User Manual

Mefisto 2.8

Volume 2: Aerodynamics

Contents

1 Introduction....2
2 Model Description....3
2.1 Description of the Input Data....3
2.1.2 Points....4
2.1.3 Lifting Surfaces....5
2.1.4 Controllers....7
2.1.5 Symmetry....8
2.1.6 Configurations....8
2.1.7 Mass properties....10
2.1.8 Motion....11
3 Functions....12
3.1 Model Definition....12
3.2 Analysis....15
3.3 Output....18
3.4 Utilities....29
4 List of Functions....31

1 Introduction

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

Currently, the only method implemented is the vortex lattice method to solve problems of incompressible linear aerodynamics. The following features are supported:

Elements:	Lifting surfaces
Analysis:	Steady aerodynamics
	Trim analysis
	Frequency response analysis

2 Model Description

The model is described by a GNU Octave structure that is input to function **mfs_new**.

In the following, ***variables*** written in italics can have any name or value. The names of **variables** 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 defined by the user:

```
model.type
     .subtype
     .points(:).id
               .coor(3)
     .ls(:).id
           .points(2)
           .chord(2)
           .alpha(2)
           .camber{2}
           .nx
           .typex
           .ny
           .typey
     .controls(:).name
                 .ls(:)
                 .factors(:)
     .symy
     .config(:).name
               .qdyn
               .alpha
               .beta
               .pitch
               .yaw
               .roll
               .cntname
               .ax
               .ay
               .az
               .racce
               .pacce
               .yacce
               .lincon{:}
     .mass.m
```

```
          .JS(3, 3)
          .cm(3)
    .cref
    .motion.refpnt(3)
           .heave
           .pitch
           .cntname
```

type	**String**	Model type: For aerodynamics, the model type is **"aero".**
subtype	**String**	The subtype further specifies the model type: **"vlm"** Vortex Lattice Method
points(:)	**Structure**	Structure array with point data (see page 4)
ls(:)	**Structure**	Structure array with the lifting surface data (see page 5)
controls(:)	**Structure**	Structure array with control surface data (see page 7)
symy	**Real**	*y*-Coordinate of symmetry plane (if any) (see page 8)
config(:)	**Structure**	Structure array with configuration data (see page 8)
mass	**Structure**	Mass properties (see page 10)
cref	**Real**	Reference chord length
motion	**Structure**	Structure defining the motion of the aerodynamic model (see page 11)

Remarks

1. The mass properties are needed in a trim analysis.
2. The reference chord length is needed for unsteady aerodynamics only.

2.1.2 Points

```
points.id
      .coor(3)
```

id	**Integer**	Point identifier
coor(3)	**Real Array**	Point coordinates $[x, y, z]$

Remarks

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

2.1.3 Lifting Surfaces

```
ls.id
  .points(2)
  .chord(2)
  .alpha(2)
  .camber{2}
  .nx
  .typex
  .ny
  .typey
```

id	**Integer**	Identifier of Lifting surface
points(2)	**Integer Array**	Identifiers of points P1 and P2 defining the leading edge (see Figure 2.1)
chord(2)	**Real Array**	Chord lengths c1 and c2 (see Figure 2.1); if the chord lengths are identical, only one value need be defined.
alpha(2)	**Real Array**	Rigging angles of incidence at P1 and P2 in degrees; if the angles are identical, only one value need be defined. (optional; default: 0)
camber{2}	**Cell Array**	Camber at P1 and P2 defined as piecewise polynomial; if the camber is identical at both ends, only one value need be defined. (optional)
nx, ny	**Integer**	Number of vortex trapezoids in *x*- respectively *y*-direction
typex	**String**	Division type in *x*-direction (optional): **"linear"** Linear division

		"cos"	Symmetric cosine division (Default)
typey	**String**	Division type in *y*-direction (optional):	
		"linear"	Linear division (Default)
		"cos"	Symmetric cosine division
		"cos>"	Cosine division with closer spacing at the end
		"<cos"	Cosine division with closer spacing at the beginning

Remarks

1. The *x*-axis is aligned with the free stream velocity.
2. The camber is defined by a piecewise polynomial structure (cf. GNU Octave function **mkpp**) that describes the normalized camber z_s/c as a function of the normalized coordinate x/c where c is the chord length. The Mefisto function **mfs_airfoil** can be used to get this piecewise polynomial structure from airfoil data.
3. If no camber is defined the airfoil is assumed to be symmetric.

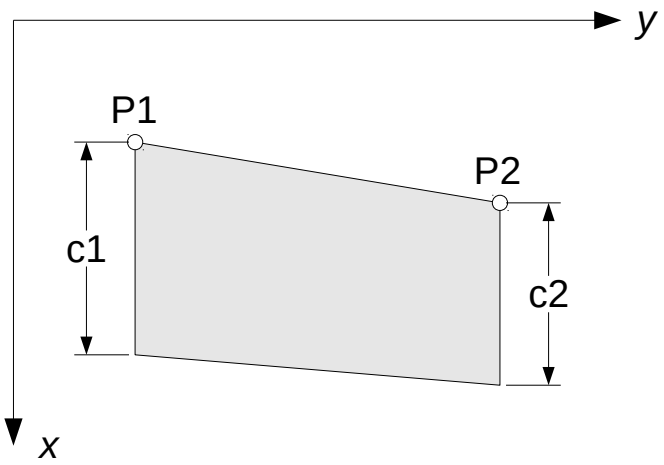


Figure 2.1: Lifting Surface

Example

```
c      =  1000; % Chord length at wing root

alpha1 =  0;    % Angle of incidence at wing root (deg.)
alpha2 = -1;    % Angle of incidence at wing tip (deg.)

naca = mfs_airfoil("NACA", 5, 51);

points = struct("id", {1, 2},
                "coor", {[0, 0, 0], [0, 800, 0]});

ls = struct("id", 10, "points", [1, 2], "chord", c,
            "nx", 11, "ny", 16, "camber", naca,
            "alpha", [alpha1, alpha2]);
```

This code defines a lifting surface with a constant chord length of 1000. The rigging angle of incidence of the inner chord is 0° and that of the outer chord is -1°. The camber is that of a NACA 251xx airfoil. In *x*-direction, the lifting surface is subdivided into 11 segments using a symmetric cosine spacing, i.e. the segments near the leading and trailing edge are smaller than those in the middle of the surface. In *y*-direction, the lifting surface is subdivided into 16 segments of equal length.

2.1.4 Controllers

```
controls.name
        .ls(:)
        .factors(:)
```

name	**String**	Name of the controller
ls(:)	**Integer Array**	List of lifting surface identifiers
factors(:)	**Real Array**	List of factors (optional: default is 1)

Remarks

1. Controller names must be unique.
2. A controller may consist of more than one lifting surface.
3. The factors multiply the controller angle.
4. A positive angle means a rotation of the control surface towards its negative side, i.e. downwards, if the normal vector on the control surface points upwards.
5. Controllers can be used to define configurations (see page 8).

Example

```
controls(1) = struct("name", "flap",    "ls", [11, 12]);
controls(2) = struct("name", "aileron", "ls", [11, 12]);
controls(2).factors = [0.1, 1];
model.controls = controls;
```

Both the flap and the aileron consist of lifting surfaces 11 and 12. The flap angle is applied to both surfaces. However, only 10 % of the aileron angle is applied to lifting surface 11, whereas 100 % are applied to lifting surface 12.

2.1.5 Symmetry

symy

The presence of this field indicates that the model is symmetric with respect to a plane that is perpendicular to the *y*-axis. It defines the *y*-coordinate of this plane. In most cases, its value is 0 indicating that the model is symmetric with respect to the *xz*-plane.

2.1.6 Configurations

```
config(:).name
         .qdyn
         .alpha
         .beta
         .pitch
         .yaw
         .roll
         .cntname
         .ax
         .ay
         .az
         .racce
         .pacce
         .yacce
         .lincon{:}
```

name	**String**	Name of the configuration
qdyn	**Real**	Dynamic pressure (Default: 1)
alpha	**Real**	Angle of attack (Default: 0)
beta	**Real**	Sideslip angle (Default: 0)
pitch	**Real**	Pitch rate divided by velocity

yaw	**Real**	Yaw rate divided by velocity
roll	**Real**	Roll rate divided by velocity
cntname	**Real**	Value of the controller (Default: 0)
ax	**Real**	Linear acceleration in *x*-direction
ay	**Real**	Linear acceleration in *y*-direction
az	**Real**	Linear acceleration in *z*-direction
racce	**Real**	Roll acceleration
pacce	**Real**	Pitch acceleration
yacce	**Real**	Yaw acceleration
lincon{:}	**Cell Array**	Cell array of structures defining linear constraints

Linear constraints

Linear constraints between trim parameters are used in a trim analysis. The constraints are linear equations of the form

$$\sum_n c_n t_n = r \quad .$$

Each structure defines one linear constraint. The field names correspond to the names of the trim parameters t_n. The values of the fields define the coefficients c_n of the linear equation. The value r of the right-hand side is defined in field **rhs**. Only values different from zero need be defined.

Remarks

1. Configuration names must be unique.
2. Angles are in degrees.
3. Pitch, yaw and roll rate are in radians per second.
4. Angular accelerations are in radians per second squared.
5. Each controller can be assigned a value, ***cntname*** being its name.
6. Controllers are defined by the **controls** structure array (see page 7).
7. Accelerations are used in a trim analysis.

Examples

```
alpha   = [0, 1];      % Angle of attack (degrees)
flap    = [-2, 0, 2]; % Flap angle (degrees)
aileron = [-4, 0, 4]; % Aileron angle (degrees)

nc = 1;
for l = 1 : length(alpha)
  for m = 1 : length(flap)
    for n = 1 : length(aileron)
      name = sprintf("Conf. %2d: alpha = %4.1f, ",
                     nc, alpha(l));
      name = [name, sprintf("flap = %4.1f, ", flap(m))];
      name = [name, sprintf("ail. = %4.1f", aileron(n))];
      config(nc++) = struct("name", name,
                            "alpha", alpha(l),
                            "flap", flap(m),
                            "aileron", aileron(n));
    endfor
  endfor
endfor

model.config = config;
```

This code defines 18 configurations with different angles of attack, flap angles and aileron angles. The configuration names are built from the values of the angles.

```
lincon{1} = struct("ay", sn, "az", cs, "rhs", g);
lincon{2} = struct("ay", sncs, "az", -ss, "pitch", vv);
lincon{3} = struct("ay", cc, "az", -sncs, "yaw", vv);
config(3).lincon = lincon;
```

This code defines the following three linear constraints between the trim parameters **ay**, **az**, **pitch** and **yaw**:

```
sn * ay + cs * az = g
sncs * ay - ss * az + vv * pitch = 0
cc * ay - sncs * az + vv * yaw = 0
```

2.1.7 Mass properties

```
mass.m
    .JS(3, 3)
    .cm(3)
```

m	**Real**	Mass of the aircraft

JS(3, 3)	**Real Array**	Inertia tensor with respect to the centre of mass
cm(3)	**Real Array**	Coordinates of the centre of mass

Remarks

1. Mass properties are needed in a trim analysis.
2. In an aeroelastic analysis, the mass properties are taken from the solid model.

2.1.8 Motion

```
motion.refpnt(3)
      .heave
      .pitch
       roll
      .cntname
```

refpnt(3)	**Real Array**	Coordinates of reference point for pitch motion (optional; default is origin)
heave	**Complex**	Complex amplitude of heave motion
pitch	**Complex**	Complex amplitude of pitch motion (in radians)
roll	**Complex**	Complex amplitude of roll motion (in radians)
cntname	**Complex**	Complex amplitude of controller angle (in radians)

Remarks

1. All types of motion can be combined.
2. The motion is used in a frequency response analysis.

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

```
[cam, con] = mfs_airfoil(type, arg1, arg2,…, "plot")
```

type	**String**	Airfoil type
arg1, arg2, …		Arguments that depend on the airfoil type
"plot"		If this keyword is present, then the function plots the camber. It must always be the last input argument.
cam	**Structure**	Structure defining a piecewise polynomial that describes the airfoil camber
con	**Structure**	Structure defining a piecewise polynomial that describes the airfoil contour (optional, only available with type **"fit"** and subtype **"contour"**)

The function returns a piecewise polynomial that describes the airfoil camber. The piecewise polynomial can be used as input to the GNU Octave functions **ppval,ppder** etc. The data refer to an airfoil with a chord length of 1.

Airfoil types

Type	Arguments		
"NACA"	NACA Airfoil		
	arg1 = 4	NACA Series 4:	
		arg2	Digits 1 and 2
		arg3	Start position of airfoil segment (optional)

Type	Arguments		
		arg4	End position of airfoil segment (optional)
	arg1 = 5	NACA Series 5:	
		arg2	Digits 2 and 3: Possible values are 10, 20, 21, 30, 31, 40, 41, 50 and 51
		arg3	Start position of airfoil segment (optional)
		arg4	End position of airfoil segment (optional)
"fit"	Spline fit approximation of an airfoil that is defined by points		
	arg1 = subtype	Subtype:	
		"camber"	The data define the camber of the airfoil.
		"contour"	The data define the contour of the airfoil.
	arg2 = data(:, 2)	Matrix with *x*- and *z*-coordinates of the camber or contour	
	arg3 = nsi	Number of camber spline intervals	
	arg3 = splopts	Structure with spline options	
	arg4 = x1	Start position of airfoil segment (optional)	
	arg5 = x2	End position of airfoil segment (optional)	

Fields of structure **splopts**

Name	Value	Default	Description
nbcam	**> 0**		Number of camber spline intervals
nbcon	**> 0**	**2*nbcam**	Number of contour spline intervals
smooth	**> 0**	**0**	Number of loops to smooth camber data

Remarks

1. If no start position is defined, the polynomial starts at the leading edge.

2. If no end position is defined, the polynomial ends at the trailing edge.
3. Both the start and the end position are defined as fractions of the chord length, i.e. their values must be between 0 and 1.
4. Type **"fit"**:
   a) Camber line points begin at the leading edge and end at the trailing edge.
   b) Contour line points begin and end at the trailing edge. The points on the upper surface precede the points on the lower surface.
   c) Either **nsi** or **splopts** may be defined.
   d) The number of camber spline intervals must always be defined.
   e) Smoothing is done by averaging the values of three adjacent points, except for the points of the leading and trailing edge. This operation is repeated as many times as requested by the value of **smooth**. Smoothing should be used with care because it changes the camber line.
   f) With subtype **"contour"** first the contour is approximated by a piecewise polynomial. Subsequently, the camber is computed from the contour. This is useful if only contour data are available or if the quality of the camber data is poor.
   g) The contour parameter is the angle, starting with 0 at the end and ending with 2π also at the end. Angles less than π correspond to points on the upper side and angles larger than π to points on the lower side of the airfoil.

Examples

```
naca = mfs_airfoil("NACA", 5, 51, "plot");
```

returns a piecewise polynomial that describes the camber of a NACA 251xx airfoil, and plots the camber line

```
airfoil_wing = mfs_airfoil("NACA", 5, 51, 0, 1-fcr);
airfoil_flap = mfs_airfoil("NACA", 5, 51, 1-fcr, 1);
```

The first call to the function returns the piecewise polynomial that describes the camber of the wing in front of the flap, and the second call returns the piecewise polynomial that describes the flap part of the NACA 251xx airfoil. **fcr** is a a real number defining the flap chord ratio.

```
data = dlmread("clarky-il.csv", ",", "A134:B194");
pp   = mfs_airfoil("fit", "camber", data, 7);
```

The coordinates of the camber of a Clark Y airfoil are read from a csv-file. Subsequently, the airfoil is approximated by a piecewise polynomial with seven intervals.

```
splopts = struct("nbcam", 8, "nbcon", 20);
data = dlmread("n63412-il.csv", ",", "A10:B60");
ppa  = mfs_airfoil("fit", "contour", data, splopts,
                   "plot");
```

The coordinates of the contour of a NACA 63-412 airfoil are read from a csv-file. The contour is approximated by a piecewise polynomial with 20 spline intervals. Subsequently, camber points are computed and approximated by a piecewise polynomial with 8 intervals. The contour is plotted together with the datapoints and the camber line.

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 (0 if errors occurred)

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

Name	Value	Default	Description
rtolp	**Real**	**1e-4**	Relative tolerance to check if points are coincident

Remarks

1. **rtolp** is used to find lifting surface trailing edges that are coincident with lifting surface leading edges. The absolute value of the tolerance is the relative tolerance multiplied by the largest chord length.

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

cmp	**Structure**	Structure with the component data

The function computes the primary results of steady aerodynamics. In case of the vortex-lattice method, the primary results are the vortices.

```
cmp = mfs_trim(cmp)
```

cmp	**Structure**	Structure with the component data

The function performs a trim analysis of a rigid aircraft.

```
cmp = mfs_freqresp(cmp, kred, opts)
```

cmp	**Structure**	Structure with the component data
kred(:)	**Real**	Array with reduced frequencies
opts	**Structure**	Structure with options (optional)

The function performs a frequency response analysis of a rigid wing or aircraft. The motion is defined by structure **motion** (see page 11).

List of options

Name	Value	Default	Description
msg	**File Handle**	**0**	File handle of message file (if 0, no messages are printed)
nx	**Integer**	**10**	Number of intervals per reference chord length in regular wake grid
m1	**Integer**	**4**	Factor to divide minimum control point distance from trailing edge to get size of first interval of graded wake grid
lenw	**Real**	**20**	Length of wake behind last control point in multiples of reference chord
ktol	**Real**	**1e-4**	Zero threshold for reduced frequencies

```
[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	Configuration 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)

Remarks

1. Back transformation is performed for those excitation frequencies that are closest to the excitation frequencies in the list.
2. Modal results are only available in an aeroelastic analysis.

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

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

The function computes secondary results that can be computed from the primary results. The items depend on the model type.

List of items

class = "statresp": Steady aerodynamics

Item	Description
"panel"	Panel results: pressure and force of each panel

class = "trim": Trim analysis

Item	Description
"panel"	Panel results: pressure and force of each panel

class = "freqresp": Frequency response analysis

Item	Description
"panel"	Panel results: pressure and force of each panel

3.3 Output

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

fid	**File Handle**	File handle of the output file
cmp	**Structure**	Structure with the component data
class	**String**	Class
item1, …	**String**	Items

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

List of items

class = "statresp": Steady aerodynamic results

Item	Description
"pressure"	Panel pressure
"vortex"	Vortex strength
"disp"	Displacements

class = "diverg": Static divergence analysis

Item	Description
"disp"	Displacements
"qdyn"	Dynamic pressure at divergence

class = "trim": Trim analysis

Item	Description
"params"	Trim parameters
"pressure"	Panel pressure
"vortex"	Vortex strength
"disp"	Displacements

class = "freqresp": Frequency response analysis

Item	Description
"pressure"	Panel pressure
"vortex"	Vortex strength

Remarks

1. Displacements are only available in an aeroelastic analysis.
2. In a trim analysis, the values of trim parameters that are angles are output in degrees.

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**	Items

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
"mesh"	The lifting surfaces are exported as flat plates (Default)
"camber"	Before exporting the mesh, the camber is applied to the lifting surfaces.
"normals"	The panel normal vectors are exported.

class = "statresp": Steady aerodynamic results

Item	Description
"pressure"	Pressure
"force"	Force vectors
"disp"	Displacements

`class = "diverg"`: Aeroelastic divergence analysis

Item	Description
`"disp"`	Displacements

`class = "trim"`: Trim results

Item	Description
`"pressure"`	Pressure
`"force"`	Force vectors
`"disp"`	Displacements

`class = "modes"`: Free vibration analysis of the solid model

Item	Description
`"disp"`	Displacements

`class = "flutter"`: Aeroelastic flutter analysis

Item	Description
`"disp"`	Displacements

`class = "freqresp"`: Aeroelastic frequency response analysis

Item	Description
`"disp"`	Displacements
`"pressure"`	Pressure

Remarks

1. Either item **`"mesh"`** or item **`"camber"`** may be present, bot not both. Both of these items can be combined with item **`"normals"`**.
2. Displacements are only available in an aeroelastic analysis.
3. Normal modes can be transferred from the solid to the aerodynamic model only if an aeroelastic model has been defined.
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("wing.msh", "msh", wing, "mesh", "camber");
```

writes the mesh of the lifting surfaces of component **wing** to the Gmsh file `wing.msh`. The mesh shows the camber of the lifting surfaces.

```
mfs_export("wing.pos", "msh", wing,
           "statresp", "pressure");
```

writes the aerodynamic pressure of a steady aerodynamic analysis of component wing to the Gmsh file `wing.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 = "mesh": Geometric data of the mesh

Item		
"area"	Area	
	arg1 = lsid(:)	List of lifting surface identifiers (optional; default is all lifting surfaces)
	out1 = A(:)	If no lifting surface identifers are defined, the total area is returned. Otherwise, an array with the areas of the lifting surfaces is returned.

class = "statresp": Steady aerodynamic results

Item	Description
"aeload"	Load resultants

Item	Description	
	arg1 = refpnt(3)	Coordinates of reference Point (optional; default is the origin)
	arg2 = lsid(:)	List of lifting surface identifiers (optional; default is all lifting surfaces)
	arg3 = cfg(:)	List of configuration numbers (optional; default is all configurations)
	out1 = F(3, :)	Resulting forces: columns correspond to configurations
	out2 = M(3, :)	Resulting moments: columns correspond to configurations

class = "diverg": Results of a static 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": Trim analysis results

Item	Description	
"aeload"	Load resultants	
	arg1 = refpnt(3)	Coordinates of reference Point (optional; default is the origin)
	arg2 = lsid(:)	List of lifting surface identifiers (optional; default is all lifting surfaces)
	arg3 = cfg(:)	List of configuration numbers (optional; default is all configurations)
	out1 = F(3, :)	Resulting forces: columns correspond to configurations
	out2 = M(3, :)	Resulting moments: columns correspond to configurations
"params"	Trim parameters	
	arg1 = cfg(:)	List of configuration numbers (optional; default is all configura-

Item	Description	
		tions)
	out1 = tp	Structure with the trim parameters (see page 23)

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

Item	Description	
"aeload"	Load resultants	
	arg1 = refpnt(3)	Coordinates of reference Point (optional; default is the origin)
	arg2 = lsid(:)	List of lifting surface identifiers (optional; default is all lifting surfaces)
	arg3 = ldc	Result case number (optional; default is 1)
	out1 = F(3, :)	Resulting forces: columns correspond to frequencies
	out2 = M(3, :)	Resulting moments: columns correspond to frequencies
	out3 = fk(:)	List of excitation frequencies or reduced frequencies

Fields of structure **tp**

nconf	**Integer**	Number of configurations
qdyn(nconf)	**Real**	Dynamic pressure
***tpnam*(nconf)**	**Real**	Values of trim parameter ***tpnam***

Remarks

1. In contrast to the printed output, values of trim parameters that are angles are in radians.
2. If the frequency response results are from an aeroelastic analysis, excitation frequencies are returned. If the results are from an aerodynamic analysis, reduced frequencies are returned.

Examples

```
Atotal = mfs_getresp(wing, "mesh", "area");
```

returns the total area of all lifting surfaces of component **wing**

```
A     = mfs_getresp(wing, "mesh", "area", [10, 20]);
Awing = sum(A);
```

The function returns the areas of the lifting surfaces 10 and 20 in the array **A(2)**. Variable **Awing** contains the sum of these areas.

```
[F, M] = mfs_getresp(wing, "statresp", "aeload");
```

returns the force and moment vectors resulting from all lifting surfaces for all configurations. Moments are computed with respect to the origin of the coordinate system.

```
[out1, out2, … ] =
    mfs_xydata(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 selected data that can be used for xy-plots, e.g. to plot the pressure along a wing section.

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

List of items

class = "statresp": Steady aerodynamic results

Item		Description
"pressure"	Aerodynamic pressure	
	arg1 = lsids(:)	List of identifiers of lifting surfaces in lifting surface group
	arg2 = ycolno	Panel column number
	arg3 = cfg(:)	List of configuration numbers (optional; default is all configurations)
	out1 = x(:)	*x*-coordinates
	out2 = p(:, :)	Pressure (columns correspond to configurations)
	out3 = y	*y*-coordinate (optional)
	out4 = z	*z*-coordinate (optional)

Item	Description	
	out5 = pids(:)	panel identifiers (optional)
"vortex"	Vortex strength Γ/v_∞	
	arg1 = lsids(:)	List of identifiers of lifting surfaces in lifting surface group
	arg2 = ycolno	Panel column number
	arg3 = cfg(:)	List of configuration numbers (optional; default is all configurations)
	out1 = x(:)	*x*-coordinates
	out2 = G(:, :)	Vortex strengths (columns correspond to configurations)
	out3 = y	*y*-coordinate (optional)
	out4 = ź	*z*-coordinate (optional)
	out5 = pids(:)	panel identifiers (optional)
"spanload"	Spanwise loads	
	arg1 = lsg{:}	Cell array with lifting surface groups
	arg2 = cfg(:)	List of configuration numbers (optional; default is all configurations)
	arg3 = refpnt(3)	Coordinates of reference Point (optional; default is the origin)
	out1 = s(:)	spanwise coordinates
	out2 = F(3,:,:)	Loads per length (2nd dim. refers to spanwise positions, 3rd dim. to configurations)
	out3 = M(3,:,:)	Moments per length (2nd dim. refers to spanwise positions, 3rd dim. to configurations)
	out4 = yz(2,:)	*y*- and *z*-coordinates
	out5 = w(:)	Panel column widths

class = "trim": Trim analysis results

Item	Description	
"pressure"	Aerodynamic pressure	
	arg1 = lsids(:)	List of identifiers of lifting surfaces in lifting surface group

Item	Description	
	arg2 = ycolno	Panel column number
	arg3 = cfg(:)	List of configuration numbers (optional; default is all configurations)
	out1 = x(:)	*x*-coordinates
	out2 = p(:, :)	Pressure (columns correspond to configurations)
	out3 = y	*y*-coordinate (optional)
	out4 = z	*z*-coordinate (optional)
	out5 = pids(:)	panel identifiers (optional)
"vortex"	Vortex strength Γ/v_∞	
	arg1 = lsids(:)	List of identifiers of lifting surfaces in lifting surface group
	arg2 = ycolno	Panel column number
	arg3 = cfg(:)	List of configuration numbers (optional; default is all configurations)
	out1 = x(:)	*x*-coordinates
	out2 = G(:, :)	Vortex strengths (columns correspond to configurations)
	out3 = y	*y*-coordinate (optional)
	out4 = z	*z*-coordinate (optional)
	out5 = pids(:)	panel identifiers (optional)
"spanload"	Spanwise loads	
	arg1 = lsg{:}	Cell array with lifting surface groups
	arg2 = cfg(:)	List of configuration numbers (optional; default is all configurations)
	arg3 = refpnt(3)	Coordinates of reference Point (optional; default is the origin)
	out1 = s(:)	spanwise coordinates
	out2 = F(3,:,:)	Loads per length (2nd dim. refers to spanwise positions, 3rd dim. to configurations)
	out3 = M(3,:,:)	Moments per length (2nd dim. refers to spanwise positions, 3rd

Item	Description	
		dim. to configurations)
	out4 = yz(2,:)	*y*- and *z*-coordinates
	out5 = w(:)	Panel column widths

class = "freqresp": Results of frequency response analysis

Item	Description	
"pressure"	Aerodynamic pressure	
	arg1 = lsids(:)	List of identifiers of lifting surfaces in lifting surface group
	arg2 = ycolno	Panel column number
	arg3 = kred(:)	List of reduced frequency or frequency indices (optional; default is all reduced frequencies)
	arg4 = ldc	Load case number (optional; default: 1)
	out1 = x(:)	*x*-coordinates
	out2 = p(:, :)	Pressure (columns correspond to frequencies)
	out3 = y	*y*-coordinate (optional)
	out4 = z	*z*-coordinate (optional)
	out5 = pids(:)	panel identifiers (optional)
"vortex"	Vortex strength Γ/v_∞	
	arg1 = lsids(:)	List of identifiers of lifting surfaces in lifting surface group
	arg2 = ycolno	Panel column number
	arg3 = kred(:)	List of reduced frequency or frequency indices (optional; default is all reduced frequencies)
	arg4 = ldc	Configuration number (optional; default: 1)
	out1 = x(:)	*x*-coordinates
	out2 = G(:, :)	Vortex strengths (columns correspond to configurations)
	out3 = y	*y*-coordinate (optional)
	out4 = z	*z*-coordinate (optional)
	out5 = pids(:)	panel identifiers (optional)
"spanload"	Spanwise loads	

`arg1 = lsg{:}`	Cell array with lifting surface groups
`arg2 = rfi(:)`	List of reduced frequency indices (optional; default is all)
`arg3 = refpnt(3)`	Coordinates of reference Point (optional; default is the origin)
`arg4 = ldc`	Result case number (optional; default is 1)
`out1 = s(:)`	spanwise coordinates
`out2 = F(3,:,:)`	Loads per length (2nd dim. refers to spanwise positions, 3rd dim. to frequencies)
`out3 = M(3,:,:)`	Moments per length (2nd dim. refers to spanwise positions, 3rd dim. to frequencies)
`out4 = yz(2,:)`	*y*- and *z*-coordinates
`out5 = w(:)`	Panel column widths

Remarks

1. Panel columns extend in *x*-direction (cf. Figure 3.1).
2. Lifting surfaces within one lifting surface group must be aligned in *x*-direction from leading to trailing edge (cf. Figure 3.1).
3. If the cell array of lifting surface groups contains one lifting surface group only, the curly braces (`{}`) can be omitted.
4. Usually, the frequency response results are from an aeroelastic analysis. In this case, the results are related to excitation frequencies. If the

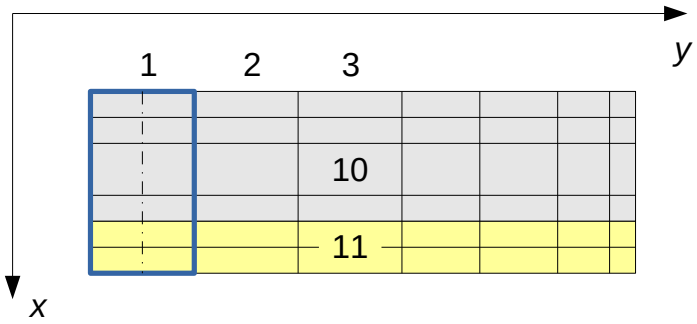


Figure 3.1: Panel Columns

frequency response results are from an aerodynamic analysis, the results are related to reduced frequencies.

Examples

```
[x(:, 1), cp(:, 1), y(1)] = ...
    mfs_xydata(wing, "statresp", "pressure",
               [10, 11], 1, 5);
```

returns the *x*-coordinates, the pressure of the 5th configuration and the *y*-coordinate of the first panel column of lifting surfaces 10 and 11 (cf. Figure 3.1)

```
refpnt = [xle + 0.25 * cr, 0, 0];
[s, F, M] = mfs_xydata(glider, "trim", "spanload",
                       {10, [11, 12]}, [], refpnt);
```

returns the spanwise coordinates and the forces and moments per length along the right wing for all configurations. The right wing consists of two lifting surface groups. The lifting surface group of the inner wing contains lifting surface 10 and that of the outer wing lifting surfaces 11 and 12. Moments are computed with respect to the reference point.

3.4 Utilities

```
[cp, xv, cL, cM] = mfs_vortex2d(x, camber, alpha)
[cp, xv, cL, cM] = mfs_vortex2d("steady", x, camber,
                                alpha)
[cp, xv, cL, cM] = mfs_vortex2d("harmonic", x, kred,
                                motion)
```

x(:)	**Real**	*x*-coordinates of interval boundaries of airfoil
camber	**Structure**	Piecewise polynomial describing the camber of the airfoil
alpha(:)	**Real**	List of angles of attack in degrees (optional; default is zero)
kred(:)	**Real**	List of reduced frequencies
motion	**Structure**	Structure defining harmonic motion
cp(:, :)	**Real**	Matrix with pressure coefficients: rows correspond to angles of attack or reduced frequencies
xv(:)	**Real**	*x*-coordinates of vortex points at which the pressure coefficients are given
cL(:)	**Real**	List of lift coefficents
cM(:)	**Real**	List of moment coefficients

The function computes the pressure coefficient, the lift coefficient and the moment coefficient of an airfoil using the discrete vortex method. Moment coefficients are computed with respect to the quarter point.

The function may be useful to perform a convergence study before starting 3-dimensional computations using the vortex-lattice method.

Fields of structure **`motion`**

`heave`	**`Complex`**		Amplitude of heave motion
`pitch`	**`Complex`**		Amplitude of pitch motion in radians (positive clockwise)
`flap`	**`Cell Array`**	**`{1}`**	Starting position of flap as fraction of the chord length (real)
		`{2}`	Amplitude of flap angle in radians (complex, positive down)

Remarks

1. All three types of motion can be combined.
2. Pitch motion is about the quarter point.

4 List of Functions

mfs_airfoil.....12
mfs_back.....16
mfs_export.....19
mfs_freqresp.....16
mfs_getresp.....21
mfs_new.....15
mfs_print.....18
mfs_results.....17
mfs_statresp.....15
mfs_trim.....16
mfs_vortex2d.....29
mfs_xydata.....24