses:

	Field								
Element	0	1	2	3	4	5	6	7	8
2-dim. membrane	σ_x	τ_{xy}	0	τ_{xy}	σ_y	0	0	0	0
3-dim. membrane	σ_x	τ_{xy}	τ_{xz}	τ_{xy}	σ_y	τ_{yz}	τ_{xz}	τ_{yz}	σ_z
Rod	σ_x	0	0	0	0	0	0	0	0
3-dim. solid	σ_x	τ_{xy}	τ_{xz}	τ_{xy}	σ_y	τ_{yz}	τ_{xz}	τ_{yz}	σ_z
Principal stresses	σ_1	σ_2	σ_3						

Stress resultants:

	Field								
Element	0	1	2	3	4	5	6	7	8
Rod	N	0	0	0	0	0	0	0	0
2-dim. beam	N	Q_y	0	0	0	M_z	0	0	0
3-dim. beam	N	Q_y	Q_z	M_x	M_y	M_z	0	0	0
Shell	N_x	N_y	N_{yx}	M_x	M_y	M_{xy}	Q_x	Q_y	0

Remarks

1. The values of the stress resultants of beam elements are computed at the midpoints of the elements.
2. All stresses are with respect to the global coordinate system, as expected by Gmsh.
3. Stress resultants are with respect to the element coordinate system.
4. If the results are complex, both the real and the imaginary part are output to the same Gmsh view. The real part is output as time step 0 and the imaginary part as time step 1. Thus, it is possible to use the HarmonicToTime plugin to transform the data to the time domain and to animate them.

Examples

`mfs_export("exa1.msh", "msh", truss, "mesh");`

writes the model description (nodal points and elements) of component **`truss`** to the Gmsh file `exa1.msh`

`mfs_export("exa1.pos", "msh", truss, "modes", "disp");`

writes the displacements of the normal modes of component **`truss`** to the Gmsh file `exa1.pos`

```
[out1, out2, …] =
  mfs_getresp(cmp, class, item, arg1, arg2, …)
```

`cmp`	**`Structure`**	Structure with the component data
`class`	**`String`**	Response class
`item`	**`String`**	Response item
`arg1, arg2, …`		Additional input arguments
`out1, out2, …`		Output arguments

The function returns the selected responses. The responses must have been computed before they can be requested.

The meaning of the additional input arguments and of the output arguments depends on the class and the item.

List of items

`class = "load"`: Loads

Item	Description	
"resultant"	Load resultants	
	arg1 = refpnt(:)	Coordinates of reference point (optional; default is the origin)
	arg2 = ldc(:)	List of load case numbers (optional; default is all load cases)
	out1 = F(:, :)	Resulting forces: columns correspond to load cases
	out2 = M(:, :)	Resulting moments: columns correspond to load cases

class = "statresp": Results of a static analysis

Item	Description	
"strnegy"	Strain energy	
	arg1 = ldc(:)	List of load case numbers (optional; default is all load cases)
	out1 = segy(:)	Array with strain energies
"workrsl"	Work of the residual load	
	arg1 = ldc(:)	List of load case numbers (optional; default is all load cases)
	out1 = wr(:)	Array with work of residual
"disp"	Displacements	
	arg1 = nodesel	Nodal point selector (see page 46)
	arg2 = ldc	Load case number (optional; default is 1)
	out1 = disp(:,:)	Matrix with displacements (rows correspond to nodal points)
"stress"	Stresses	
	arg1 = eltsel	Element selector (see page 46)
	arg2 = ldc	List of load case numbers (optional; default is 1)
	out1 = stress{:}	Cell array of structures with stresses (see Sections 4.1.2 and 4.2.2)
"resultant"	Stress resultants	
	arg1 = eltsel	Element selector (see page 46)
	arg2 = ldc	List of load case numbers

Item	Description	
		(optional; default is 1)
	out1 = rslt{:}	Cell array of structures with stress resultants (see Sections 4.1.2 and 4.2.2)

class = "modes": Results of a normal modes analysis

Item	Description	
"freq"	Eigenfrequencies	
	arg1 = modno(:)	Array with normal mode numbers (optional: default is all normal modes)
	out1 = freq(:)	Array with eigenfrequencies
"disp"	Displacements	
	arg1 = nodesel	Nodal point selector (see page 46)
	arg2 = modno	Mode number (optional; default is 1)
	out1 = disp(:,:)	Matrix with displacements (rows correspond to nodal points)

class = "freqresp": Results of a frequency response analysis

Item	Description	
"freq"	Excitation frequencies	
	arg1 = ldc	Load case number (optional; default is 1)
	out1 = freq(:)	Array with excitation frequencies
"strnegy"	Strain energy	
	arg1 = ldc	Load case number (optional; default is 1)
	out1 = segy(:)	Array with strain energies
"workrsl"	Work of the residual load	
	arg1 = ldc	Load case number (optional; default is 1)
	out1 = wr(:)	Array with work of residual
"Q"	Modal coordinates	
	arg1 = modno(:)	Array with normal mode numbers
	arg2 = ldc	Load case number (optional; default is 1)

Item	Description	
	out1 = Q(:, :)	Array with modal coordinates: columns correspond to excitation frequencies
	out2 = freq(:)	Array with excitation frequencies
"disp"	Displacements	
	arg1 = dofsel	Degree of freedom selector (see page 46)
	arg2 = ldc	Load case number (optional; default is 1)
	out1 = d(:, :)	Array with displacements: columns correspond to excitation frequencies
	out2 = freq(:)	Array with excitation frequencies
"velo"	Velocities	
	arg1 = dofsel	Degree of freedom selector (see page 46)
	arg2 = ldc	Load case number (optional; default is 1)
	out1 = v(:, :)	Array with velocities: columns correspond to excitation frequencies
	out2 = freq(:)	Array with excitation frequencies
"acce"	Accelerations	
	arg1 = dofsel	Degree of freedom selector (see page 46)
	arg2 = ldc	Load case number (optional; default is 1)
	out1 = a(:, :)	Array with accelerations: columns correspond to excitation frequencies
	out2 = freq(:)	Array with excitation frequencies
"reldisp"	Displacements relative to rigid body motion	
	arg1 = dofsel	Degree of freedom selector (see page 46)
	arg2 = ldc	Load case number (optional; default is 1)
	out1 = d(:, :)	Array with displacements: columns correspond to excitation

Item	Description	
		frequencies
	out2 = freq(:)	Array with excitation frequencies
"stress"	Stresses	
	arg1 = eltsel	Element selector (see page 46)
	arg2 = ldc	Load case number (optional; default is 1)
	out1 = stress{:}	Cell array of structures with stresses (see Sections 4.1.2 and 4.2.2)
	out2 = freq(:)	Array with excitation frequencies
"resultant"	Stress resultants	
	arg1 = eltsel	Element selector (see page 46)
	arg2 = ldc	Load case number (optional; default is 1)
	out1 = rslt{:}	Cell array of structures with stress resultants (see Sections 4.1.2 and 4.2.2)
	out2 = freq(:)	Array with excitation frequencies

class = "transresp": Results of a transient response analysis

Item	Description	
"time"	Time steps	
	arg1 = ldc	Result case number (optional; default is 1)
	out1 = t(:)	Array with time steps
"q"	Modal coordinates	
	arg1 = modno(:)	Array with normal mode numbers
	arg2 = ldc	Result case number (optional; default is 1)
	out1 = q(:, :)	Array with modal coordinates: columns correspond to time steps
	out2 = t(:)	Array with time steps
"qd"	Modal velocities	
	arg1 = modno(:)	Array with normal mode numbers
	arg2 = ldc	Result case number (optional; default is 1)
	out1 = qd(:, :)	Array with modal velocities:

Item	Description	
		columns correspond to time steps
	out2 = t(:)	Array with time steps
"qdd"	Modal accelerations	
	arg1 = modno(:)	Array with normal mode numbers
	arg2 = ldc	Result case number (optional; default is 1)
	out1 = qdd(:, :)	Array with modal accelerations: columns correspond to time steps
	out2 = t(:)	Array with time steps
"disp"	Displacements	
	arg1 = dofsel	Degree of freedom selector (see page 46)
	arg2 = ldc	Result case number (optional; default is 1)
	out1 = d(:, :)	Array with displacements: columns correspond to time steps
	out2 = t(:)	Array with time steps
"velo"	Velocities	
	arg1 = dofsel	Degree of freedom selector (see page 46)
	arg2 = ldc	Result case number (optional; default is 1)
	out1 = v(:, :)	Array with velocities: columns correspond to time steps
	out2 = t(:)	Array with time steps
"acce"	Accelerations	
	arg1 = dofsel	Degree of freedom selector (see page 46)
	arg2 = ldc	Result case number (optional; default is 1)
	out1 = a(:, :)	Array with accelerations: columns correspond to time steps
	out2 = t(:)	Array with time steps
"reldisp"	Displacements relative to rigid body motion	
	arg1 = dofsel	Degree of freedom selector (see page 46)
	arg2 = ldc	Result case number

Item	Description	
		(optional; default is 1)
	out1 = d(:, :)	Array with displacements: columns correspond to time steps
	out2 = t(:)	Array with time steps
"stress"	Stresses	
	arg1 = eltsel	Element selector (see page 46)
	arg2 = ldc	Load case number (optional; default is 1)
	out1 = stress{:}	Cell array of structures with stresses (see Sections 4.1.2 and 4.2.2)
	out2 = t(:)	Array with time steps
"resultant"	Stress resultants	
	arg1 = eltsel	Element selector (see page 46)
	arg2 = ldc	Load case number (optional; default is 1)
	out1 = rslt{:}	Cell array of structures with stress resultants (see Sections 4.1.2 and 4.2.2)
	out2 = t(:)	Array with time steps

class = "diverg": Results of an aeroelastic divergence analysis

Item	Description	
"qdyn"	Dynamic pressure at divergence	
	arg1 = divno(:)	Array with divergence numbers (optional; default is all)
	out1 = qdyn(:)	Array with dynamic pressures

class = "trim": Results of an aeroelastic trim analysis

Item	Description	
"disp"	Displacements	
	arg1 = nodesel	Nodal point selector (see page 46)
	arg2 = ldc	Configuration number (optional; default is 1)
	out1 = disp(:,:)	Matrix with displacements (rows correspond to nodal points)
"stress"	Stresses	

Item	Description	
	arg1 = eltsel	Element selector (see page 46)
	arg2 = ldc	List of configuration numbers (optional; default is 1)
	out1 = stress{:}	Cell array of structures with stresses (see Sections 4.1.2 and 4.2.2)
"resultant"	Stress resultants	
	arg1 = eltsel	Element selector (see page 46)
	arg2 = ldc	List of configuration numbers (optional; default is 1)
	out1 = rslt{:}	Cell array of structures with stress resultants (see Sections 4.1.2 and 4.2.2)

Selectors

Nodal point selectors:

nodid(:)	**Integer**	Array with nodal point identifiers
name	**String**	Name of a nodal point set
names{:}	**String**	Cell array of names of nodal point sets

Element selectors:

eltid(:)	**Integer**	Array with element identifiers
name	**String**	Name of an element set
names{:}	**String**	Cell array of names of element sets

Degree of freedom selectors:

rid(:, 2)	**Integer**	**rid(:, 1)**	Nodal point identifier
		rid(:, 2)	Degree of freedom identifier
rid{:, 2}	**String, Integer**	**rid{:, 1}**	Name of a nodal point set
		rid{:, 2}	Degree of freedom identifier

Remarks

1. Item **"workrsl"** is only supported in a direct frequency response analysis.

2. Relative displacements are only available with the force summation method.

Examples

```
f = mfs_getresp(frame, "freqresp", "freq");
```

returns the excitation frequencies used in a frequency response analysis of load case 1 of component **frame**

```
rid = [idB, 1; idE, 1; idG, 1];
A   = mfs_getresp(frame, "freqresp", "acce", rid, 2);
```

returns the accelerations of degree of freedom 1 (*x*-translations) of the nodal points with identifiers **idB**, **idE** and **idG** computed in a frequency response analysis of load case 2 of component **frame**:

- **A(1, :)** contains the accelerations of nodal point **idB**
- **A(2, :)** contains the accelerations of nodal point **idE**
- **A(3, :)** contains the accelerations of nodal point **idG**

```
Q = mfs_getresp(truss, "freqresp", "Q", 1 : 5);
```

returns the response of the modal coordinates of normal modes 1 to 5 computed in a frequency response analysis of load case 1 of component **truss**

```
   …
rslt   = mfs_getresp(roof, "statresp", "resultant",
                     "edge");
r      = cell2mat(rslt);
coor   = reshape([r.coor], 3, length(eltids));
phi    = 180 * atan2(-coor(2, :), coor(3, :)) / pi;
figure(1);
plot(phi, [r.Ny], "marker", "o");
grid;
xlabel('\phi'); ylabel('N_y');
```

extracts the stress resultants of the elements in set **edge**, converts the cell array to a structure array, accesses the coordinates and plots the membrane force N_y as a function of the angle. The cell array can only be converted into a structure array if all results are of the same type.

```
eltid = [108, 467];
R = mfs_getresp(beam, "freqresp", "resultant", eltid);
```

returns the frequency dependent stress resultants of elements 108 und 467. **R{1}** contains the results of element 108 and **R{2}** those of element 467.

```
res = mfs_modecont(fid, cmp, rid, freq, ldc, thresh)
res = mfs_modecont(cmp, rid, freq, ldc, thresh)
```

fid	**File Handle**	File handle of the output file
cmp	**Structure**	Structure with the component data
rid		Degree of freedom selector (see page 46)
freq(:)	**Real**	List of excitation frequencies
ldc	**Integer**	List of load case numbers (optional; default is all load cases
thresh(2)	**Real**	Thresholds (optional) **(1)** Threshold for printed output (default: 1 %) **(2)** Threshold for output in structure **polar** (default: 10 %)
res(:, :)	**Structure**	Structure array with output data (optional): rows correspond to degrees of freedom and columns to load cases

The function computes the modal contribution factors. If a file handle is given, the results are printed to the output file. If an output argument is given, the results are stored in a structure array.

Structure **res** has the following fields:

Field	Value	Description
rid(2)	**Integer**	Nodal point and degree of freedom identifier
rfreq(:)	**Real**	Resonance frequencies
freq(:)	**Real**	Excitation frequencies
resp(:)	**Complex**	Total response
mcf(:, :)	**Complex**	Modal contribution factors: rows correspond to normal modes and columns to excitation frequencies
polar(:)	**Structure**	Structure array with data to create polar plots

The elements of structure array **polar** correspond to excitation frequencies. They have the following fields:

Field	Value	Description
title	**String**	Title containing load case number, nodal point and degree of freedom identifier and excitation frequency

Field	Value	Description
theta(2, :)	**Real**	Phase angles of contribution factors
rho(2, :)	**Real**	Magnitudes of contribution factors
legend{:}	**String**	Cell array of strings containing the legend

```
GAB = mfs_randresp(Gxx, Gxy, HA, HB)
[G, f] = mfs_randresp(cmp, Gxx, Gxy, item, select, fpsd,
                      ldcs, cflag)
```

cmp	**Structure**	Structure with the component data
Gxx(n, :)	**Complex**	Array with power spectral densities (rows correspond to load cases, columns to frequencies)
Gxy(:, :)	**Complex**	Array with cross spectral densities (columns correspond to frequencies) (may be empty)
HA(n, :)	**Complex**	Array with transfer functions of response A (rows correspond to load cases, columns to frequencies)
HB(n, :)	**Complex**	Array with transfer functions of response B (rows correspond to load cases, columns to frequencies) (optional, default is **HA**)
item	**String**	Response item
select		Item selector (see page 46)
fpsd(:)	**Real**	List of frequencies corresponding to spectra (optional)
ldcs(:)	**Integer**	List of load case identifiers corresponding to the transfer functions (optional; default is 1 to **n**)
cflag	**Integer**	Flag controlling computation of cross spectral densities between element results: 0 do not compute cross spectral densities (default) 1 compute cross spectral densities
GAB(:)	**Complex**	Power or cross spectral density of result
G		Power and cross spectral densities of results
f	**Real**	List of frequencies (optional)

The function computes power and cross spectral densities of results. In the

first form, the power or cross spectral density of the result is computed from the power and cross spectral densities of the excitation and the transfer functions. The frequencies of the spectra must match the frequencies of the transfer functions.

In the second form, the transfer functions are obtained from the results of frequency response analyses. The output depends on the response item:

1. For nodal point results (displacements, velocities or accelerations), **G** is an array. The rows correspond to the response items and the columns to the frequencies.
2. For element results, **G** is a cell array of structures. Each structure contains the results of one element. Power spectral densities are contained in the fields described in Sections 4.1.2 and 4.2.2. The names of the fields with the cross spectral densities are of the form ***result1_result2***, e.g. field **N_My** of a beam element contains the cross spectral density between the normal force and the bending moment about the *y*-axis. Only the elements above the diagonal of the spectral matrix are computed.

The power and cross power spectral densities are computed at the excitation frequencies **f** of the frequency response analysis. The power and cross spectral densities **Gxx** and **Gxy** are interpolated from **fpsd** to **f**.

Remarks

1. Array **Gxy** contains the elements above the diagonal of the spectral matrix. The first **n−1** rows contain the elements of the first row of the spectral matrix, the next **n−2** rows those of the second row and so on.
2. If all cross spectral densities are zero, array **Gxy** may be empty.
3. If **HB** is not defined, the power spectral density **GAA** is computed.
4. If **fpsd** is not defined, the frequencies of the spectra must match the frequencies of the transfer functions.
5. If requested, cross spectral densities are computed between element results of the same element and the same result point.

Examples

Gyy = mfs_randresp(Gxx, Gxy, HQ2R);

computes the power spectral density using the transfer functions **HQ2R**

Gyz = mfs_randresp(Gxx, Gxy, HMy, HMz)

computes the cross spectral density between **My** and **Mz** using the

transfer functions **HMy** and **HMz**

```
[G, f] = mfs_randresp(waggon, Gxx, Gxy, "acce",
                      {"Q", 3}, fpsd);
```

computes the power spectral densities of the accelerations in z-direction at the nodes in set **Q**.

```
[G, f] = mfs_randresp(waggon, Gxx, Gxy, "resultant",
                      [611, 615], fpsd,[], 1);
```

computes the power and cross spectral densities of the stress resultants of elements 611 and 615. Cross spectral densities are computed between the stress resultants of each element.

3.4 Utilities

```
items = mfs_getset(cmp, settype, setnam)
```

cmp	**Structure**	Structure with the component data
settype	**String**	Set type:
		"nset" Node set
		"eset" Element set
setnam	**String**	Name of the set
items(:)	**Integer**	List of identifiers of the items in the set

The function returns the identifiers of the items in the set.

```
nset = mfs_eset2nset(cmp, eset)
cmp  = mfs_eset2nset(cmp, eset, setnam)
```

cmp	**Structure**	Structure with the component data
eset(:)	**Integer**	Array with element identifiers or name of an existing element set
setnam	**String**	Name for the node set
nset(:)	**Integer**	Array with identifiers of nodes referenced by elements in **eset** (in ascending order)

The function builds a set of nodes from a set of elements. The set of nodes contains all nodes that are referenced by the elements in the element set.

In the first form, the function returns the list of identifiers of the nodal points in the set. In the second form, the set is stored in the component.

```
[cmp, set] = mfs_newset(cmp, settype, fct, ops, setnam)
```

cmp	**Structure**	Structure with the component data
settype	**String**	Set type:
		"nset" Node set
		"eset" Element set
fct	**String**	Function to create new set (see page 52)
ops		Operands from which to create set (see page 52)
setnam	**String**	Name of the set
set(:)	**Integer**	List of identifiers of the items in the set (optional)

The function creates a new set from existing sets or based on geometrical information.

List of functions

Name	Description
"near"	Find the point or element closest to a reference point
	ops = coor(nc) Coordinates of the reference point
"box"	Find the points or elements inside a box
	ops = box(nc, 2) Minima and maxima of the coordinates
"union"	Build the union of existing sets
	ops = sets{:} Cell array with names of existing sets
"intersect"	Build the intersection of existing sets
	ops = sets{:} Cell array with names of existing sets

Remarks

1. In case of element sets, the coordinates of the midpoint of the element are used to perform the geometrical operations.
2. **nc** is the number of coordinates of a point (2 or 3).

```
[idx, val] = mfs_peaks(x, option)
```

x(:)	**Real**	Array with input data
option	**String**	Type of peak (optional)
		"max" Maxima (default)
		"min" Minima
		"both" Maxima and minima

idx(:)	**Integer**	Peak indices
val(:)	**Real**	Peak values (optional)

The function finds the maxima, minima or both in an array of data and returns their indices and values. If there are no peaks, the output arguments are empty.

```
[x, t] = mfs_freq2time(X, df, dt, bc)
cmp    = mfs_freq2time(cmp, spectra, rcase, dt, bc)
```

X(:, :)	**Complex**	Array with Fourier transforms (columns correspond to frequencies)
df	**Real**	Frequency increment
dt	**Real**	Time step (optional)
bc	**Integer**	Base correction (optional): 0 No correction (default) 1 Subtract a constant 2 Subtract a linear function
cmp	**Structure**	Structure with the component data
spectra	**Structure**	Structure assigning spectra to load cases (see p. 53)
rcase	**Integer**	Result case number for results (optional; default: 1)
x(:, :)	**Real**	Array with time series (columns correspond to time steps)
t(:)	**Real**	Array with points in time (optional)

In the first form, the function computes the time series from the coefficients of the Fourier transforms corresponding to positive frequencies.

In the second form, the function uses the results from a frequency response analysis, multiplies them by the spectra and performs the inverse Fourier transform. The results are stored in the component as results of a transient response analysis.

Fields of structure **spectra**

Field	Value	Description
df	**Real**	Frequency step
lc(:)	**Integer**	Array with identifiers of load cases with transfer functions

Field	Value	Description
spec(:,:)	**Complex**	Spectra assigned to load cases: rows correspond to load cases, columns to frequencies

Remarks

1. If no time step is defined, it is determined from the frequency increment and the number of frequencies.
2. The base correction can be used to make the time series start at zero and to remove a linear shift. This may help to obtain physically correct results in cases where the Fourier transform becomes infinite or includes a Dirac distribution.
3. In the first form, the Fourier transforms must include the value at the zero frequency. They must not include the values at the negative frequencies.
4. In the second form, the spectra must include the value at the zero frequency. The values at the negative frequencies may or may not be included.

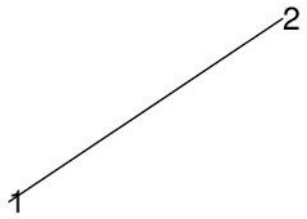
4 Elements

4.1 Elements for 2-Dimensional Problems

These elements are available with subtype **"2d"**.

4.1.1 Elements

r2

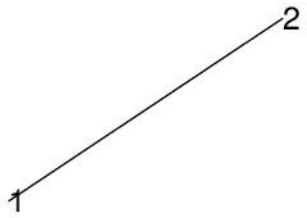


Rod element with 2 nodes
Material data: needed
Results: Rod results (see p. 60) at midpoint
Gmsh geometrical type: 1
Geometrical data:

Field	Content
A	Cross section area

Degrees of freedom at nodal points: u_x, u_y

b2

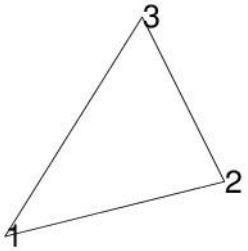


Euler-Bernoulli beam with 2 nodes
Material data: needed
Results: Beam results (see p. 60) at midpoint
Gmsh geometrical type: 1
Geometrical data:

Field	Content
A	Cross section area
I	Area moment of inertia

Degrees of freedom at nodal points: u_x, u_y, ϕ_z

t3



Membrane element with 3 nodes
Material data: needed
Results: Plane stress results (see p. 61) at mid-

point

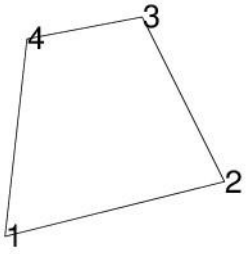
Gmsh geometrical type: 2

Geometrical data:

Field	Content
t	Thickness

Degrees of freedom at nodal points: u_x, u_y

q4



Membrane element with 4 nodes

Material data: needed

Results: Plane stress results (see p. 61) at mid-point

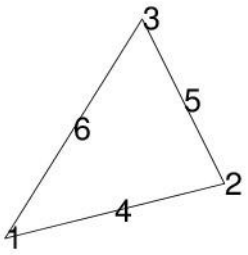
Gmsh geometrical type: 3

Geometrical data:

Field	Content
t	Thickness

Degrees of freedom at nodal points: u_x, u_y

t6



Membrane element with 6 nodes

Material data: needed

Results: Plane stress results (see p. 61) at mid-point

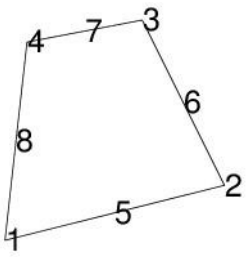
Gmsh geometrical type: 9

Geometrical data:

Field	Content
t	Thickness

Degrees of freedom at nodal points: u_x, u_y

q8



Membrane element with 8 nodes

Material data: needed

Results: Plane stress results (see p. 61) at stress

points

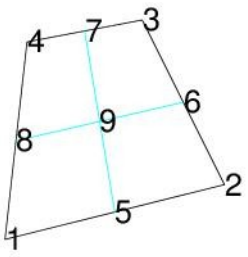
Gmsh geometrical type: 16

Geometrical data:

Field	Content
t	Thickness

Degrees of freedom at nodal points: u_x, u_y

q9



Membrane element with 9 nodes

Material data: needed

Results: Plane stress results (see p. 61) at stress points

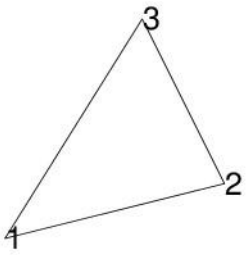
Gmsh geometrical type: 10

Geometrical data:

Field	Content
t	Thickness

Degrees of freedom at nodal points: u_x, u_y

t3r



Membrane element with 3 nodes and drilling degrees of freedom

Material data: needed

Results: Plane stress results (see p. 61) at midpoint

Gmsh geometrical type: 2

Geometrical data:

Field	Content
t	Thickness

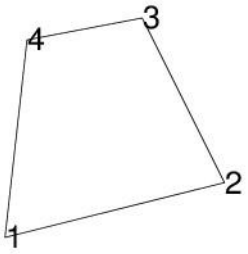
Degrees of freedom at nodal points: u_x, u_y, ϕ_z

Remarks

1. The element has one spurious mode where all three rotations have the same value. To prevent this spurious mode, it is sufficient to constrain

the rotation at one single node of the mesh. The spurious mode is also prevented if at least one element is connected to a beam element.

q4r



Membrane element with 4 nodes and drilling degrees of freedom

Material data: needed

Results: Plane stress results (see p. 61) at midpoint

Gmsh geometrical type: 3

Geometrical data:

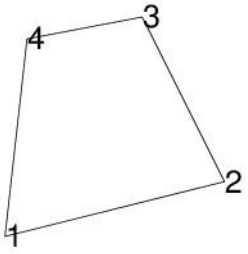
Field	Content
t	Thickness

Degrees of freedom at nodal points: u_x, u_y, ϕ_z

Remarks

1. The element has one spurious mode where all three rotations have the same value. To prevent this spurious mode, it is sufficient to constrain the rotation at one single node of the mesh. The spurious mode is also prevented if at least one element is connected to a beam element.

q4e



Membrane element with 4 nodes

Material data: needed

Results: Plane stress results (see p. 61) at midpoint

Gmsh geometrical type: 3

Geometrical data:

Field	Content
t	Thickness

Degrees of freedom at nodal points: u_x, u_y

Remarks

1. The element has an additional shape function related to an internal

node located at the midpoint of the element. The degrees of freedom of this internal node are eliminated by static condensation.

2. The element is more robust with respect to distortion but computation time is increased.
3. This element is experimental.

m1

Point mass



Material data: not needed

Result type: none

Gmsh geometrical type: 15

Geometrical data:

Field	Content
m	Mass

Degrees of freedom at nodal points: none

m2

Rigid body mass



Material data: not needed

Result type: none

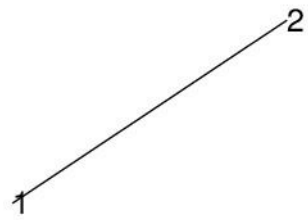
Gmsh geometrical type: 15

Geometrical data:

Field	Content
m	Mass
J	Mass moment of inertia

Degrees of freedom at nodal points: none

g2



Graphic element with 2 nodes

Material data: not needed

Results: none

Gmsh geometrical type: 1

Geometrical data: not needed

Degrees of freedom at nodal points: none

Remark

This element contributes neither to the stiffness matrix nor to the mass matrix. It can be used, for example, to visualise rigid bodies.

4.1.2 Element Results

Element results are returned by function **`mfs_getresp`** (see page 39) in a cell array of structures. The first five fields of the structures are identical for all result types:

`id`	Element identifier
`nofpnt`	Number of result points
`nofcol`	Number of results per item:
	`statresp` Number of loadcases requested
	`freqresp` Number of excitation frequencies
	`transresp` Number of time steps
`coor(nofpnt,2)`	Coordinates of result points
`rtype`	Result type: 1 = Beam results
	2 = Rod results
	3 = Plane stress results

The remaining fields depend on the result type.

Rod Results

Both stresses and stress resultants are available.

Stresses:

`sig(nofcol)`	Normal stress (positive if tension, negative if compression)

Stress Resultants:

`N(nofcol)`	Normal force (positive if tension, negative if compression)

Beam Results

Only stress resultants are available. The positive directions of the stress resultants can be seen in Figure 4.1.

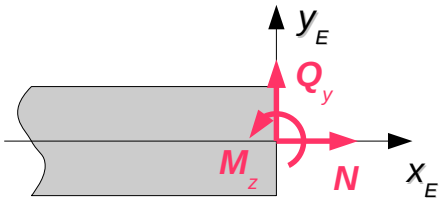


Figure 4.1: Positive Directions of Beam Stress Resultants

N(nofcol)	Normal force
Qy(nofcol)	Shear force in y_E-direction
Mz(nofcol)	Bending moment about z_E-axis

Plane Stress Results

The stresses are given with respect to the global coordinate system.

sigx(nofpnt,nofcol)	Stress component σ_x
sigy(nofpnt,nofcol)	Stress component σ_y
tauxy(nofpnt,nofcol)	Stress component τ_{xy}

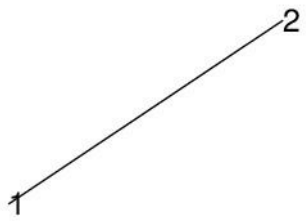
The locations of the stress points of the quadratic quadrilateral elements can be seen in Figure 4.2.

4.2 Elements for 3-Dimensional Problems

These elements are available with subtype **"3d"**.

4.2.1 Elements

r2



Rod element with 2 nodes

Material data: needed

Results: Rod results (see p. 71) at midpoint

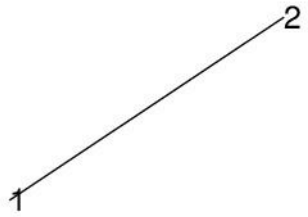
Gmsh geometrical type: 1

Geometrical data:

Field	Content
A	Cross section area

Degrees of freedom at nodal points: u_x, u_y, u_z

b2



Euler-Bernoulli beam with 2 nodes

Material data: needed

Results: Beam results (see p. 71) at midpoint

Gmsh geometrical type: 1

Geometrical data:

Field	Content
A	Cross section area
Iy	Area moment of inerta $I_y = \int_A z^2 \, dA$
Iz	Area moment of inertia $I_z = \int_A y^2 \, dA$
Iyz	Moment of deviation $I_{yz} = -\int_A y\, z \, dA$
IT	Torsional constant I_T
v(3)	Vector in the $x_E z_E$-plain of the element (see Figure 4.3)
Q(3)	Coordinates of a point in the $x_E z_E$-plain of the element in the global coordinate system (see Figure 4.3)
CE(2)	Coordinates of the elastic centre in the element coordinate system
P(2)	Coordinates of the nodal points in the element coordinate system

Degrees of freedom at nodal points: u_x, u_y, u_z, ϕ_x, ϕ_y, ϕ_z

Remarks

1. The x_E-axis of the element connects the centres of area of the cross

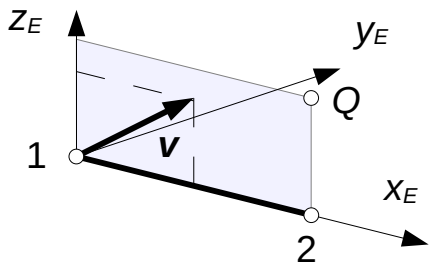
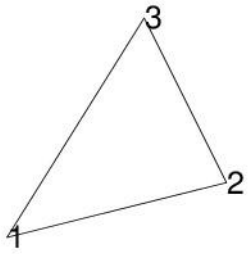


Figure 4.3: Beam Element Coordinate System

sections.

2. The element coordinate system of a beam element can be defined by specifying either a vector ***v*** or a point *Q* in the $x_E z_E$-plane.
3. If the elastic centre is not defined, it coincides with the centre of area.
4. If the coordinates of the nodal points are not defined, they coincide with the centre of area.

t3



Membrane element with 3 nodes

Material data: needed

Results: Plane stress results (see p. 72) at mid-point

Gmsh geometrical type: 2

Geometrical data:

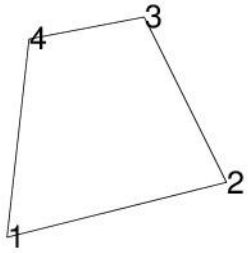
Field	Content
t	Thickness
wtol	not used

Degrees of freedom at nodal points: u_x, u_y, u_z

Remarks

1. Membrane elements have no bending stiffness.

q4



Membrane element with 4 nodes

Material data: needed

Results: Plane stress results (see p. 72) at mid-point

Gmsh geometrical type: 3

Geometrical data:

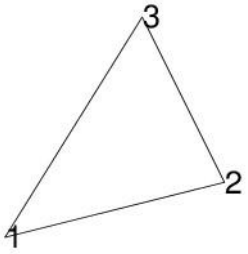
Field	Content
t	Thickness
wtol	Warping tolerance (optional; default: 10^{-3})

Degrees of freedom at nodal points: u_x, u_y, u_z

Remarks

1. Membrane elements have no bending stiffness.
2. If the warping of the element is larger than **wtol**, the warping correction is applied (see the Theory Manual for details).

t3r



Membrane element with 3 nodes and drilling degrees of freedom

Material data: needed

Results: Plane stress results (see p. 72) at midpoint

Gmsh geometrical type: 2

Geometrical data:

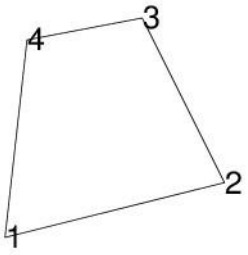
Field	Content
t	Thickness
wtol	not used

Degrees of freedom at nodal points: u_x, u_y, u_z, ϕ_x, ϕ_y, ϕ_z

Remarks

1. Membrane elements have no bending stiffness.
2. This element has six degrees of freedom at each nodal point, three translations and three rotations, but it has a stiffness only for the in-plane translations and the rotation about the element normal.
3. The element has one spurious mode where the rotations about the element normal at all nodes of the element have the same value. To prevent this spurious mode it is sufficient to constrain the rotation at one single node. The spurious mode is also prevented if at least one element is connected to a beam element, or if not all elements are in the same plane.

q4r



Membrane element with 4 nodes

Material data: needed

Results: Plane stress results (see p. 72) at midpoint

Gmsh geometrical type: 3

Geometrical data:

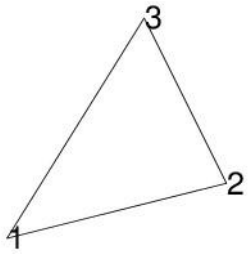
Field	Content
t	Thickness
wtol	Warping tolerance (optional; default: 10^{-3})

Degrees of freedom at nodal points: u_x, u_y, u_z, ϕ_x, ϕ_y, ϕ_z

Remarks

1. Membrane elements have no bending stiffness.
2. If the warping of the element is larger than **wtol**, the warping correction is applied (see the Theory Manual for details).
3. This element has six degrees of freedom at each nodal point, three translations and three rotations, but it has a stiffness only for the in-plane translations and the rotation about the element normal.
4. The element has one spurious mode where the rotations about the element normal at all nodes of the element have the same value. To prevent this spurious mode it is sufficient to constrain the rotation at one single node. The spurious mode is also prevented if at least one element is connected to a beam element, or if not all elements are in the same plane.

s3, s3r



Shell element with 3 nodes

Material data: needed

Results: Shell results (see p. 72) at midpoint

Gmsh geometrical type: 2

Geometrical data:

Field	Content
t	Thickness
zo	Offset (optional)
wtol	not used
ka	Factor for artificial stiffness to stabilise the drilling degrees of freedom (optional; default: 10^{-2}) (not used for **s3r** element)
l2trat	Reference value for the ratio of the element length to the

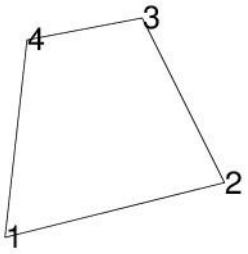
Field	Content
	thickness (optional; default: 10)

Degrees of freedom at nodal points: u_x, u_y, u_z, ϕ_x, ϕ_y, ϕ_z

Remarks

1. The offset defines the distance of the nodal points from the mid surface of the element.
2. Triangular elements with 3 nodes are never warped. **`wtol`** is accepted to make the input compatible with the **`s4`** element.
3. The **`s3`** element has no physical stiffness for the drilling degrees of freedom. They are stabilised by applying an artificial stiffness (see the Theory Manual for details).
4. The element has one spurious mode where the rotations about the element normal at all nodes of the element have the same value. To prevent this spurious mode it is sufficient to constrain the rotation at one single node. The spurious mode is also prevented if at least one element is connected to a beam element, or if not all elements are in the same plane.
5. The value of **`l2trat`** controls the scaling of the shear modulus used for the transverse shear to ensure the element converges to the Kirchhoff theory even when the element size is not large compared to the element thickness (see the Theory Manual for details).
6. The performance of the triangular shell element is not as good as that of the quadrilateral shell element. With very coarse meshes shear locking may occur.

`s4, s4r`



Shell element with 4 nodes

Material data: needed

Results: Shell results (see p. 72) at midpoint

Gmsh geometrical type: 3

Geometrical data:

Field	Content
`t`	Thickness
`zo`	Offset (optional)
`wtol`	Warping tolerance (optional; default: 10^{-3})

Field	Content
ka	Factor for artificial stiffness to stabilise the drilling degrees of freedom (optional; default: 10^{-2}) (not used for **s4r** element)
l2trat	Reference value for the ratio of the element length to the thickness (optional; default: 10)

Degrees of freedom at nodal points: u_x, u_y, u_z, ϕ_x, ϕ_y, ϕ_z

Remarks

1. The offset defines the distance of the nodal points from the mid surface of the element.
2. If the warping of the element is larger than **wtol**, the warping correction is applied (see the Theory Manual for details).
3. The **s4** element has no physical stiffness for the drilling degrees of freedom. They are stabilised by applying an artificial stiffness (see the Theory Manual for details).
4. The element has one spurious mode where the rotations about the element normal at all nodes of the element have the same value. To prevent this spurious mode it is sufficient to constrain the rotation at one single node. The spurious mode is also prevented if at least one element is connected to a beam element, or if not all elements are in the same plane.
5. The value of **l2trat** controls the scaling of the shear modulus used for the transverse shear to ensure the element converges to the Kirchhoff theory even when the element size is not large compared to the element thickness (see the Theory Manual for details).

p6

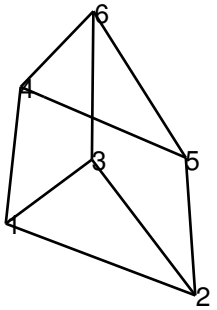
Solid element with 6 nodes

Material data: needed

Results: General stress results (see p. 73) at midpoint

Gmsh geometrical type: 6

Geometrical data: none



h8

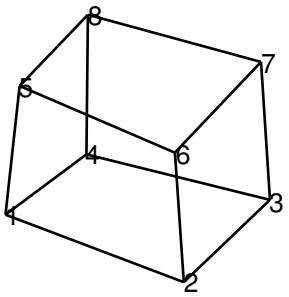
Solid element with 8 nodes

Material data: needed

Results: General stress results (see p. 73) at mid-point

Gmsh geometrical type: 5

Geometrical data: none



p15

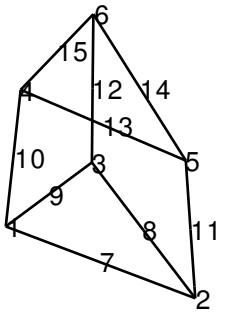
Solid element with 15 nodes

Material data: needed

Results: General stress results (see p. 73) at stress points

Gmsh geometrical type: 18

Geometrical data: none



h20

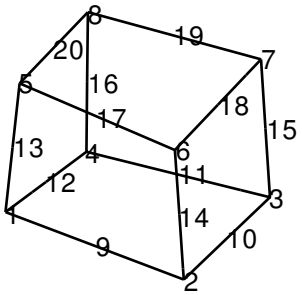
Solid element with 20 nodes

Material data: needed

Results: General stress results (see p. 73) at stress points

Gmsh geometrical type: 17

Geometrical data: none



p18

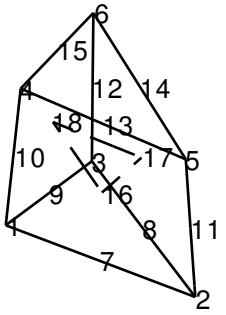
Solid element with 18 nodes

Material data: needed

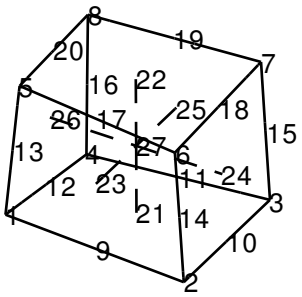
Results: General stress results (see p. 73) at stress points

Gmsh geometrical type: 13

Geometrical data: none



h27



Solid element with 27 nodes

Material data: needed

Results: General stress results (see p. 73) at stress points

Gmsh geometrical type: 12

Geometrical data: none

m1

Point mass

Material data: not needed

Result type: none

Gmsh geometrical type: 15

Geometrical data:

Field	Content
m	Mass

Degrees of freedom at nodal points: none

m2

Rigid body mass

Material data: not needed

Result type: none

Gmsh geometrical type: 15

Geometrical data:

Field	Content
m	Mass
Jx	Mass moment of inertia $J_x = \int_K (y^2 + z^2)\, dm$
Jy	Mass moment of inertia $J_y = \int_K (y^2 + z^2)\, dm$
Jz	Mass moment of inertia $J_z = \int_K (x^2 + y^2)\, dm$

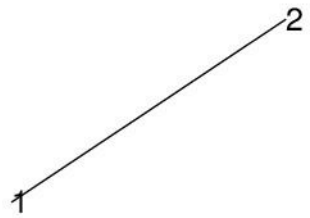
Jxy Mass moment of deviation $J_{xy} = -\int_K x\,y\,dm$

Jxz Mass moment of deviation $J_{xz} = -\int_K x\,z\,dm$

Jyz Mass moment of deviation $J_{yz} = -\int_K y\,z\,dm$

Degrees of freedom at nodal points: none

g2



Graphic element with 2 nodes

Material data: not needed

Results: none

Gmsh geometrical type: 1

Geometrical data: not needed

Degrees of freedom at nodal points: none

Remark

This element contributes neither to the stiffness matrix nor to the mass matrix. It can be used, for example, to visualize rigid bodies.

4.2.2 Element Results

Element results are returned by function **mfs_getresp** (see page 39) in a cell array of structures. The first five fields of the structures are identical for all result types:

id	Element identifier
nofpnt	Number of result points
nofcol	Number of results per item:
	statresp Number of load cases requested
	freqresp Number of excitation frequencies
	transresp Number of time steps
	trim Number of configurations
coor(nofpnt,3)	Coordinates of result points
rtype	Result type: 1 = Beam results
	2 = Rod results
	3 = Plane stress results

4 = Shell results
5 = General stress results

The remaining fields depend on the result type.

Rod Results

Both stress resultants and stresses are available.

Stresses:

sig(nofcol) Normal stress (positive if tension, negative if compression)

Stress Resultants:

N(nofcol) Normal force (positive if tension, negative if compression)

Beam Results

Only stress resultants are available. The positive directions of the stress resultants can be seen in Figure 4.4.

N(nofcol) Normal force
Qy(nofcol) Shear force in *y*-direction
Qz(nofcol) Shear force in *z*-direction
Mx(nofcol) Torque
My(nofcol) Bending moment about *y*-axis
Mz(nofcol) Bending moment about *z*-axis

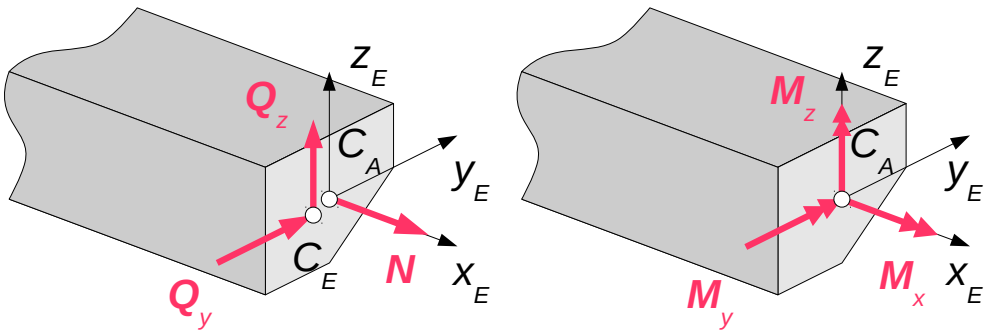


Figure 4.4: Positive Directions of Beam Stress Resultants

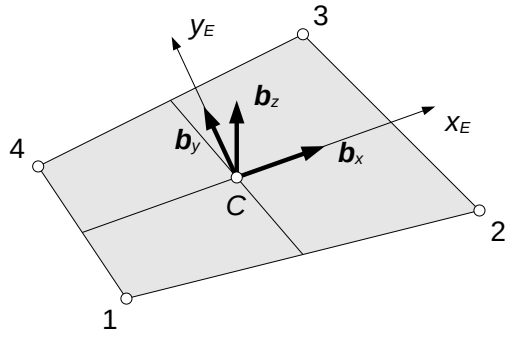


Figure 4.5: Coordinate System of Quadrilateral Element

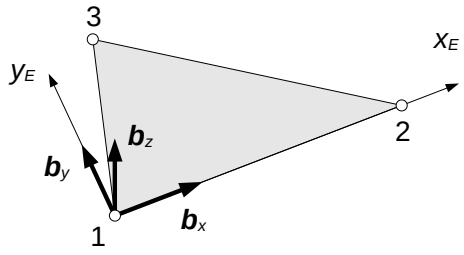


Figure 4.6: Coordinate System of Triangular Element

Plane Stress Results

The stresses are given with respect to the element coordinate system (see Figures 4.5 and 4.6).

TE(3, 3)	Transformation matrix to element coordinate system
sigx(nofpnt,nofcol)	Stress component σ_x
sigy(nofpnt,nofcol)	Stress component σ_y
tauxy(nofpnt,nofcol)	Stress component τ_{xy}

Shell Results

Both stresses and stress resultants are available. Stresses are computed at the upper and the lower surface of the shell. All results are given with respect to the element coordinate system (see Figures 4.5 and 4.6).

Stresses:

TE(3, 3)	Transformation matrix to element coordinate system
sigxu(nofcol)	σ_x on upper surface
sigxl(nofcol)	σ_x on lower surface
sigyu(nofcol)	σ_y on upper surface
sigyl(nofcol)	σ_y on lower surface
tauxyu(nofcol)	τ_{xy} on upper surface
tauxyl(nofcol)	τ_{xy} on lower surface

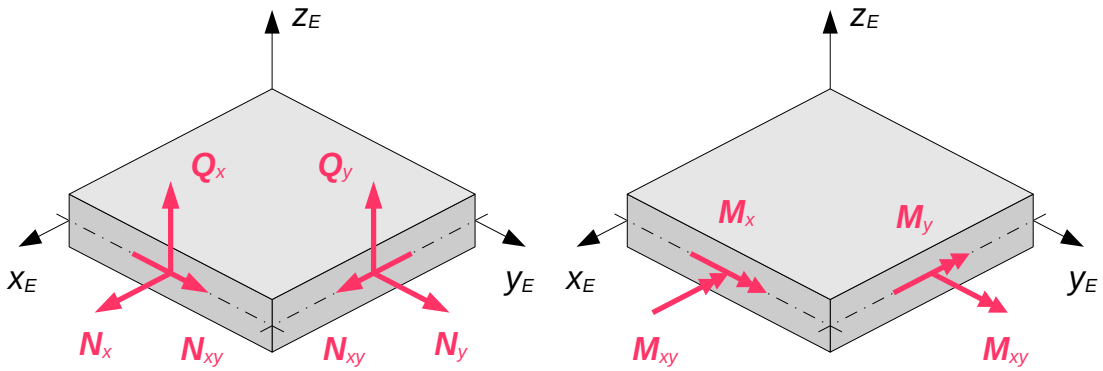


Figure 4.7: Positive Directions of Shell Stress Resultants

Stress resultants:

TE(3, 3)	Transformation matrix to element coordinate system
Nx(nofcol)	Force per length resulting from σ_x
Ny(nofcol)	Force per length resulting from σ_y
Nxy(nofcol)	Force per length resulting from τ_{xy}
Mx(nofcol)	Bending moment per length resulting from σ_x
My(nofcol)	Bending moment per length resulting from σ_y
Mxy(nofcol)	Twisting moment per length resulting from τ_{xy}
Qx(nofcol)	Transverse shear force per length resulting from τ_{xz}
Qy(nofcol)	Transverse shear force per length resulting from τ_{yz}

The positive directions of the stress resultants can be seen in Figure 4.7. A detailed definition of the stress resultants can be found in the Theory Manual.

General Stress Results

The stresses are given with respect to the global coordinate system.

sigx(nofpnt,nofcol)	Stress component σ_x
sigy(nofpnt,nofcol)	Stress component σ_y
sigz(nofpnt,nofcol)	Stress component σ_z
tauxy(nofpnt,nofcol)	Stress component τ_{xy}
tauyz(nofpnt,nofcol)	Stress component τ_{yz}
tauxz(nofpnt,nofcol)	Stress component τ_{xz}

5 List of Functions

mfs_back.....28
mfs_beamsection.....17
mfs_beamstress.....36
mfs_eset2nset.....51
mfs_export.....36
mfs_freevib.....22
mfs_freq2time.....53
mfs_freqresp.....25
mfs_getresp.....39
mfs_getset.....51
mfs_import.....15
mfs_line.....16
mfs_linenodes.....16
mfs_mass.....21
mfs_massproperties.....22
mfs_meffmass.....24
mfs_midnodes.....16
mfs_modecont.....47
mfs_new.....21
mfs_newset.....51
mfs_peaks.....52
mfs_plot.....30
mfs_print.....32
mfs_randresp.....49
mfs_reductionerror.....23
mfs_results.....30
mfs_statresp.....21
mfs_stiff.....21
mfs_transresp.....26