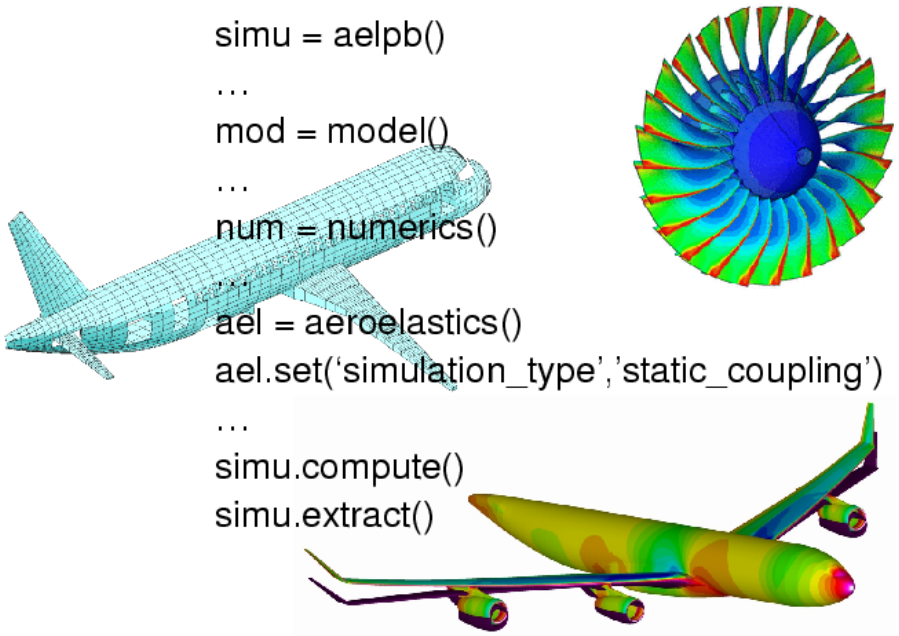
Aeroelastic Module User's Reference Manual



	Quality	Authors	For the reviewers	Approver
Function		Engineers	Head of quality	Project head, DDSS/MS head
Name		Ph. Girodroux Lavigne, A. Dugeai	A.M. Vuillot	L. Cambier, J.P. Grisval
Visa				

Software management : ELSA SCM
Applicability date : immediate
Diffusion : see last page

HISTORY

version edition	DATE	CAUSE and/or NATURE of EVOLUTION
1.0	Apr 11, 2007	Creation

CONTENTS

1. INTRODUCTION

This user's reference manual presents the aeroelastic functionalities which are available in the *elsA* software. These aeroelastic functionalities have been developed in an optional module of *elsA* by the "Structural Dynamics and Coupled Systems Department" of ONERA.

The development of this aeroelastic module, named "Ael", has been achieved adopting from the beginning a general framework, giving access to the simulation of different types of aeroelastic applications :

- non-linear and linearised harmonic forced motion simulations for a given mode ;
- fluid-structure static coupling using either a structural approach based on a "reduced flexibility matrix", or a modal structural approach ;
- fluid-structure dynamic coupling using presently only a modal approach for the structure.

Main parts of this document

- chapter 2 describes the additional description classes which are used in the Python language, and in its Python–*elsA* extension, to define an aeroelastic elsa application ;
- chapter 3 lists the *attributes* of each *class*, with their possible values and usage advice, if any ;
- appendix A presents the general features of an aeroelastic simulation.

  The following appendices present the usage of each specific aeroelastic simulations, the associated Python commands and detail the necessary inputs and the delivered outputs:
- appendix B : Harmonic forced motion
- appendix C : Static coupling
- appendix D : Dynamic coupling
- appendix E : Linearized simulation
- appendix F provides concrete examples of and Python–*elsA* scripts for each kind of aeroelastic simulation.

2. PYTHON DESCRIPTION CLASSES FOR ELSA-AEL

Additional description classes are needed in order to describe in the Python script the aeroelastic problem.

These description classes are listed hereafter with their *signature* (list of the constructor arguments). Constructor arguments are stored as attribute values, and listed in chapter 3 together with the other class attributes.

2.1 Constructors for the various classes

☐ `aelpb()` **aelpb**

For aeroelastic applications, the first object created must be an instance of this class, instead of a `cfdpb` ; this object corresponds to the problem which the user wants to solve ; all the computational operations will be done on this object.

```
<aelpb> = aelpb()
```

Ex. :

```
myProblem = aelpb(name='myProblem')
```

☐ `aelblock()` **aelblock**

`<aelblock>` objects must be used instead of classical `<block>` objects for all aeroelastic simulations with deforming blocks. Only aeroelastic applications using the concept of "aeroelastic injection boundary condition" (valid only for thin bodies and Euler type flow) may use classical `<block>` objects.

```
<aelblock> = aelblock()
```

Ex. :

```
wing = aelblock(name='wing')
```

☐ `aelitf(<block>, <window>` **aelitf**

Defines a local aeroelastic interface, associated to a `<window>` object and to a `<block>` object.

```
<aelitf> = aelitf(<block>, <window>)
<aelitf> = aelitf('<block_name>', '<window_name>')
```

Ex. :

```
wall1 = aelitf( blk1 ,  wnd1 , name='wall1')
wall2 = aelitf( blk3 , 'wnd9', name='wall2')
```

☐ `aeroelastics()` **aeroelastics**

Gathers all the elements defining the aeroelastic simulation.

```
<aeroelastics> = aeroelastics()
```

Ex. :

```
ael = aeroelastics(name='ael')
```

☐ strupart() — strupart

This class is used to define a partitioning of the structural nodes in "subsets of structural nodes". Each subset of structural nodes is affected to a given structural component. A similar partitioning of the "local aeroelastic interfaces" `<aelitf>` will also be defined, in order to link the aeroelastic interfaces to a structural component. This will allow to manage, for each structural component defined by this partition, the transfer of loads and displacements between the aerodynamic surface grid (stored on the local aeroelastic interfaces) and the structural grid.

```
<strupart> = strupart()
```

Ex. :

```
str1 = strupart(name='str1')
```

☐ displtransfer() — displtransfer

Specifies the technique which will be used by a given structural component for the transfer of displacements.

```
<displtransfer> = displtransfer()
```

Ex. :

```
displtrf11 = displtransfer(name='displtrf11')
```

3. ATTRIBUTES AND ATTRIBUTES VALUES (LISTED BY CLASS)

We describe here the attributes of the description classes defined in the Aeroelasticity module of elsA.

Remark : For classes which are derived from existing *elsA* classes, only specific additional attributes, or attributes whose value are important for the aeroelastic simulation, are described.

3.1 Attributes of the 'aelblock' class

□ `aelblock()` **aelblock**

<aelblock> objects and <block> objects have exactly the same attributes. It is however necessary to use <aelblock> objects instead of <block> objects in all aeroelastic simulations involving mesh deformation.

▷ `deform` **aelblock.deform**

Specifies if the block is deformable for ALE.
default value : *rigid*

```
<aelblock>.set('deform', '<deform>')

<deform> =
```

rigid : rigid block;
deformable : deformable block;
staticdeformable : static deformable block.

Remark : ALE blocks are required for unsteady computations with mesh deformation (value *deformable* of this attribute). ALE blocks are presently also required in *elsA* for steady state computations with mesh deformation. The value *staticdeformable* of this attribute allows then to avoid the computation of grid-velocities, and must be used for example for static fluid-structure coupling, and also for linearised harmonic forced motion computations.

▷ `deformation_type` **aelblock.deformation_type**

Type of deformation

```
<aelblock>.set('deformation_type', '<deformation_type>')

<deformation_type> =
```

aeroelastic : computed by aeroelastic deformation (needs Aeroelasticity module);
aeroelastic_lin : computed by aeroelastic deformation for linearized applications (needs the Ael and Lur modules);
function : specified time function.

3.2 Attributes of the 'aelboundary' class

□ aelboundary() — aelboundary

<aelboundary> objects are used in aeroelastic applications using the aelwallslip aeroelastic wallslip boundary condition.

▷ ael_interface — aelboundary.ael_interface

Aeroelastic interface for aeroelastic wallslip boundary condition

```
<aelboundary>.set('ael_interface', '<string>')
```

▷ choro_type — aelboundary.choro_type

Specify the type of chorochronic boundary condition.
default value : *other*

```
<aelboundary>.set('choro_type', '<string>')
```

```
<choro_type> =
```

aeroelastic : specify the use of an aeroelasticity chorochronic boundary condition;
other : specify the use of a standard chorochronic boundary condition;

3.3 Attributes of the 'aelinit' class

□ aelinit() — aelinit

<aelinit> objects are used in aeroelastic linearized simulations to define the initial steady state fields.

▷ args — aelinit.args

Macro-attribute of the constructor arguments ; look below for individual definitions.

```
<aelinit>.set('args', <args>)
args = [window_name]
```

▷ window_name — aelinit.window_name

Name of the corresponding window.
default value : *<empty string>=″*

```
<aelinit>.set('window_name', '<string>')
```

▷ `format` `aelinit.format`

Record structure of the files for the current object.
default value : *`fmt_tp`*

```
<aelinit>.set('format', '<format>')
```

```
<format> =
```

`bin_v3d` : unformatted file (Voir3D);
`bin_v3d_i4_r4` : as above, using `i4` and `r4` precisions ;
`fmt_tp` : formatted file (Tecplot);
`fmt_v3d` : formatted file (Voir3D).

▷ `steady_field_file` `aelinit.steady_field_file`

Name of the file providing the initial steady state fields for linearised simulations.

```
<aelinit>.set('steady_field_file', '<string>')
```

3.4 Attributes of the 'aelitf' class

□ `aelitf()` `aelitf`

An `<aelitf>` object must be defined for each deformable window in aeroelastic simulations. The attributes which must be defined depend on the type of structural model used in the simulation.

▷ `args` `aelitf.args`

Macro-attribute of the constructor arguments ; look below for individual definitions.

```
<aelitf>.set('args', <args>)
args = [block_name, window_name]
```

▷ `block_name` `aelitf.block_name`

Name of the corresponding block.
default value : *`<empty string>="`*

```
<aelitf>.set('block_name', '<string>')
```

▷ `window_name` `aelitf.window_name`

Name of the corresponding window.
default value : *`<empty string>="`*

```
<aelitf>.set('window_name', '<string>')
```

▷ `displ_group` **aelitf.displ_group**

Name of the structural group for displacement transfer

```
<aelitf>.set('displ_group', '<string>')
```

▷ `force_group` **aelitf.force_group**

Name of the structural group for load transfer

```
<aelitf>.set('force_group', '<string>')
```

▷ `association_type` **aelitf.association_type**

Defines direction for structural force nodes association
default value : *`min_distance`*

```
<aelitf>.set('association_type', '<string>')

<association_type> =
```

`min_distance` : selects the 3D minimum distance for association;
`min_x_distance` : selects the X-wise minimum distance for association;
`min_y_distance` : selects the Y-wise minimum distance for association;
`min_z_distance` : selects the Z-wise minimum distance for association;

▷ `format` **aelitf.format**

Record structure of the files for the current object.
default value : *`fmt_tp`*

```
<aelitf>.set('format', '<format>')

<format> =
```

`bin_v3d` : unformatted file (Voir3D);
`fmt_tp` : formatted file (Tecplot);
`fmt_v3d` : formatted file (Voir3D).

▷ `mode_file` **aelitf.mode_file**

Name of the file providing the mode shapes.

```
<aelitf>.set('mode_file', '<string>')
```

3.5 Attributes of the 'aelpb' class

□ aelpb() — aelpb

An <aelpb> object must be defined instead of a <cfdpb> object to access the specific description classes and keywords which have been defined to perform aeroelastic simulations.

▷ attachments — aelpb.attachments

Macro-attribute of the linked objects (through attach() call) ; look below for individual definitions.

```
<aelpb>.set('attachments', <attachments>)
attachments = [global_model, global_numerics, global_aeroelastics]
```

▷ global_model — aelpb.global_model

Name of model object associated by <aelpb>.attach().

```
<aelpb>.set('global_model', '<string>')
```

▷ global_numerics — aelpb.global_numerics

Name of numerics object associated by <aelpb>.attach().

```
<aelpb>.set('global_numerics', '<string>')
```

▷ global_aeroelastics — aelpb.global_aeroelastics

Name of <aeroelastics> object associated by <aelpb>.attach().

```
<aelpb>.set('global_aeroelastics', '<string>')
```

3.6 Attributes of the 'aeroelastics' class

□ aeroelastics() — aeroelastics

The attributes of this class allow to describe the type of aeroelastic simulation, the structural approach, and all the global parameters of the simulation.

▷ fluid_units — aeroelastics.fluid_units

Macro-attribute of the fluid units ; look below for individual definitions.

```
<aeroelastics>.set('fluid_units', <fluid_units>)
fluid_units = [fluid_density_unit, fluid_length_unit,
               fluid_velocity_unit, fluid_temperature_unit]
```

▷ `fluid_density_unit` **aeroelastics.fluid_density_unit**

Fluid density unit. Defines the adimensionalizing factor for fluid density (in SI units).

```
<aeroelastics>.set('fluid_density_unit', <float>)
```

▷ `fluid_length_unit` **aeroelastics.fluid_length_unit**

Fluid length unit. Defines the adimensionalizing factor for fluid length (in SI units).

```
<aeroelastics>.set('fluid_length_unit', <float>)
```

▷ `fluid_velocity_unit` **aeroelastics.fluid_velocity_unit**

Fluid velocity unit. Defines the adimensionalizing factor for fluid velocity (in SI units).

```
<aeroelastics>.set('fluid_velocity_unit', <float>)
```

▷ `fluid_temperature_unit` **aeroelastics.fluid_temperature_unit**

Fluid temperature unit. Defines the adimensionalizing factor for fluid temperature (in SI units).

```
<aeroelastics>.set('fluid_temperature_unit', <float>)
```

▷ `structure_units` **aeroelastics.structure_units**

Macro-attribute of the structure units ; look below for individual definitions.

```
<aeroelastics>.set('structure_units', <structure_units>)
structure_units = [structure_mass_unit, structure_length_unit,
                   structure_time_unit, structure_temperature_unit]
```

▷ `structure_mass_unit` **aeroelastics.structure_mass_unit**

Structure mass unit. Defines the adimensionalizing factor for structural mass (in SI units).

```
<aeroelastics>.set('structure_mass_unit', <float>)
```

▷ `structure_length_unit` **aeroelastics.structure_length_unit**

Structure length unit. Defines the adimensionalizing factor for structural length (in SI units).

```
<aeroelastics>.set('structure_length_unit', <float>)
```

▷ `structure_time_unit` **aeroelastics.structure_time_unit**

Structure time unit. Defines the adimensionalizing factor for structural time (in SI units).

```
<aeroelastics>.set('structure_time_unit', <float>)
```

▷ `structure_temperature_unit` **aeroelastics.structure_temperature_unit**

Structure temperature unit. Defines the adimensionalizing factor for structural temperature (in SI units).

```
<aeroelastics>.set('structure_temperature_unit', <float>)
```

▷ **mesh_deformation** — aeroelastics.mesh_deformation

Macro-attribute of the parameters for mesh deformation ; look below for individual definitions.

```
<aeroelastics>.set('mesh_deformation', <mesh_deformation>)
mesh_deformation = [mesh_deformation_type, mesh_deformation_technique]
```

▷ **mesh_deformation_type** — aeroelastics.mesh_deformation_type

Type of mesh deformation technique.

default value : *none*

```
<aeroelastics>.set('mesh_deformation_type', '<mesh_deformation_type>')
```

```
<mesh_deformation_type> =
```

computed : mesh deformation is computed from wall displacements at each physical time step in unsteady forced motion, and at each coupling iteration in the case of static or dynamic coupling simulation ;

interpolated : mesh deformation is a linear combination of modal mesh deformations computed in a pre-processing step ;

none : no mesh deformation.

▷ **mesh_deformation_technique** — aeroelastics.mesh_deformation_technique

Technique of mesh deformation.

default value : *move3d*

```
<aeroelastics>.set('mesh_deformation_technique', '<mesh_deformation_technique>')
```

```
<mesh_deformation_technique> =
```

move3d : mesh deformation using structural analogy;

tfi : mesh deformation using mixed analytical/transfinite interpolation.

▷ **mesh_deformation_data_file** — aeroelastics.mesh_deformation_data_file

Input file for mesh deformation (mmove3d).

```
<aeroelastics>.set('mesh_deformation_data_file', '<string>')
```

▷ **mesh_deformation_output_file** — aeroelastics.mesh_deformation_output_file

Output file for mesh deformation (mmove3d.out).

```
<aeroelastics>.set('mesh_deformation_output_file', '<string>')
```

▷ **structural_model** — aeroelastics.structural_model

Type of structural modeling.

```
<aeroelastics>.set('structural_model', '<structural_model>')
```

```
<structural_model> =
```

flexibility : reduced flexibility matrix approach ;
modal : modal approach ;
projected : non-modal projected model approach ;
finite_element : full finite element model approach.

▷ **modal_basis_kind** aeroelastics.modal_basis_kind

Kind of the modal basis.
default value : *real_modes*

```
<aeroelastics>.set('modal_basis_kind', '<modal_basis_kind>')
```

```
<modal_basis_kind> =
```

real_modes : real modes;
complex_modes : complex modes.

▷ **modal_basis_size** aeroelastics.modal_basis_size

Size of the modal basis.
default value : *1*

```
<aeroelastics>.set('modal_basis_size', <int>)
```

▷ **modal_data** aeroelastics.modal_data

Location of modal data.
default value : *on_ael_interface*

```
<aeroelastics>.set('modal_data', '<modal_data>')
```

```
<modal_data> =
```

on_ael_interface : modal data provided on aeroelastic interfaces;
on_structural_grid : modal data provided on structural grid.

▷ **mode_file** aeroelastics.mode_file

Mode file in the case of modal data defined on the structural grid.

```
<aeroelastics>.set('mode_file', '<string>')
```

▷ **format** aeroelastics.format

Record structure of the files for the current object.
default value : *fmt_tp*

```
<aeroelastics>.set('format', '<format>')
```

```
<format> =
```

bin_v3d : unformatted file (Voir3D);
fmt_tp : formatted file (Tecplot);
fmt_v3d : formatted file (Voir3D).

▷ `simulation_type` **aeroelastics.simulation_type**

Type of aeroelastic simulation

```
<aeroelastics>.set('simulation_type', '<simulation_type>')
```

```
<simulation_type> =
```

forced_motion : forced-motion aeroelastic simulation;
static_coupling : fluid-structure static coupling simulation;
dynamic_coupling : fluid-structure dynamic coupling simulation;
linearized : linearized Euler or Navier-Stokes aeroelastic simulation.

▷ `forced_motion_type` **aeroelastics.forced_motion_type**

Subtype of aeroelastic forced motion simulation.
default value : *harmonic*

```
<aeroelastics>.set('forced_motion_type', '<forced_motion_type>')
```

```
<forced_motion_type> =
```

file_driven : file_driven motion;
harmonic : harmonic motion.

▷ `forced_motion_init` **aeroelastics.forced_motion_init**

Type of aeroelastic forced motion simulation initializing procedure.
default value : *none*

```
<aeroelastics>.set('forced_motion_init', '<forced_motion_init>')
```

```
<forced_motion_init> =
```

sinesquare : uses sinesquare smoothing in the first fourth of first period;
none : no initial smoothing procedure.

▷ `complex_excitation` **aeroelastics.complex_excitation**

Subtype of aeroelastic forced motion simulation for complex modes.
default value : *direct*

```
<aeroelastics>.set('complex_excitation', '<complex_excitation>')
```

```
<complex_excitation> =
```

direct : combined complex excitation is $Re(e^{j\omega t}\Phi_i)$;
conjugate : combined complex excitation is $Re(e^{j\omega t}\bar{\Phi}_i)$;

▷ `mode_number` **aeroelastics.mode_number**

Mode number of forced motion simulation.
default value : *1*

```
<aeroelastics>.set('mode_number', <int>)
```

▷ `frequency` **aeroelastics.frequency**

Frequency of forced motion simulation

```
<aeroelastics>.set('frequency', <float>)
```

▷ `initial_phase` **aeroelastics.initial_phase**

Initial phase of forced motion simulation.
default value : *0.0*

```
<aeroelastics>.set('initial_phase', <float>)
```

▷ `modal_amplitude` **aeroelastics.modal_amplitude**

Modal amplitude of forced motion simulation.
default value : *1.0*

```
<aeroelastics>.set('modal_amplitude', <float>)
```

▷ `fourier_frequency_file` **aeroelastics.fourier_frequency_file**

Fourier Analysis Frequency file (file driven forced motion only)

```
<aeroelastics>.set('fourier_frequency_file', '<string>')
```

▷ `static_frequency` **aeroelastics.static_frequency**

Frequency of displacements update in static simulation.
default value : *1*

```
<aeroelastics>.set('static_frequency', <int>)
```

▷ `static_nbiterini` **aeroelastics.static_nbiterini**

Number of fluid iterations before starting the static coupling loop.
default value : *1*

```
<aeroelastics>.set('static_nbiterini', <int>)
```

▷ `static_relaxation` **aeroelastics.static_relaxation**

Relaxation factor for displacements in static coupling simulation.
default value : *1.0*

```
<aeroelastics>.set('static_relaxation', <float>)
```

▷ `structural_grid_file` **aeroelastics.structural_grid_file**

Structural grid file

```
<aeroelastics>.set('structural_grid_file', '<string>')
```

▷ `structural_matrix_file` **aeroelastics.structural_matrix_file**

Structural matrices file

```
<aeroelastics>.set('structural_matrix_file', '<string>')
```

▷ `structural_vars_file` **aeroelastics.structural_vars_file**

Displacements, velocities, and force file (on structural grid)

```
<aeroelastics>.set('structural_vars_file', '<string>')
```

▷ `association_type` **aeroelastics.association_type**

Defines how fluid wall interfaces are associated to structural force nodes.
default value : *`min_distance`*

```
<aeroelastics>.set('association_type', '<association_type>')
```

```
<association_type> =
```

`min_distance` : association to closest structural force node;
`x_dir` : association to closest structural force node in x-direction;
`y_dir` : association to closest structural force node in y-direction;
`z_dir` : association to closest structural force node in z-direction.

▷ `dynamic_pressure_ref` **aeroelastics.dynamic_pressure_ref**

Dynamic pressure reference.
default value : *`1.0`*

```
<aeroelastics>.set('dynamic_pressure_ref', <float>)
```

▷ `surface_ref` **aeroelastics.surface_ref**

Surface reference.
default value : *`1.0`*

```
<aeroelastics>.set('surface_ref', <float>)
```

▷ `nit_mecaloop` **aeroelastics.nit_mecaloop**

Number of structural iterations for dynamic coupling.
default value : *2*

```
<aeroelastics>.set('nit_mecaloop', <int>)
```

3.7 Attributes of the 'displtransfer' class

□ displtransfer() displtransfer

This class describes how displacements are transfered from the structural grid to the aerodynamic surface grid, using interpolation or fitting techniques. The transfer of displacements is achieved on a component-wise basis, each component being defined by the displ_group attribute. The technique used to transfer the displacements for a specific component is defined by the displ_transfer_type attribute.

▷ method displtransfer.method

Macro-attribute of the transfer method definition ; look below for individual definitions.

```
<displtransfer>.set('method', <method>)
method = [displ_transfer_type, displ_group]
method = [displ_transfer_type, displ_group, u_vector]
method = [displ_transfer_type, displ_group, u_vector, v_vector]
method = [displ_transfer_type, displ_group, u_vector, v_vector, u_order, v_order]
```

▷ displ_transfer_type displtransfer.displ_transfer_type

Type of method used for displacement transfer.

```
<displtransfer>.set('displ_transfer_type', '<displ_transfer_type>')
```

```
<displ_transfer_type> =
```

infinite_plate : infinite-plate interpolation technique ;
infinite_volume : infinite-volume interpolation technque ;
join : join technique (to get continuity of displacements between 2 structural components ;
polynomial_1d : 1d-polynomial fitting technique ;
polynomial_2d : 2d-polynomial fitting technique ;
solid : transfer of displacements assuming solid motion ;
solid_section : transfer of displacements assuming solid deformation per section.

▷ displ_group displtransfer.displ_group

Structural group name for displacement transfer.

```
<displtransfer>.set('displ_group', '<string>')
```

▷ u_vector displtransfer.u_vector

Macro-attribute of the vector defining u-direction ; look below for individual definitions.

```
<displtransfer>.set('u_vector', <u_vector>)
u_vector = [u_vector_x, u_vector_y, u_vector_z]
```

▷ u_vector_x displtransfer.u_vector_x

X-component of u-vector.

```
<displtransfer>.set('u_vector_x', <float>)
```

▷ **u_vector_y** — displtransfer.u_vector_y

Y-component of u-vector.

```
<displtransfer>.set('u_vector_y', <float>)
```

▷ **u_vector_z** — displtransfer.u_vector_z

Z-component of u-vector.

```
<displtransfer>.set('u_vector_z', <float>)
```

▷ **v_vector** — displtransfer.v_vector

Macro-attribute of the vector defining v-direction ; look below for individual definitions.

```
<displtransfer>.set('v_vector', <v_vector>)
v_vector = [v_vector_x, v_vector_y, v_vector_z]
```

▷ **v_vector_x** — displtransfer.v_vector_x

X-component of v-vector.

```
<displtransfer>.set('v_vector_x', <float>)
```

▷ **v_vector_y** — displtransfer.v_vector_y

Y-component of v-vector.

```
<displtransfer>.set('v_vector_y', <float>)
```

▷ **v_vector_z** — displtransfer.v_vector_z

Z-component of v-vector.

```
<displtransfer>.set('v_vector_z', <float>)
```

▷ **u_order** — displtransfer.u_order

Order of u-polynomes.

```
<displtransfer>.set('u_order', <int>)
```

▷ **v_order** — displtransfer.v_order

Order of v-polynomes.

```
<displtransfer>.set('v_order', <int>)
```

▷ `fixed_point_file` **displtransfer.fixed_point_file**

File defining fixed points, when using the infinite-volume technique>

```
<displtransfer>.set('fixed_point_file', '<string>')
```

▷ `format` **displtransfer.format**

Record structure of the files for the current object.
default value : *`fmt_tp`*

```
<displtransfer>.set('format', '<format>')
```

```
<format> =
```

`bin_v3d` : unformatted file (Voir3D) ;
`fmt_tp` : formatted file (Tecplot) ;
`fmt_v3d` : formatted file (Voir3D).

▷ `join_group_a` **displtransfer.join_group_a**

Structural group name for join method of displacement transfer.

```
<displtransfer>.set('join_group_a', '<string>')
```

▷ `join_group_b` **displtransfer.join_group_b**

Structural group name for join method of displacement transfer.

```
<displtransfer>.set('join_group_b', '<string>')
```

3.8 Attributes of the '`strupart`' class

□ `strupart()` **strupart**

This description class is used to define a partitioning of the structural nodes in subset of nodes. Each subset of nodes is linked to a structural component, or structural sub-group, for the transfer of loads and for the transfer of displacements. It may be useful to have a different decomposition of the whole structure in individual components on one hand for load transfer and on the other hand for displacement transfer.

The `force_group` attribute defines the structural component to which the subset of nodes is linked for the transfer of forces. All the structural nodes linked to a given structural component are associated, for the transfer of loads, with the aeroelastic interfaces which are linked to the same structural component (see attributes of `aelitf` class).

The `displ_group` attribute allows, in the same way, to associate subsets of structural nodes and subsets of aeroelastic interfaces for the transfer of displacements.

▷ `part` **strupart.part**

Macro-attribute of the structural part ; look below for individual definitions.

```
<strupart>.set('part', <part>)
part = [ael_family, force_group, displ_group, first_label, last_label]
```

▷ **ael_family** — **strupart.ael_family**

Structural family name.

```
<strupart>.set('ael_family', '<string>')
```

▷ **force_group** — **strupart.force_group**

Structural group name for force transfer.

```
<strupart>.set('force_group', '<string>')
```

▷ **displ_group** — **strupart.displ_group**

Structural group name for displacement transfer.

```
<strupart>.set('displ_group', '<string>')
```

▷ **first_label** — **strupart.first_label**

First Nastran label of structural part.
default value : *1*

```
<strupart>.set('first_label', <int>)
```

▷ **last_label** — **strupart.last_label**

Last Nastran label of structural part.
default value : *1*

```
<strupart>.set('last_label', <int>)
```

Appendix A. AEROELASTIC MODULE GENERAL USAGE

A.1 Using the *elsA* Aeroelastic module

The main components to be defined in an aeroelastic simulation are as follows:

- the aeroelastic problem, which takes place of the usual cfd problem;
- the aeroelastic interface, which defines the fluid boundaries interacting with the structural model;
- the aeroelastic simulation, which defines the kind of aeroelastic simulation;
- the structural model;
- the fluid and structure normalization data;
- the fluid-structure transfer data (in the non-modal case);
- the mesh deformation control data.

These informations are prescribed through the use of the aeroelastic module following objects :

- `<aelpb>` : aeroelastic problem unique instance;
- `<aeroelastics>` : aeroelastic simulation unique instance;
- `<aelitf>` : aeroelastic individual interface matching with a fluid window;
- `<aelblock>` : aeroelastic block (deformable or not);
- `<aelbnd>` : aeroelastic boundary (for unsteady aeroelastic injection);
- `<strupart>` : partitionning of structural node set (in the non-modal case);
- `<displtransfer>` : displacement transfer technique from structure to fluid (in the non-modal case).

Depending on the choosen aeroelastic simulation, some of these objects are or not activated. However some of them are mandatory: `<aelpb>`, `<aeroelastics>`, `<aelitf>`.

A.2 General overview of the aeroelastic Python sequence

We present here an overall description of the Python items necessary for an aeroelastic simulation.

- Import the *Ael* python module :
```
from elsA_Ael_user import *
```

- Declare an <aelpb> instead of the <cfdpb> :
```
myAelpb = aelpb(name='myAelpb')
```

- Declare an <aeroelastics> object :
```
ael = aeroelastics(name='ael')
```

- Define the aeroelastic type of simulation and global parameters :
```
ael.set('simulation_type','forced_motion')
```

- Define the aeroelastic normalization data :
```
ael.set('fluid_velocity_unit',Uref)
```

- Define the structural model :
```
ael.set('structural_model','modal')
```

- If necessary, define the mesh deformation technique :
```
ael.set('mesh_deformation_technique','tfi')
```

- Define one or several local aeroelastic interfaces <aelitf> associated to fluid windows :
```
aelitf1 = aelitf(name='aelitf1','blk2','win38')
```

- Associate the <aeroelastics> object with the problem :
```
myAelpb.set('global_aeroelastics','ael')
```

- Run the aeroelastic simulation :
```
myAelpb.compute()
```

Appendix B. HARMONIC FORCED MOTION SIMULATION

B.1 General description of harmonic motion simulation

The harmonic forced motion simulation provides the unsteady aerodynamic response to an harmonic excitation of the aeroelastic interface. It implies the use of a modal structural model, and the specification of the structural modal basis. A specific mode is selected for the simulation. The motion is prescribed at a given frequency, starting with a prescribed initial phase, and with a specified amplitude coefficient. The unsteady motion of the aeroelastic interface drives the aerodynamic computation through the use of an aeroelastic injection boundary condition in the case of fixed mesh (in Euler), or through mesh deformation. Global, dual and Gear time stepping for the fluid are available. The specific output of the aeroelastic simulation consists in the unsteady and first harmonic generalized forces, and in the first harmonic of the static pressure on the aeroelastic interface.

B.1.1 Prescribing a forced motion simulation in the python script

The python scripts used for harmonic forced motion simulations follow the general rules described before. But specific items are necessary to provide data for the proper definition of the simulation.

- Specify the kind of aeroelastic simulation :
```
ael.set('simulation_type','forced_motion')
```
- Specify the subtype of aeroelastic forced motion simulation :
```
ael.set('forced_motion_type','harmonic')
```
- Specify the kind of initializing procedure for the harmonic aeroelastic forced motion simulation :
```
ael.set('forced_motion_init','none')
```
- Specify the mandatory modal structural model :
```
ael.set('structural_model','modal')
```
- Specify the kind of the modal basis (real or complex) :
```
ael.set('modal_basis_kind','real_modes')
```
- Specify the size of the modal basis :
```
ael.set('modal_basis_size',4)
```
- Selects the excitation mode :
```
ael.set('mode_number',1)
```
- Selects the excitation amplitude coefficient :
```
ael.set('modal_amplitude',1.)
```
- Selects the initial phase in radian :
```
ael.set('initial_phase',phi0radian)
```

B.1.2 Defining the modal data on the aeroelastic interface

The structural modal basis must be defined on the aeroelastic interface. This is achieved by defining an associated mode file on each local aeroelastic interface `<aelitf>`.

- Defines a local aeroelastic interface :
```
aelitf1 = aelitf(name='aelitf1','blk2','win38')
```
- Associates a mode file to the `<aelitf>` :
```
aelitf1.set('mode_file','modeai1.dat')
```

See section Input and output Files for information on the mode file format.

B.1.3 Using the aeroelastic injection boundary condition in fixed meshes

Aeroelastic injection boundary condition prescribing the harmonic velocity of the aeroelastic interfaces nodes must be prescribed in the case of fixed meshes.
- Defines an aeroelastic injection boundary :

```
aelbnd1 = aelboundary(name='aelbnd1','blk2','win38')
aelbnd1.set('type','aelwallslip')
```

- Link boundary to the associated local aeroelastic interface :

```
aelbnd1.set('ael_interface','aelitf1')
```

B.1.4 Activating mesh deformation for aeroelasticity

In order to activate mesh deformation, blocks must be declared as deformable aeroelastic blocks. Mesh deformation control informations must also be provided to the solver. In the case of the TFI technique *tfi*, no additional data is needed. In the case of the structural analogy technique *move3d*, an additional control file is mandatory. The mesh deformation technique may be *interpolated* or *computed*. This feature is compatible with both mesh deformation utilities, i.e. *tfi* or *move3d*. When *interpolated*, the mesh deformation generates extremal modal mesh deformations in a pre-processing step and only performs interpolation at mesh deformation time steps. When *computed*, the full mesh deformation process is conducted at each mesh deformation time step. Interpolation is of course cheaper than computation, but the deformation is assumed to be linear between the original and the extremal deflected positions (valid for small deflections).

- Defines a deforming aeroelastic block :

```
blk0001 = aelblock(name='blk0001')
blk0001.set('deform','deformable')
blk0001.set('deformation_type','aeroelastic')
blk0001.set('freq_comp_vol',1)
blk0001.set('vol_comp_meth','hwd')
blk0001.submit
```

If structural analogy mesh deformation technique is selected :

- Selects the 'move3d' interpolated mesh deformation technique :

```
ael.set('mesh_deformation_technique','move3d')
ael.set('mesh_deformation_type','interpolated')
```

- Prescribes the 'move3d' mesh deformation control data and output file names :

```
ael.set('mesh_deformation_data_file','move3d.dat')
ael.set('mesh_deformation_output_file','move3d.out')
```

If TFI mesh deformation technique is selected :

- Selects the 'move3d' interpolated mesh deformation technique :

```
ael.set('mesh_deformation_technique','tfi')
ael.set('mesh_deformation_type','interpolated')
```

B.2 Input and output files

For forced motion simulations, the following input files are necessary :

B.2.1 Modal displacements file

This file describes for each mode the displacements on each node of a local aeroelastic interface. The format of this file is ASCII Tecplot POINT or BLOCK format. The aeroelastic interface nodes are described in the following order :

```
     DO K=k1,k2
       DO J=j1,j2
         DO I=i1,i2
```

```
TITLE = "Modal Displacements file on aeroelastic interface aelitf1"
VARIABLES = "hx" "hy" "hz"
ZONE T="mode 1" I=  51, J=12, K=1, F=POINT
  0.0000000E+00    0.0000000E+00  1.0000000E+00
  0.0000000E+00    0.0000000E+00  5.0000000E-01
  ...
ZONE T="mode 2" I=  51, J=12, K=1, F=POINT
  0.0000000E+00    2.0000000E+00  0.0000000E+00
  0.0000000E+00    3.0000000E+00  5.0000000E-02
  ...
```

B.2.2 Mesh deformation data file

When the TFI technique is used, no additional data is needed. In the case the structural analogy is used, the complete description of the move3d submodule is provided in a specific manual, which is not detailled here. Please refer to the corresponding document.
Several automatic outputs are delivered during or at the end of the forced motion simulation.

B.2.3 Generalized variables history file

This file provides the time histories of the generalized coordinate, velocity and aerodynamic forces integrated on the aeroelastic interface. The format of this file is ASCII Tecplot POINT format. One file is produced for each mode in the modal basis. The time history of the following variables are given : iteration index, generalized coordinate, generalized velocity, generalized force, physical time.
The name of the file is *gcvf_history_mode_XXX_on_YYY.dat* where *XXX* is the excitation mode number, and *YYY* the projecting mode number.

```
TITLE = "Modal History file for forced motion on mode 1"
VARIABLES = "iter" "coord" "velo" "forces" "time"
ZONE T="Observed forces for mode 1" I=  512, F=POINT
  0.1000000E+01  0.2000000E+00  0.0000000E+00 -0.5968558E-12  0.3125000E-02
  0.2000000E+01  0.1961570E+00 -0.2451577E+01 -0.2578687E+02  0.6250000E-02
  0.3000000E+01  0.1847759E+00 -0.4808941E+01 -0.2533036E+02  0.9375000E-02
  0.4000000E+01  0.1662939E+00 -0.6981501E+01 -0.1053807E+02  0.1250000E-01
  ...
```

B.2.4 First harmonic of the generalized aerodynamic forces file

This file gives the first harmonic analysis of the generalized aerodynamic forces integrated on the aeroelastic interface. The format of this file is ASCII Tecplot POINT format. A single file is produced during the simulation. One Zone of the Tecplot file gives one row of the GAF matrix at the end of the corresponding cycle. The name of the file is *harmonic_forces_mode_XXX.dat*

```
TITLE = "Forces Harmonic Analysis File for forced motion on mode 1"
VARIABLES = "Imod" "Jmod" "ReFiOnHj" "ImFiOnHj"
ZONE T="Window(s) : Observed forces for cycle 1",  I=2, F=POINT
  0.1000000E+01  0.1000000E+01  0.2729172E+04 -0.1207292E+04
  0.1000000E+01  0.2000000E+01 -0.1982318E+04  0.5697731E+03
ZONE T="Window(s) : Observed forces for cycle 2",  I=2, F=POINT
  0.1000000E+01  0.1000000E+01  0.2744234E+04 -0.8159578E+03
  0.1000000E+01  0.2000000E+01 -0.2069096E+04  0.2585280E+03
  ...
```

B.2.5 First harmonic of static pressure file

This file gives the first harmonic analysis of the static pressure on each node of a local aeroelastic interface. The format of this file is ASCII Tecplot POINT format. One file is produced for each local aeroelastic interface. One Zone of the Tecplot file gives the real, imaginary and mean value of static pressure at the end of the corresponding cycle. The name of the file is *harmonic_pressure_mode_XXX_aiZZZZ.dat* where *aiZZZ* is the name of the corresponding aeroelastic interface.

```
TITLE = "Pressure Harmonic Analysis File on aelitf1"
VARIABLES = "x" "y" "z" "pr" "pi" "pm"
ZONE T="Ps harmonic analysis on aelitf1 cycle 1 ",  I=51,  J=12, F=POINT
  0.10814E+00  0.20290E-01  0.34284E+00 -0.32821E+05  0.23898E+04  0.87939E+05
  0.10816E+00  0.20239E-01  0.34285E+00 -0.32521E+05  0.23850E+04  0.87947E+05
  0.10819E+00  0.20193E-01  0.34286E+00 -0.32304E+05  0.24105E+04  0.87961E+05
  ...
```

Appendix C. STATIC COUPLING SIMULATION

C.1 General description of static coupling simulations

The static coupling simulation solves the static problem of the aerodynamic and elastic forces equilibrium. It is solved using either the modal or projected structural model, or the reduced flexibility matrix approach. In the *modal* or *projected* case the projected stiffness matrix must be given. In the case of the *reduced flexibility matrix* approach, the user must provide a structural grid, a reduced flexibility matrix, and data controling the fluid-structure displacement and force transfers. The simulation is performed with a local time stepping scheme for the fluid over a number of time steps, with periodic fluid-structure coupling steps, whose frequency is prescribed. An under-relaxation coefficient may be used to stabilize the process in case of large displacements.

C.1.1 Prescribing a static coupling simulation in the python script

The python scripts used for static coupling simulations follow the general rules described before. But specific items are necessary to provide data for the proper definition of the simulation.

In the *modal* case the basic python commands are :
- Specify the kind of aeroelastic simulation :
```
ael.set('simulation_type','static_coupling')
```
- Specify the modal structural model :
```
ael.set('structural_model','modal')
```
- Specify the kind of the projection basis which is real :
```
ael.set('modal_basis_kind','real_modes')
```
- Specify the size of the projection basis :
```
ael.set('modal_basis_size',4)
```
- Specify the structural matrices file :
```
ael.set('structural_matrix_file','StructMatrices.dat')
```
- Specify the initial structural variables file :
```
ael.set('structural_vars_file','StructVars.dat')
```
- Selects the relaxation coefficient value :
```
ael.set('static_relaxation',.8)
```
- Selects the starting iteration index of the coupling process :
```
ael.set('static_nbiterini',200)
```
- Selects the number of CFD iterations between two coupling steps :
```
ael.set('static_frequency',200)
```

In the *projected* case the basic python commands are similar to the *modal* case except :
- Specify the projected structural model :
```
ael.set('structural_model','projected')
```

In the *flexibility* case the basic python commands are :
- Specify the kind of aeroelastic simulation :
```
ael.set('simulation_type','static_coupling')
```

- Specify the modal structural model :
```
ael.set('structural_model','flexibility')
```
- Specify the structural grid file :
```
ael.set('structural_grid_file','StructGrid.dat')
```
- Specify the structural matrices file :
```
ael.set('structural_matrix_file','StructMatrices.dat')
```
- Specify the initial structural variables file :
```
ael.set('structural_vars_file','StructVars.dat')
```
- Selects the relaxation coefficient value :
```
ael.set('static_relaxation',.8)
```
- Selects the starting iteration index of the coupling process :
```
ael.set('static_nbiterini',200)
```
- Selects the number of CFD iterations between two coupling steps :
```
ael.set('static_frequency',200)
```
- Defines the dynamic pressure and reference area for the output of load coefficients :
```
ael.set('dynamic_pressure_ref',15000.)
ael.set('surface_ref',180.)
```

For static coupling simulations, aeroelastic blocks must be used, with the `staticdeformable` value for the `deform` attribute.
- Defines a *staticdeformable* aeroelastic block :
```
blk0001 = aelblock(name='blk0001')
blk0001.set('deform','staticdeformable')
blk0001.set('deformation_type','aeroelastic')
```

C.2 Input and output files

The ascii Tecplot format is used in the aeroelastic module of *elsA* for input files as well as output files. Different input files are needed, depending on the kind of structural approach.

C.2.1 Structural grid file

A structural grid file is needed for fluid-structure static coupling, in the case of the approach based on a "reduced flexibility matrix". The name of this file is defined by the attribute "structural_grid_file" of the `<aeroelastics>` object.

Example of structural grid file :

```
TITLE = "Reduced Structural Grid"
VARIABLES = "x" "y" "z" "id" "familynum" "forcenode" "displnode"
ZONE T="Structural nodes" I=  51, J=1, K=1, F=POINT
  0.000000000000E+00  0.111700000000E+00 -0.210000000000E-01        1   1   0 1
  0.790900000000E-01  0.238200000000E+00 -0.116000000000E-01        3   1   0 1
  0.143200000000E+00  0.340800000000E+00 -0.390000000000E-02        5   1   0 1
       ....                ....                ....              ..   .   . .
  0.422100000000E+00  0.111700000000E+00 -0.210000000000E-01      201   2   0 1
  0.462200000000E+00  0.238200000000E+00 -0.116000000000E-01      203   2   0 1
  0.494700000000E+00  0.340800000000E+00 -0.390000000000E-02      205   2   0 1
       ....                ....                ....              ..   .   . .
  0.673000000000E+00  0.111256500000E+01  0.170000000000E-02      122   3   1 0
  0.682200000000E+00  0.112890000000E+01  0.160000000000E-02      123   3   1 0
  0.692300000000E+00  0.114690000000E+01  0.150000000000E-02      124   3   1 0
  0.696700000000E+00  0.115470000000E+01  0.140000000000E-02      125   3   1 0
```

The different columns are :

- the x, y, z coordinates of the structural node ;
- A node identification number (for example NASTRAN ID of the node) ;
- "familynum" : all structural nodes with the same "familynum" are grouped in the same "subset" of nodes ;
- "forcenode" : 0 means that this node will not be used for the transfer of loads, 1 means that it will be used for the transfer of loads ;
- "displnode" : 0 means that this node will not be used for the transfer of displacements, 1 means that it will be used for the transfer of displacements ;

C.2.2 Structural matrix file

A structural matrix file is needed for fluid-structure static coupling. The content of this file depends on the structural approach used for the aeroelastic simulation. The name of this file is defined by the attribute "structural_matrix_file" of the `<aeroelastics>` object.

"reduced flexibility matrix" approach

In this case, the file presents two distinct zones : the first zone contains the "reduced flexibility matrix", and the second zone defines which degrees of freedom (TX, TY, TZ, RX, RY, RZ) are used (1.0 : used, 0.0 : not used). Let us recall that the "reduced flexibility matrix" is built in a pre-processing step, and links the degrees of freedom (dof) of displacement (3 translations and 3 rotations per node) of the selected structural nodes to the loads applied on the subset of "force nodes" (3 components of force and 3 components of moment per "force node'). If N denotes the total number of structural nodes ("displacement" nodes and "force" nodes), M the number of "force" nodes, the "reduced flexibility matrix" is a rectangular matrix S with $6N$ lines and $6M$ columns :

$$\begin{bmatrix} Tx_1 \\ Ty_1 \\ Tz_1 \\ Rx_1 \\ Ry_1 \\ Rz_1 \\ . \\ . \\ . \\ Tx_{6N} \\ Ty_{6N} \\ Tz_{6N} \\ Rx_{6N} \\ Ry_{6N} \\ Rz_{6N} \end{bmatrix} = \begin{pmatrix} S_{1,1} & . & . & . & . & . & . & . & . & S_{1,6M} \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ S_{i,1} & . & . & . & S_{i,j} & . & . & . & . & S_{i,6M} \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ S_{6N,1} & . & . & . & . & . & . & . & . & S_{6N,M} \end{pmatrix} \begin{bmatrix} Fx_1 \\ Fy_1 \\ Fz_1 \\ Mx_1 \\ My_1 \\ Mz_1 \\ . \\ . \\ . \\ Fx_{6M} \\ Fy_{6M} \\ Fz_{6M} \\ Mx_{6M} \\ My_{6M} \\ Mz_{6M} \end{bmatrix} \tag{C.1}$$

The "reduced flexibility matrix" S must be written with the following format :

write(*,*)((S(i,j),i=1,6N),j=1,6M)

Example of structural matrix file :

```
TITLE = "Reduced Flexibility Matrix"
VARIABLES = "flexibility"
ZONE T="flexibility" I=     45900, J=1, K=1, F=BLOCK
 0.70663E-07  0.46388E-07  0.88752E-07  0.27004E-06 0.56745E-06 -0.48369E-06
 0.13718E-06  0.55941E-08  0.78033E-07  0.27004E-06 0.56745E-06 -0.48369E-06
 0.19118E-06 -0.27495E-07  0.69361E-07  0.27004E-06 0.56745E-06 -0.48369E-06
   .....        .......      .......      .........   .........     .......
-0.27963E-04  0.15015E-04 -0.47679E-04 -0.38763E-03 0.19999E-03  0.15793E-03
-0.29215E-04  0.15672E-04 -0.51582E-04 -0.38763E-03 0.19999E-03  0.15793E-03
ZONE T="flexibility" I=6, J=1, K=1, F=BLOCK
 0.00000E+00  0.00000E+00  1.00000E+00  0.00000E+00 0.00000E+00  0.00000E+00
```

Modal structural approach

In the case of a modal approach, the matrices are diagonal, and only the diagonal terms of the mass, damping and stiffness matrices are provided in the file.

Example of modal structural matrix file :

```
TITLE = "Structural matrices for modal approach"
VARIABLES = "mass" "damp" "stiff"
ZONE T= "matrices" I=2 F=POINT
1.1258576e-02  0.0000000e+00  4.7455891e+02
8.6257757e-04  0.0000000e+00  7.7247012e+01
```

Non-modal projected structural approach

In the case of a non-modal projected structural approach, the format is identical, but each term of the full matrices is provided, column by column. The number of records is therefore equal to the square of the size of the projection basis.

Example of non-modal projected structural matrix file :

```
TITLE = "Structural projected matrices"
VARIABLES = "mass" "damp" "stiff"
ZONE T= "matrices" I=4 F=POINT
0.500000  0.0010000e+00   100000.
0.050000  0.0000000e+00        0.
0.050000  0.0000000e+00        0.
0.270000  0.0100000e+00   100000.
```

C.2.3 Structural variables file

A file with initial values for the structural variables must be provided in the case of fluid-structure static coupling. The content of this file depends on the structural approach used for the aeroelastic simulation. The name of this file is defined by the attribute "structural_vars_file" of the `<aeroelastics>` object.

"reduced flexibility matrix" approach

In this case, the file presents two distinct zones : the first zone contains the 6 displacement components (TX, TY, TZ, RX, RY, RZ) for each structural node, and the second zone contains the 6 load components (FX, FY, FZ, MX, MY, MZ) for each structural node. *Example of structural variables file :*

```
TITLE = "Structural displacements and forces"
VARIABLES = "v1" "v2" "v3" "v4" "v5" "v6" "id" "familynum"
ZONE T="Displacements" I=  51, J=1, K=1, F=POINT
0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
    ...         ...         ...         ...         ...          ...
0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
ZONE T="ForceS" I=  51, J=1, K=1, F=POINT
0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
    ...         ...         ...         ...         ...          ...
0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
```

Modal or projected structural approach

In the case of a modal or projected structural approach, the structural variables are the generalised coordinates, generalised velocities, and generalised forced.

Example of structural variables file :

```
TITLE = "Generalised coordinates for modal approach"
VARIABLES = "coord" "velo" "forces"
ZONE T= "vars" I=2 F=POINT
0.0000000e+00  0.0000000e+00  0.0000000e+00
0.0000000e+00  0.0000000e+00  0.0000000e+00
```

Appendix D. DYNAMIC COUPLING SIMULATION

D.1 General description of dynamic coupling simulation

The dynamic coupling simulation gives the unsteady fluid-structure coupled time response to an initial perturbation of the statically coupled flow. The corresponding structural model may be presently *modal* or *projected*. In the *modal* case the projection basis is the modal basis which leads to diagonal mass and stiffness matrices, and also under the Basile assumption which is adopted here, to a diagonal structural damping matrix. In the *projected* case, the projection basis has no specific properties and the resolution is performed using full mass, damping and stiffness matrices. In both cases, the projection vector basis must be specified, which must be real. The initial conditions of the simulation are prescribed through the definition of the initial values for generalized coordinates and velocities. The use ot the dual time stepping for the fluid is recommended (the developpements for the Gear scheme are available but not yet completely tested). The specific output of the aeroelastic simulation consists in the time histories of the generalized coordinates, velocities and forces.

D.1.1 Prescribing a dynamic coupling simulation in the python script

The python scripts used for dynamic coupling simulations follow the general rules described before. But specific items are necessary to provide data for the proper definition of the simulation.
- Specify the kind of aeroelastic simulation :
```
ael.set('simulation_type','dynamic_coupling')
```

- Specify the modal or projected structural model :
```
ael.set('structural_model','modal')
```

- Specify the kind of the projection basis which is real :
```
ael.set('modal_basis_kind','real_modes')
```

- Specify the size of the projection basis :
```
ael.set('modal_basis_size',4)
```

- Selects the number of fluid-structure coupling inner iterations :
```
ael.set('nit_mecaloop',3)
```

- Specify the structural matrices file :
```
ael.set('structural_matrix_file','StrucMatrices.dat')
```

- Specify the initial structural variables file :
```
ael.set('structural_vars_file','StrucVars.dat')
```

D.1.2 Defining the projection vector basis on the aeroelastic interface

The real structural projection basis (modal or not) must be defined on the aeroelastic interface. This is done by defining an associated file on each local aeroelastic interface `<aelitf>` through the use of the `mode_file` attribute.

- Defines a local aeroelastic interface :
```
aelitf1 = aelitf(name='aelitf1','blk2','win38')
```

- Associates a projection vector file to the `<aelitf>` :
```
aelitf1.set('mode_file','modeai1.dat')
```

See section Input and output Files for information on the file format.

D.1.3 Activating mesh deformation for dynamic coupling

The activation of the mesh deformation feature is the same as in the forced motion case. Please refer to the corresponding section. However, the difference here is that in the *interpolated* case, the extremal mesh deformations are computed in a pre-processing step for all the members of the projection vector basis.

D.2 Input and output files

For dynamic coupling simulations, the following input files are necessary :

D.2.1 Projection vector basis file

This file describes for each basis vector the displacements on each node of a local aeroelastic interface. The format of this file is ASCII Tecplot POINT or BLOCK format. The included data is basically the same as that of the modal displacements file used for forced simulations. The aeroelastic interface nodes are described in the following order :

```
    DO K=k1,k2
      DO J=j1,j2
        DO I=i1,i2
```

```
TITLE = "Projection basis file on aeroelastic interface aelitf1"
VARIABLES = "hx" "hy" "hz"
ZONE T="mode 1" I=  51, J=12, K=1, F=POINT
  0.0000000E+00    0.0000000E+00  1.0000000E+00
  0.0000000E+00    0.0000000E+00  5.0000000E-01
  ...
ZONE T="mode 2" I=  51, J=12, K=1, F=POINT
  0.0000000E+00    2.0000000E+00  0.0000000E+00
  0.0000000E+00    3.0000000E+00  5.0000000E-02
  ...
```

D.2.2 Structural matrices file

In the *modal* case, the structural matrices files provides the values of the diagonal terms of the modal projected mass, damping, and stiffness matrices. The format of this file is ASCII Tecplot POINT or BLOCK format. The number of records is equal to the size of the modal basis.

```
TITLE = "Structural modal matrices"
VARIABLES = "mass" "damp" "stiff"
ZONE T= "matrices" I=2 F=POINT
0.500000  0.0010000e+00   100000.
0.270000  0.0100000e+00   400000.
```

In the *projected* case, the format is the same, but each term of the full matrices is provided, column by column. The number of records is therefore equal to the square of the size of the projection basis.

```
TITLE = "Structural projected matrices"
VARIABLES = "mass" "damp" "stiff"
ZONE T= "matrices" I=4 F=POINT
0.500000  0.0010000e+00   100000.
0.050000  0.0000000e+00        0.
0.050000  0.0000000e+00        0.
0.270000  0.0100000e+00   100000.
```

D.2.3 Initial Structural variables file

This file contains the initial values of the generalized coordinates, velocities and forces. It prescribes the initial condition for dynamic coupled simulation. The format of this file is ASCII Tecplot POINT or BLOCK format with a single ZONE. The number of records is equal to the size of the projection basis.

```
VARIABLES = "coord" "velo" "forces"
ZONE T= "vars" I=4 F=POINT
  0.450000E-02  0.100000E+00  0.500000E+03
 -0.125000E-03  0.000000E+00 -0.100000E+03
```

D.2.4 Mesh deformation data file

When the TFI technique is used, no additional data is needed. when the structural analogy is used, the complete description of the move3d submodule is given in a specific manual, which is not detailed here. Please refer to the corresponding document.
Automatic outputs are delivered during or at the end of dynamic coupling simulation.

D.2.5 Generalized variables history file

This file provides the time histories of the generalized coordinates, velocities and aerodynamic forces integrated on the aeroelastic interface. The format of this file is ASCII Tecplot POINT format. One file is produced for each projection basis vector. The time history of the following variables are given : iteration index, generalized coordinate, generalized velocity, generalized force, physical time. The name of the file is *gcvf_history_mode_XXX.dat* where *XXX* is the projection vector number.

```
TITLE = "Modal History file for dynamic coupling"
VARIABLES = "iter" "coord" "velo" "forces" "time"
ZONE T="Observed variables" I=  512, F=POINT
  0.1000000E+01  0.2000000E+00  0.0000000E+00 -0.5968558E-12  0.3125000E-02
  0.2000000E+01  0.1961570E+00 -0.2451577E+01 -0.2578687E+02  0.6250000E-02
  0.3000000E+01  0.1847759E+00 -0.4808941E+01 -0.2533036E+02  0.9375000E-02
  0.4000000E+01  0.1662939E+00 -0.6981501E+01 -0.1053807E+02  0.1250000E-01
  ...
```

D.2.6 Final Structural variables file

This file contains the final values of the generalized coordinates, velocities and forces. It provides the final values obtained at the end of a dynamic coupled simulation. These values may be used for a future restart.

Appendix E. LINEARIZED SIMULATION

E.1 General description of a linearized aeroelastic simulation

The linearized aeroelastic simulation solves the linearized unsteady Euler or RANS equations in frequency domain for an harmonic excitation of the aeroelastic interface. This problem is similar to the one solved in an harmonic forced motion simulation, but the formulation is in this case linearized and solved in the frequency domain. The linearized problem is solved as a fixed-point problem using a pseudo-time marching procedure allowing to use acceleration techniques such as local time step, multigrid methods and implicit algorithms. It implies the use of a modal structural model, and the specification of the structural modal basis. A specific excitation mode is selected for the simulation. The motion is prescribed at a given frequency, with a specified amplitude coefficient. The unsteady motion of the aeroelastic interface drives the aerodynamic computation through the use of an aeroelastic injection boundary condition. Local time stepping for the fluid is used. The specific output of the linearized aeroelastic simulation consists in the first harmonic of the generalized forces, and of the static pressure on the aeroelastic interface.

E.1.1 Prescribing a linearized aeroelastic simulation in the python script

The python scripts used for linearized aeroelastic simulations follow the general rules described before. It is very similar to the case of the forced harmonic motion simulation. But specific items are necessary to provide data for the proper definition of the simulation and are detailled below.
In particular, the steady flow field must be provided at the beginning of the computation.
- Specify the steady flow field file :

```
aelini1 = aeliinit(name='aelini1','windom1')
aelini1.set('steady_field_file','fac0001')
aelini1.set('format','bin_v3d')
```

A linearized simulation must be defined :
- Specify the kind of aeroelastic simulation :

```
ael.set('simulation_type','linearized')
```

- Specify the mandatory modal structural model :

```
ael.set('structural_model','modal')
```

- Specify the size of the modal basis :

```
ael.set('modal_basis_size',2)
```

- Selects the excitation mode :

```
ael.set('mode_number',1)
```

- Selects the excitation frequency :

```
ael.set('frequency',20.)
```

- Selects the excitation amplitude coefficient :

```
ael.set('modal_amplitude',1.)
```

- Defines normalizing reference values for GAF and pressure output (optional) :

```
ael.set('dynamic_pressure\_ref',Pdyn)
ael.set('surface_ref',Sref)
```

E.1.2 Defining the modal data on the aeroelastic interface

The structural modal basis must be defined on the aeroelastic interface. This is done by defining an associated mode file on each local aeroelastic interface `<aelitf>`.

- Defines a local aeroelastic interface :

```
aelitf1 = aelitf(name='aelitf1','blk2','win38')
```

- Associates a mode file to the `<aelitf>` :

```
aelitf1.set('mode_file','modeai1.dat')
```

See section Input and output Files for information on the mode file format.

E.1.3 Using the aeroelastic injection boundary condition in fixed meshes

Aeroelastic injection boundary condition prescribing the harmonic velocity of the aeroelastic interfaces nodes must be prescribed in the case of fixed meshes.
- Defines an aeroelastic injection boundary :

```
aelbnd1 = aelboundary(name='aelbnd1','blk2','win38')
aelbnd1.set('type','aelwallslip')
```

- Link boundary to the associated local aeroelastic interface :

```
aelbnd1.set('ael_interface','aelitf1')
```

E.1.4 Activating mesh deformation for aeroelasticity

The activation of the mesh deformation feature is the same as in the forced motion case. Please refer to the corresponding section. However, no actual mesh deformation is performed during the computation, but additional flux terms involving mesh deformation velocities are taken into account in this case.
`<aelblock>` objects must be used, with `deform` = *`staticdeformable`*, and `deformation_type` = *`aeroelastic_lin`* :

```
Blk1 = aelblock(name='Blk1')
Blk1.set('deform','staticdeformable')
Blk1.set('deformation_type','aeroelastic_lin')
```

E.2 Input and output files

For linearized simulations, the following input files are necessary :

E.2.1 Modal displacements file

This file describes for each mode the displacements on each node of a local aeroelastic interface. The format of this file is ASCII Tecplot POINT or BLOCK format. The aeroelastic interface nodes are described in the following order :

```
     DO K=k1,k2
       DO J=j1,j2
         DO I=i1,i2
```

```
TITLE = "Modal Displacements file on aeroelastic interface aelitf1"
VARIABLES = "hx" "hy" "hz"
ZONE T="mode 1" I=  51, J=12, K=1, F=POINT
  0.0000000E+00    0.0000000E+00  1.0000000E+00
  0.0000000E+00    0.0000000E+00  5.0000000E-01
  ...
ZONE T="mode 2" I=  51, J=12, K=1, F=POINT
  0.0000000E+00    2.0000000E+00  0.0000000E+00
  0.0000000E+00    3.0000000E+00  5.0000000E-02
  ...
```

E.2.2 Mesh deformation data file

When the TFI technique is used, no additional data is needed. In the case the structural analogy is used, the complete move3d submodule description is given in a specific manual, which is not detailled here. Please refer to the corresponding document.
Several automatic outputs are delivered during or at the end of the forced motion simulation.

E.2.3 First harmonic of the generalized aerodynamic forces file

This file provides the first harmonic analysis of the generalized aerodynamic forces integrated on the aeroelastic interface. The format of this file is ASCII Tecplot POINT format. A single file is produced during the simulation which gives one row of the GAF matrix at the end of the simulation. The name of the file is *gaf_XXX.dat*. The output values may be normalized using the *`dynamic_pressure_ref`* and *`surface_ref`* attributes of the `aeroelastics` object.

```
TITLE = "Forces Harmonic Analysis File for forced motion on mode 1"
VARIABLES = "Imod" "Jmod" "ReFiOnHj" "ImFiOnHj"
ZONE T="Window(s) : Observed generalized forces",  I=2, F=POINT
  0.1000000E+01  0.1000000E+01  0.2729172E+04 -0.1207292E+04
  0.1000000E+01  0.2000000E+01 -0.1982318E+04  0.5697731E+03
```

E.2.4 First harmonic of static pressure file

This file provides the first harmonic analysis of the static pressure on each node of a local aeroelastic interface. The format of this file is ASCII Tecplot POINT format. One file is produced for each local aeroelastic interface. This file gives the real and imaginary parts of static pressure at the end of the simulation. The name of the file is *harmonic_pressure_mode_XXX_aiZZZ.dat* where *aiZZZ* is the name of the corresponding aeroelastic interface. The output values may be normalized using the *`dynamic_pressure_ref`* attribute of the `aeroelastics` object.

```
TITLE = "Pressure Harmonic Analysis File on aelitf1"
VARIABLES = "x" "y" "z" "pr" "pi"
ZONE T="Ps harmonic analysis on aelitf1 cycle 1 ",  I=51,  J=12, F=POINT
  0.10814E+00  0.20290E-01  0.34284E+00 -0.32821E+05  0.23898E+04
  0.10816E+00  0.20239E-01  0.34285E+00 -0.32521E+05  0.23850E+04
  0.10819E+00  0.20193E-01  0.34286E+00 -0.32304E+05  0.24105E+04
  ...
```

E.2.5 Generalized aerodynamic forces convergence file

This file gives the history of the first harmonic of the generalized aerodynamic forces.

Appendix F. EXAMPLE SCRIPTS

F.1 Python–*elsA* script examples for Aeroelasticity

F.1.1 Fluid-structure static coupling using a "reduced flexibility matrix"

This is an example of static coupling simulation in the case of a wing alone configuration, using a reduced flexibility matrix for the computation of the structural displacements.

```
from elsA_user import *
from elsA_Ael_user import *

#.. Declare an aelpb instead of a cfdpb
problem = aelpb(name='problem')

#.. Infinite state
stateInf = state(name='stateInf')
...

#.. Meshes
msh0001 = mesh(name='msh0001')
msh0001.set('file','fgv0001.dat')
...

#.. Blocks
blk0001 = aelblock(name='blk0001')
blk0001.set('mesh','msh0001')
blk0001.set('deform','staticdeformable')
blk0001.set('deformation_type','aeroelastic')
blk0001.set('freq_comp_vol',1)
blk0001.set('local_model','mod0001')
blk0001.set('local_numerics','num0001')
blk0001.set('proc',0)
...

#.. Model
mod0001 = model(name='mod0001')
mod0001.set('fluid','pg')
mod0001.set('gamma',Gamma)
...

#.. Numerics
num0001 = numerics(name='num0001')
num0001.set('time_algo','steady')
num0001.set('ode','rk4')
num0001.set('implicit','irs')
num0001.set('flux','jameson')
num0001.set('artviscosity','dissca')
num0001.set('av_base',[CHI2, CHI4, PVSIGMA])
...

#.. Aeroelastic Problem definition
ael=aeroelastics(name='ael')
ael.set('mesh_deformation_type','computed')
ael.set('mesh_deformation_technique','tfi')
```

```
ael.set('fluid_density_unit',RoadimF)
ael.set('fluid_length_unit',LadimF)
ael.set('fluid_velocity_unit',QadimF)
ael.set('fluid_temperature_unit',TadimF)
ael.set('structure_mass_unit',1.)
ael.set('structure_length_unit',1.)
ael.set('structure_time_unit',1.)
ael.set('structure_temperature_unit',1.)
ael.set('simulation_type','static_coupling')
ael.set('static_frequency',2)
ael.set('static_relaxation',0.8)
ael.set('static_nbiterini',1)
ael.set('structural_model','flexibility')
ael.set('association_type','min_distance')
ael.set('dynamic_pressure_ref',28031.43)
ael.set('surface_ref',0.289)
ael.set('structural_grid_file','structuralGrid')
ael.set('structural_matrix_file','structuralMatrix')
ael.set('structural_vars_file','structuralVars')
ael.set('format','fmt_tp')
ael.show()

#.. Boundaries
w0005 = window('blk0001',name='w0005')
w0005.set('wnd',[  33, 105,   1,  33,   1,   1])
b0005 = boundary('blk0001','w0005','undefined',name='b0005')
b0005.set('type','wallslip')
b0005.set('ael_interface','ai5')

#.. Aeroelastic interface for window w0005
ai5=aelitf('blk0001','w0005',name='ai5')
ai5.set('force_group','WING')
ai5.set('displ_group','WING')

#.. Structural Parts
stru1 = strupart(name='stru1')
stru1.set('part',['WING-LE','WING','WING',1,25])
stru2 = strupart(name='stru2')
stru2.set('part',['WING-TE','WING','WING',201,225])
stru3 = strupart(name='stru3')
stru3.set('part',['WING-FORCE','WING','WING',101,125])

#.. Transfer methods for displacements
#displtrf1 = displtransfer(name='displtrf11')
#displtrf1.set('method',['infinite_plate','WING',[1.,0.,0.],[0.,1.,0.]])
#displtrf2 = displtransfer(name='displtrf2')
#displtrf2.set('method',['infinite_volume','WING'])
displtrf3 = displtransfer(name='displtrf3')
displtrf3.set('method',['polynomial_2d','WING',[1.,0.,0.],[0.,1.,0.],2,3])

#.. Connecting numerics, model, and aeroelastics with problem
problem.set('global_numerics','num0001')
problem.set('global_model','mod0001')
problem.set('global_aeroelastics','ael')
problem.compute()
problem.extract()
```

F.1.2 Fluid-structure static coupling using a modal approach

This is an example of static coupling simulation in the case of a 2D wing profile configuration, using a modal approach for the computation of the structural displacements.

```
from elsA_user import *
from elsA_Ael_user import *
from math import *
import time

#.. Declare an aelpb instead of a cfdpb
nlr7301 = aelpb(name='nlr7301')
nlr7301.set_ghostcell(2,2,2,2,0,1)
nlr7301.set('config','2d')
...

#.. Aeroelastic Problem definition
ael=aeroelastics(name='ael')
ael.set('mesh_deformation_technique','move3d')
ael.set('mesh_deformation_type','interpolated')
ael.set('fluid_density_unit',RoadimF)
ael.set('fluid_length_unit',LadimF)
ael.set('fluid_velocity_unit',QadimF)
ael.set('fluid_temperature_unit',TadimF)
ael.set('structure_mass_unit',1.)
ael.set('structure_length_unit',1.)
ael.set('structure_time_unit',1.)
ael.set('structure_temperature_unit',1.)
ael.set('structural_model','modal')
ael.set('modal_basis_kind','real_modes')
ael.set('modal_basis_size',2)
ael.set('modal_data','on_ael_interface')
ael.set('simulation_type','static_coupling')
ael.set('static_frequency',2)
ael.set('static_relaxation',1.0)
ael.set('static_nbiterini',1)
ael.set('structural_matrix_file','./structMatrices')
ael.set('structural_vars_file','./structVars')
ael.set('format','fmt_tp')
ael.set('mesh_deformation_data_file','./mmove.d')
ael.set('mesh_deformation_output_file','./mmove.out')
#ael.show()

#.. Model
Mod0001 = model(name='Mod0001')
Mod0001.set('cv',Cv)
Mod0001.set('fluid','pg')
Mod0001.set('gamma',Gamma)
Mod0001.set('phymod','nstur')
Mod0001.set('prandtl',0.72)
Mod0001.set('prandtltb',0.9)
Mod0001.set('turbmod',TURBMOD)
...
```

```
#.. Numerics
Num0001 = numerics(name='Num0001')
#
if TIMESTEPPING == 'RK4IRS' :
  Num0001.set('time_stepping', ['rk4', 'steady', 1.E-06])
  Num0001.set('implicit','irs')
  Num0001.set('irs_params', [-0.15, -0.15, -0.15])
  Num0001.set('cfl',4.0)
elif TIMESTEPPING == 'BWLU' :
  Num0001.set('time_stepping', ['backwardeuler', 'steady', 1.E-06])
  Num0001.set('implicit','lussorsca')
  Num0001.set('ssorcycle',4)
  Num0001.set('cfl',25.0)

Num0001.set('freqcomptimestep',1)
Num0001.set('freezing',1)
...

#.. Infinite state
StaInfNref = state(name='StaInfNref')
StaInfNref.set('var','conservative')
StaInfNref.set('conservative',[ RoInf, RouInf, RovInf, RowInf, RoeInf ])
StaInfNref.set('v6',RoTur1)
StaInfNref.set('v7',RoTur2)
...

#.. Meshes
R=sequence("Msh%4d = mesh(name='Msh%4d')", 1,  1)
R=sequence("Msh%4d.set('file','./xyz%4d.dat')", 1,  1)
R=sequence("Msh%4d.set('format','fmt_tp')", 1,  1)
R=sequence("Msh%4d.set('proc',0)", indices=[1])
Msh0001.set('dim', [  337,   65,   2 ] )
...

#.. Blocks
R=sequence("Blk%4d = aelblock(name='Blk%4d')", 1,  1)
R=sequence("Blk%4d.attach(Msh%4d)", 1,  1)
R=sequence("Blk%4d.set('deform','staticdeformable')", 1,  1)
R=sequence("Blk%4d.set('deformation_type','aeroelastic')", 1,  1)
R=sequence("Blk%4d.set('freq_comp_vol',1)", 1,  1)
R=sequence("Blk%4d.set('proc',0)", indices=[1])
...

R=sequence("Dom%4d = window('Blk%4d',name='Dom%4d')", 1,  1)
Dom0001.set('wnd',[   1,  337,    1,   65,    1,   2])

#.. Init Flow
R=sequence("Ini%4d = init('Dom%4d',name='Ini%4d')", 1,  1)
R=sequence("Ini%4d.set('window_name','Dom%4d')", 1,  1)
if ITER1 == 1:
   R=sequence("Ini%4d.set('state','StaInfNref')", 1,  1)
   R=sequence("Ini%4d.set('coeffmutinit',COEFFMUTINIT)", 1,  1)
else :
   R=sequence("Ini%4d.set('file','fac%4d.v3d')", 1,  1)
   R=sequence("Ini%4d.set('format','bin_v3d')", 1,  1)
...
```

```
#.. Init Wall Distance
if Dst_READ == 'Y' :
   R=sequence("Iwd%4d = init('Dom%4d',name='Iwd%4d')", 1,  1)
   R=sequence("Iwd%4d.set('window_name','Dom%4d')", 1,  1)
   R=sequence("Iwd%4d.set('file','E_Dst%4d_nlr7301.ini')", 1,  1)
   R=sequence("Iwd%4d.set('format','bin_v3d')", 1,  1)
   R=sequence("Iwd%4d.set('var',Dst_VarS)", 1,  1)
...

#.. BLOCK 0001
R=sequence("Fen%4d = window('Blk0001',name='Fen%4d')",   1,  7)
Fen0001.set('wnd',[  49,  169,    1,    1,    1,    2])
Fen0002.set('wnd',[ 169,  289,    1,    1,    1,    2])
Fen0003.set('wnd',[   1,  337,   65,   65,    1,    2])
Fen0004.set('wnd',[   1,    1,    1,   65,    1,    2])
Fen0005.set('wnd',[ 337,  337,    1,   65,    1,    2])
Fen0006.set('wnd',[ 289,  337,    1,    1,    1,    2])
Fen0007.set('wnd',[   1,   49,    1,    1,    1,    2])
...

R=sequence("Bnd%4d = boundary('Blk0001','Fen%4d',name='Bnd%4d')",   1,  7)

Bnd0001.set('type','walladia')
ai0001 = aelitf('Blk0001','Fen0001',name='ai0001')
ai0001.setDict({'mode_file':'./lowerModes.dat','format':'fmt_tp'})

Bnd0002.set('type','walladia')
ai0002 = aelitf('Blk0001','Fen0002',name='ai0002')
ai0002.setDict({'mode_file':'./upperModes.dat','format':'fmt_tp'})

Bnd0003.setDict({'type':'nref', 'state':'StaInfNref'})
Bnd0004.setDict({'type':'nref', 'state':'StaInfNref'})
Bnd0005.setDict({'type':'nref', 'state':'StaInfNref'})
Bnd0006.setDict({'type':'join','jtype':'match','links':['Blk0001','Fen0007']})
Bnd0007.setDict({'type':'join','jtype':'match','links':['Blk0001','Fen0006']})

#.. Connecting numerics, model, and aeroelastics with problem
nlr7301.set('global_numerics','Num0001')
nlr7301.set('global_model','Mod0001')
nlr7301.set('global_aeroelastics','ael')

nlr7301.submit()
nlr7301.compute()
nlr7301.extract()
```

F.1.3 Fluid-structure dynamic coupling using a modal approach

This is an example of dynamic coupling simulation in the case of a 2D wing profile configuration, using a modal approach for the computation of the structural displacements and velocities.

```
from elsA_user import *
from elsA_Ael_user import *
from math import *
import time
```

```
#.. Declare an aelpb instead of a cfdpb
nlr7301 = aelpb(name='nlr7301')
nlr7301.set_ghostcell(2,2,2,2,0,1)
nlr7301.set('config','2d')
...

#.. Aeroelastic Problem definition
ael=aeroelastics(name='ael')
ael.set('mesh_deformation_technique','move3d')
ael.set('mesh_deformation_type','interpolated')
ael.set('fluid_density_unit',RoadimF)
ael.set('fluid_length_unit',LadimF)
ael.set('fluid_velocity_unit',QadimF)
ael.set('fluid_temperature_unit',TadimF)
ael.set('structure_mass_unit',1.)
ael.set('structure_length_unit',1.)
ael.set('structure_time_unit',1.)
ael.set('structure_temperature_unit',1.)
ael.set('simulation_type','dynamic_coupling')
ael.set('nit_mecaloop',3)
ael.set('structural_model','modal')
ael.set('modal_basis_kind','real_modes')
ael.set('modal_basis_size',2)
ael.set('modal_data','on_ael_interface')
ael.set('structural_matrix_file','./structMatrices')
ael.set('structural_vars_file','./structVars')
ael.set('format','fmt_tp')
ael.set('mesh_deformation_data_file','./mmove.d')
ael.set('mesh_deformation_output_file','./mmove.d.out')

#.. Model
Mod0001 = model(name='Mod0001')
Mod0001.set('cv',Cv)
Mod0001.set('fluid','pg')
Mod0001.set('gamma',Gamma)
Mod0001.set('phymod','nstur')
Mod0001.set('prandtl',0.72)
Mod0001.set('prandtltb',0.9)
Mod0001.set('turbmod',TURBMOD)

#.. Numerics
Num0001 = numerics(name='Num0001')

if TIMESTEPPING == 'RK4IRS' :
  Num0001.set('time_stepping', ['rk4', 'steady', 1.E-06])
  Num0001.set('implicit','irs')
  Num0001.set('irs_params', [-0.15, -0.15, -0.15])
  Num0001.set('cfl',4.0)
elif TIMESTEPPING == 'BWLU' :
  Num0001.set('time_stepping', ['backwardeuler', 'steady', 1.E-06])
  Num0001.set('implicit','lussorsca')
  Num0001.set('ssorcycle',4)
  Num0001.set('cfl',25.0)
elif TIMESTEPPING == 'DTSBWLU' :
  Num0001.set('time_stepping', ['backwardeuler', 'dts', 1.E-06])
  Num0001.set('implicit','lussorsca')
```

```
  Num0001.set('ssorcycle',4)
  Num0001.set('cfl',25.0)
  Num0001.set('dual_iteration',5)
  Num0001.set('restoreach_cons',0.05)
  Num0001.set('timestep',AelDt)
  Num0001.set('itime',0.)
...

#.. Infinite state
StaInfNref = state(name='StaInfNref')
StaInfNref.set('var','conservative')
StaInfNref.set('conservative',[ RoInf, RouInf, RovInf, RowInf, RoeInf ])
StaInfNref.set('v6',RoTur1)
StaInfNref.set('v7',RoTur2)
...

#.. Meshes
R=sequence("Msh%4d = mesh(name='Msh%4d')", 1,  1)
R=sequence("Msh%4d.set('file','./xyz%4d.dat')", 1,  1)
R=sequence("Msh%4d.set('format','fmt_tp')", 1,  1)
R=sequence("Msh%4d.set('proc',0)", indices=[1])
Msh0001.set('dim', [  337,   65,   2 ] )
R=sequence("Msh%4d.submit()", 1,  1)
...

#.. Blocks
R=sequence("Blk%4d = aelblock(name='Blk%4d')", 1,  1)
R=sequence("Blk%4d.attach(Msh%4d)", 1,  1)
R=sequence("Blk%4d.set('deform','deformable')", 1,  1)
R=sequence("Blk%4d.set('deformation_type','aeroelastic')", 1,  1)
R=sequence("Blk%4d.set('freq_comp_vol',1)", 1,  1)
R=sequence("Blk%4d.set('proc',0)", indices=[1])
R=sequence("Blk%4d.submit()", 1,  1)
...

R=sequence("Dom%4d = window('Blk%4d',name='Dom%4d')", 1,  1)
Dom0001.set('wnd',[   1,  337,    1,   65,    1,   2])

R=sequence("Ini%4d = init('Dom%4d',name='Ini%4d')", 1,  1)
R=sequence("Ini%4d.set('window_name','Dom%4d')", 1,  1)
if ITER1 == 1:
   R=sequence("Ini%4d.set('state','StaInfNref')", 1,  1)
   R=sequence("Ini%4d.set('coeffmutinit',COEFFMUTINIT)", 1,  1)
else :
   R=sequence("Ini%4d.set('file','fac%4d.v3d')", 1,  1)
   R=sequence("Ini%4d.set('format','bin_v3d')", 1,  1)
if Ini_SHOW == 'Y' :
   R=sequence("Ini%4d.show()", 1,  1)

if Dst_READ == 'Y' :
   R=sequence("Iwd%4d = init('Dom%4d',name='Iwd%4d')", 1,  1)
   R=sequence("Iwd%4d.set('window_name','Dom%4d')", 1,  1)
   R=sequence("Iwd%4d.set('file','E_Dst%4d_nlr7301.ini')", 1,  1)
   R=sequence("Iwd%4d.set('format','bin_v3d')", 1,  1)
   R=sequence("Iwd%4d.set('var',Dst_VarS)", 1,  1)
```

```
R=sequence("Fen%4d = window('Blk0001',name='Fen%4d')",   1,  7)
Fen0001.set('wnd',[  49,  169,    1,    1,    1,    2])
Fen0002.set('wnd',[ 169,  289,    1,    1,    1,    2])
Fen0003.set('wnd',[   1,  337,   65,   65,    1,    2])
Fen0004.set('wnd',[   1,    1,    1,   65,    1,    2])
Fen0005.set('wnd',[ 337,  337,    1,   65,    1,    2])
Fen0006.set('wnd',[ 289,  337,    1,    1,    1,    2])
Fen0007.set('wnd',[   1,   49,    1,    1,    1,    2])

R=sequence("Bnd%4d = boundary('Blk0001','Fen%4d',name='Bnd%4d')",   1,  7)

Bnd0001.set('type','walladia')
Bnd0001.set('mobile_coef',-1.)
ai0001 = aelitf('Blk0001','Fen0001',name='ai0001')
ai0001.setDict({'mode_file':'./lowerModes.dat','format':'fmt_tp'})

Bnd0002.set('type','walladia')
Bnd0002.set('mobile_coef',-1.)
ai0002 = aelitf('Blk0001','Fen0002',name='ai0002')
ai0002.setDict({'mode_file':'./upperModes.dat','format':'fmt_tp'})

Bnd0003.setDict({'type':'nref', 'state':'StaInfNref'})
Bnd0004.setDict({'type':'nref', 'state':'StaInfNref'})
Bnd0005.setDict({'type':'nref', 'state':'StaInfNref'})
Bnd0006.setDict({'type':'join','jtype':'match','links':['Blk0001','Fen0007']})
Bnd0007.setDict({'type':'join','jtype':'match','links':['Blk0001','Fen0006']})
...

nlr7301.set('global_numerics','Num0001')
nlr7301.set('global_model','Mod0001')
nlr7301.set('global_aeroelastics','ael')

nlr7301.submit()
nlr7301.compute()
nlr7301.extract()
```

F.1.4 Forced motion simulation for aircraft application

This is an example of forced motion simulation in the case of a wing configuration.

```
from elsA_user import *
from elsA_Ael_user import *
import time
from math import *

#.. Declare an aelpb instead of a cfdpb
problem = aelpb(name='problem')

2D problem ?
if TWODPB == 'yes':
  problem.set_ghostcell(2,2,2,2,0,1)
  problem.set('config','2d')

#.. Aeroelastic Problem definition
ael=aeroelastics(name='ael')
ael.set('mesh_deformation_technique','move3d')
```

```
ael.set('mesh_deformation_type','interpolated')
ael.set('fluid_density_unit',RoadimF)
ael.set('fluid_length_unit',LadimF)
ael.set('fluid_velocity_unit',QadimF)
ael.set('fluid_temperature_unit',TadimF)
ael.set('structure_mass_unit',1.)
ael.set('structure_length_unit',1.)
ael.set('structure_time_unit',1.)
ael.set('structure_temperature_unit',1.)
ael.set('simulation_type','forced_motion')
ael.set('forced_motion_type','harmonic')
ael.set('structural_model','modal')
ael.set('modal_basis_kind','complex_modes')
ael.set('modal_basis_size',1)
ael.set('modal_data','on_ael_interface')
ael.set('frequency',AELFREQUENCY)
ael.set('mode_number',1)
ael.set('modal_amplitude',0.005)
ael.set('initial_phase',4.712388980)
ael.set('format','fmt_tp')
ael.set('mesh_deformation_data_file','./mmove.d')
ael.set('mesh_deformation_output_file','./mmove.d.out')

#.. Model
mod0001 = model(name='mod0001')
mod0001.set('fluid','pg')
mod0001.set('gamma',Gamma)
mod0001.set('cv',Cv)
...

#.. Numerics
num0001 = numerics(name='num0001')
num0001.set('time_algo','dts')
num0001.set('dual_iteration',50)
num0001.set('restoreach_cons',0.05)
num0001.set('timestep',AelDt)
num0001.set('itime',0.)
...

#.. Infinite state
stateInf = state(name='stateInf')
stateInf.set('ro',RoInf)
stateInf.set('rou',RouInf)
stateInf.set('rov',RovInf)
stateInf.set('row',RowInf)
stateInf.set('roe',RoeInf)
stateInf.set('v6',RoTur1)
stateInf.set('v7',RoTur2)

#.. Meshes
R=sequence("msh%4d = mesh(name='msh%4d')",1,   1)
R=sequence("msh%4d.set('file','../Static/fgv%4d')",1,   1)
R=sequence("msh%4d.set('format','bin_v3d')",1,   1)
R=sequence("msh%4d.set('proc', 0)",indices=[1])
...
```

```
#.. Blocks
R=sequence("blk%4d = aelblock(name='blk%4d')",1,   1)
R=sequence("blk%4d.set('mesh','msh%4d')",1,   1)
R=sequence("blk%4d.set('deform','deformable')", 1,  1)
R=sequence("blk%4d.set('deformation_type','aeroelastic')", 1,  1)
R=sequence("blk%4d.set('freq_comp_vol',1)", 1,  1)
R=sequence("blk%4d.set('proc', 0)",indices=[1])
...

#.. Init Flow
R=sequence("ini%4d = init('windom%4d',name='ini%4d')",1,Nbloc)
R=sequence("ini%4d.set('window_name','windom%4d')",1,Nbloc)
if RESTART == 'norestart':
    R=sequence("ini%4d.set('state','stateInf')",1,Nbloc)
    R=sequence("ini%4d.set('coeffmutinit',COEFFMUTINIT)",1,Nbloc)
else :
    R=sequence("ini%4d.set('file','../Static/fac%4d')",1,Nbloc)
    R=sequence("ini%4d.set('format','bin_v3d')",1,Nbloc)
...

#.. Boundary conditions
w0001 = window('blk0001',name='w0001')
w0001.set('wnd',[   1,   1,   1,  81,   1,  61])
b0001 = boundary('blk0001','w0001',name='b0001')
b0001.set('type','wallslip')
b0001.set('extrap',0)
#
w0002 = window('blk0001',name='w0002')
w0002.set('wnd',[ 185, 185,   1,  81,   1,  61])
b0002 = boundary('blk0001','w0002',name='b0002')
b0002.set('type','wallslip')
b0002.set('extrap',0)

w0003 = window('blk0001',name='w0003')
w0003.set('wnd',[   1, 185,   1,   1,   1,  61])
b0003 = boundary('blk0001','w0003',name='b0003')
b0003.set('type','walladia')
b0003.set('mobile_coef',0.)

w0004 = window('blk0001',name='w0004')
w0004.set('wnd',[   1, 185,  81,  81,   1,  61])
b0004 = boundary('blk0001','w0004',name='b0004')
b0004.set('type','nref')
b0004.set('nref_type','unst')
b0004.set('state','stateInf')

w0005 = window('blk0001',name='w0005')
w0005.set('wnd',[  21,  93,   1,  61,   1,   1])
b0005 = boundary('blk0001','w0005',name='b0005')
b0005.set('type','walladia')
b0005.set('mobile_coef',0.)
ali0005=aelitf('blk0001','w0005',name='ali0005')
ali0005.setDict({'mode_file':'../LissageMotion/motionCplx0005.dat','format':'fmt_tp'})

w0006 = window('blk0001',name='w0006')
w0006.set('wnd',[  93, 165,   1,  61,   1,   1])
```

```
b0006 = boundary('blk0001','w0006',name='b0006')
b0006.set('type','walladia')
b0006.set('mobile_coef',0.)
ali0006=aelitf('blk0001','w0006',name='ali0006')
ali0006.setDict({'mode_file':'../LissageMotion/motionCplx0006.dat','format':'fmt_tp'})

w0007 = window('blk0001',name='w0007')
w0007.set('wnd',[   1,  21,   1,  61,   1,   1])
b0007 = boundary('blk0001','w0007',name='b0007')
b0007.set('type','join')
b0007.set('trirac',[  -1,   2,  -3])
b0007.set('jtype','match')
b0007.set('blkrac','blk0001')
b0007.set('wndrac','w0008')

w0008 = window('blk0001',name='w0008')
w0008.set('wnd',[ 165, 185,   1,  61,   1,   1])
b0008 = boundary('blk0001','w0008',name='b0008')
b0008.set('type','join')
b0008.set('trirac',[  -1,   2,  -3])
b0008.set('jtype','match')
b0008.set('blkrac','blk0001')
b0008.set('wndrac','w0007')

w0009 = window('blk0001',name='w0009')
w0009.set('wnd',[   1,  93,  61,  81,   1,   1])
b0009 = boundary('blk0001','w0009',name='b0009')
b0009.set('type','join')
b0009.set('trirac',[  -1,   2,  -3])
b0009.set('jtype','match')
b0009.set('blkrac','blk0001')
b0009.set('wndrac','w0010')

w0010 = window('blk0001',name='w0010')
w0010.set('wnd',[  93, 185,  61,  81,   1,   1])
b0010 = boundary('blk0001','w0010',name='b0010')
b0010.set('type','join')
b0010.set('trirac',[  -1,   2,  -3])
b0010.set('jtype','match')
b0010.set('blkrac','blk0001')
b0010.set('wndrac','w0009')

w0011 = window('blk0001',name='w0011')
w0011.set('wnd',[   1, 185,   1,  81,  61,  61])
b0011 = boundary('blk0001','w0011',name='b0011')
b0011.set('type','wallslip')
b0011.set('extrap',0)
...

#.. Connecting numerics, model, and aeroelastics with problem
problem.set('global_numerics','num0001')
problem.set('global_model','mod0001')
problem.set('global_aeroelastics','ael')
problem.submit()
problem.compute()
problem.extract()
```

F.1.5 Forced motion simulation for turbomachinary application

This is an example of forced motion simulation in the case of a compressor fan. The computation is restricted to one interblade channel using a chorochronic boundary condition.

```
from elsA import *
from elsA_Ael import *
import time

problem = DesAelPb('problem')
problem.set_binv3d('i8','r8')
problem.set_nb_error_max(30)

#.. 2D problem ?
if TWODPB == 'yes':
  problem.set_ghostcell(2,2,2,2,0,1)
  problem.setI('config',1)

#.. Infinite state
stateInf = DesState('stateInf')
stateInf.setF('ro',RoInf)
stateInf.setF('rou',RouInf)
stateInf.setF('rov',RovInf)
stateInf.setF('row',RowInf)
stateInf.setF('roe',RoeInf)
stateInf.setF('v6',RoTur1)
stateInf.setF('v7',RoTur2)
...

#.. Downstream state
stateAv = DesState('stateAv')
stateAv.setF('ro',RoAv)
stateAv.setF('rou',RouAv)
stateAv.setF('rov',RovAv)
stateAv.setF('row',RowAv)
stateAv.setF('roe',RoeAv)
stateAv.setF('v6',RoTur1)
stateAv.setF('v7',RoTur2)
...

#.. Meshes
msh0001 = DesMesh('msh0001')
msh0001.setS('file','sgc2-4d.0001.mv3d')
msh0001.setS('format','bin_v3d')
msh0001.setI('proc',0)

#.. Blocks
blk0001 = DesAelBlock('blk0001')
blk0001.setS('mesh','msh0001')

#.. Rotating bloc
if ROTATION == 'yes':
  blk0001.setS('config','axi')
  blk0001.setF('axis_pnt_x',0.0)
  blk0001.setF('axis_pnt_y',0.0)
  blk0001.setF('axis_pnt_z',0.0)
  blk0001.setF('axis_vct_x',1.0)
```

```
  blk0001.setF('axis_vct_y',0.0)
  blk0001.setF('axis_vct_z',0.0)
  blk0001.setI('axis_ang_1',NSECTEURS)
  blk0001.setI('axis_ang_2',1)
  blk0001.setI('motion',1)
  blk0001.setF('transl_speed_x',0.0)
  blk0001.setF('transl_speed_y',0.0)
  blk0001.setF('transl_speed_z',0.0)
  blk0001.setF('omega',omega)

#.. mesh deformation
blk0001.setI('deform',1)
blk0001.setS('deformation_type','aeroelastic')
blk0001.setI('freq_comp_vol',1)
blk0001.setI('vol_comp_meth',0)
blk0001.setI('proc',0)
...

#.. Field initialization
windom0001 = DesWindow('windom0001','blk0001')
windom0001.setI('iw1',   1)
windom0001.setI('iw2',  41)
windom0001.setI('jw1',   1)
windom0001.setI('jw2',  41)
windom0001.setI('kw1',   1)
windom0001.setI('kw2',  53)
ini0001 = DesInit('ini0001','windom0001')
if RESTART == 'norestart':
  ini0001.setS('state','stateInf')
  ini0001.setF('coeffmutinit',COEFFMUTINIT)
else:
  LOCRESTART=RESTART+'0001.av3d'
  ini0001.setS('file',LOCRESTART)
  ini0001.setS('format','bin_v3d')

#.. Create model blk0001
mod0001 = DesModel('mod0001')
mod0001.setS('fluid','pg')
mod0001.setF('gamma',Gamma)
mod0001.setF('cv',Cv)
...

#.. Create numerics blk0001
num0001 = DesNumerics('num0001')
...
if ROTATION == 'yes':
  num0001.setI('vel_formulation',1)

#.. Chorochronic boundary condition
#   "rot" parameter=+1 :  positive X-axis rotation
#   when moving from boundary 2 to boundary 1
w0004 = DesWindow('w0004','blk0001')
w0004.setI('iw1',   1)
w0004.setI('iw2',  41)
w0004.setI('jw1',  41)
w0004.setI('jw2',  41)
```

```
w0004.setI('kw1',   1)
w0004.setI('kw2',  53)
b0004 = DesBoundary('b0004','blk0001','w0004','undefined')
b0004.setS('type','chorochrono')
b0004.setS('choro_type','aeroelastic')
b0004.setS('border','delay')
b0004.setF('time_period',TIMEPERIOD)
b0004.setI('dephasing',NDIAM)
b0004.setI('harm_num',NBHARM)
b0004.setI('harm_freq_comp',FREQCOMPHARM)
b0004.setS('format','fmt_tp')
b0004.setS('choro_file','choro_b0004.dat')
b0004.setS('jtopo','periodic')
b0004.setS('ptype','rot')
b0004.setI('pangle',-1)
b0004.setF('join_tolerance',1.e-04)
b0004.setS('blkrac','blk0001')
b0004.setS('wndrac','w0005')
b0004.setI('t1rac',1)
b0004.setI('t2rac',2)
b0004.setI('t3rac',3)

#.. Chorochronic boundary condition
#   "rot" parameter=+1 :  positive X-axis rotation
#   when moving from boundary 2 to boundary 1
w0005 = DesWindow('w0005','blk0001')
w0005.setI('iw1',   1)
w0005.setI('iw2',  41)
w0005.setI('jw1',   1)
w0005.setI('jw2',   1)
w0005.setI('kw1',   1)
w0005.setI('kw2',  53)
b0005 = DesBoundary('b0005','blk0001','w0005','undefined')
b0005.setS('type','chorochrono')
b0005.setS('choro_type','aeroelastic')
b0005.setS('border','advance')
b0005.setF('time_period',TIMEPERIOD)
b0005.setI('dephasing',NDIAM)
b0005.setI('harm_num',NBHARM)
b0005.setI('harm_freq_comp',FREQCOMPHARM)
b0005.setS('format','fmt_tp')
b0005.setS('choro_file','choro_b0005.dat')
b0005.setS('jtopo','periodic')
b0005.setS('ptype','rot')
b0005.setI('pangle',+1)
b0005.setF('join_tolerance',1.e-04)
b0005.setS('blkrac','blk0001')
b0005.setS('wndrac','w0004')
b0005.setI('t1rac',1)
b0005.setI('t2rac',2)
b0005.setI('t3rac',3)

w0006 = DesWindow('w0006','blk0001')
w0006.setI('iw1',   1)
w0006.setI('iw2',  41)
w0006.setI('jw1',   1)
```

```
w0006.setI('jw2',  41)
w0006.setI('kw1',   1)
w0006.setI('kw2',   1)
b0006 = DesBoundary('b0006','blk0001','w0006','undefined')
b0006.setS('type','walladia')
#.. Fixed wall (stator) in relative velocity formulation
#b0006.setF('mobile_coef',+1.)
#.. Moving wall (rotor) in relative velocity formulation
b0006.setF('mobile_coef',0.)


#.. Aeroelastic window w0014 (Upper surface)
wa0014 = DesWindow('wa0014','blk0002')
wa0014.setI('iw1',   1)
wa0014.setI('iw2', 111)
wa0014.setI('jw1',   1)
wa0014.setI('jw2',   1)
wa0014.setI('kw1',   1)
wa0014.setI('kw2',  53)
ai0014 = DesAelItf('ai0014','blk0002','wa0014')
ai0014.setS('mode_file','cmodesai0014-D1.dat')
ai0014.setS('format','fmt_tp')

#.. Aeroelastic Problem definition
ael1=DesAeroelastics("ael1")
ael1.setS('format','fmt_tp')
# .. interpolated mesh deformation
ael1.setS('mesh_deformation_type','interpolated')
ael1.setS('mesh_deformation_technique','move3d')
#.. Fluids Dimensions (Metric values)
ael1.setF('fluid_density_unit',RoadimF)
ael1.setF('fluid_length_unit',LadimF)
ael1.setF('fluid_velocity_unit',QadimF)
ael1.setF('fluid_temperature_unit',TadimF)
#.. Structure Dimensions (Metric values)
ael1.setF('structure_mass_unit',1.)
ael1.setF('structure_length_unit',1.)
ael1.setF('structure_time_unit',1.)
ael1.setF('structure_temperature_unit',1.)
#.. kind of aeroelastic simulation
ael1.setS('simulation_type','forced_motion')
ael1.setS('forced_motion_type','harmonic')
ael1.setS('forced_motion_init','none')
#.. kind of structural model
ael1.setS('structural_model','modal')
#.. size of modal basis
ael1.setI('modal_basis_size',4)
#.. kind of modal basis
ael1.setS('modal_basis_kind','complex_modes')
ael1.setS('complex_excitation','direct')
#.. frequency for forced motion
#   F=206. Hz
ael1.setF('frequency',AELFREQUENCY)
#.. selected mode for forced motion
ael1.setI('mode_number',1)
#.. selected mode for forced motion
```

```
ael1.setF('modal_amplitude',0.001)
#.. initial phase for forced motion
ael1.setF('initial_phase',1.570796327)
ael1.setS('mesh_deformation_data_file','mmove.dat')
ael1.setS('mesh_deformation_output_file','mmove.dat.out')

#.. Connecting numerics, model, and aeroelastics with problem
problem.setS('global_numerics','num0001')
problem.setS('global_model','mod0001')
problem.setS('global_aeroelastics','ael1')
problem.compute()
problem.extract()
```

F.1.6 Linearised Euler computation with aeroelastic injection boundary condition

This is an example of Linearised Euler computation in the case of a fixed mesh, using an aeroelastic injection boundary condition. Using an injection boundary condition is valid only for Euler computations, and gives good results only for thin bodies.

```
from elsA_user import *
from elsA_Ael_user import *

#.. Declare an aelpb instead of a cfdpb
problem = aelpb(name='problem')
...

#.. Model
mod1 = model(name='mod1')
mod1.set('fluid','pg')
mod1.set('phymod','euler')
...

#.. Numerics
num1 = numerics(name='num1')
num1.set('time_algo','steady')
num1.set('ode','rk4')
num1.set('implicit','irs')
num1.set('flux','jameson')
num1.set('artviscosity','dismrt')
...

#.. Infinite state
stateInf = state(name='stateInf')
stateInf.set('ro',RoInf)
stateInf.set('rou',RouInf)
stateInf.set('rov',RovInf)
stateInf.set('row',RowInf)
stateInf.set('roe',RoeInf)
...

#.. Meshes
Msh0001 = mesh(name='Msh0001')
Msh0001.set('file','fgv0001')
...
```

```
#.. Blocks
Blk0001 = block(name='Blk0001')
Blk0001.set('deform','rigid')
...

#.. Boundaries
...

# Defines an aeroelastic interface associated to a wall window
w0004 = window('Blk0001',name='w0004')
w0004.set('wnd',[  31, 131,   1,   1,   1,   2])
ai0004=aelitf('Blk0001','w0004',name='ai0004')
ai0004.set('mode_file','mode0004.dat')
ai0004.set('format','fmt_tp')

# Defines a linearised aelwallslip aeroelastic boundary condition
b0004 = aelboundary('Blk0001','w0004',name='b0004')
b0004.set('type','aelwallslip')
b0004.set('ael_interface','ai0004')
...

#.. Aeroelastic Problem definition
ael1=aeroelastics(name='ael1')
ael1.set('fluid_density_unit',RoadimF)
ael1.set('fluid_length_unit',LadimF)
ael1.set('fluid_velocity_unit',QadimF)
ael1.set('fluid_temperature_unit',TadimF)
ael1.set('structure_mass_unit',1.)
ael1.set('structure_length_unit',1.)
ael1.set('structure_time_unit',1.)
ael1.set('structure_temperature_unit',1.)
# Select a linearised aeroelastic simulation type
ael1.set('simulation_type','linearized')
ael1.set('structural_model','modal')
ael1.set('modal_basis_size',2)
#.. selected frequency
ael1.set('frequency',34.4)
#.. selected mode
ael1.set('mode_number',1)
#.. selected amplitude
ael1.set('modal_amplitude',1.)
#.. initial phase for forced motion
ael1.set('initial_phase',1.57)
ael1.set('dynamic_pressure_ref',Pdyn)
ael1.set('surface_ref',1.)

#.. Initial steady state field
Ini0001 = aelinit('Dom0001',name='Ini0001')
Ini0001.set('steady_field_file','fac0001')
Ini0001.set('format','bin_v3d')
...

#.. Connecting numerics, model, and aeroelastics with problem
problem.set('global_numerics','num1')
problem.set('global_model','mod1')
problem.set('global_aeroelastics','ael1')
```

```
problem.compute()
problem.extract()
```

F.1.7 Linearised RANS computation in ALE formulation

This is an example of Linearised RANS computation. The mesh is frozen, but its deformation is taken into account through additional ALE terms in the linearised equations.

```
from elsA_user import *
from elsA_Ael_user import *

#.. Declare an aelpb instead of a cfdpb
problem = aelpb(name='problem')
...

#.. Model
mod1 = model(name='mod1')
mod1.set('fluid','pg')
mod1.set('phymod','nstur')
mod1.set('turbmod','spalart')
...

#.. Numerics
num1 = numerics(name='num1')
num1.set('time_algo','steady')
num1.set('ode','rk4')
num1.set('implicit','irs')
num1.set('flux','jameson')
num1.set('artviscosity','dismrt')
...

#.. Infinite state
stateInf = state(name='stateInf')
stateInf.set('ro',RoInf)
stateInf.set('rou',RouInf)
stateInf.set('rov',RovInf)
stateInf.set('row',RowInf)
stateInf.set('roe',RoeInf)
stateInf.set('v6',RoTur1)
...

#.. Meshes
Msh0001 = mesh(name='Msh0001')
Msh0001.set('file','fgv0001')
...

#.. Blocks
# For linearised ALE simulations, it is mandatory
# 1) to define <aelblock> objects instead of <block> objects
# 2) to set deform='staticdeformable'
# 3) to set deformation_type='aeroelastic_lin'
Blk0001 = aelblock(name='Blk0001')
Blk0001.set('deform','staticdeformable')
Blk0001.set('deformation_type','aeroelastic_lin')
```

```
Blk0001.set('mesh','Msh0001')
...

#.. Boundaries
...

# Defines an aeroelastic interface associated to a wall window
w0004 = window('Blk0001',name='w0004')
w0004.set('wnd',[  49, 169,   1,   1,   1,   2])
ai0004=aelitf('Blk0001','w0004',name='ai0004')
ai0004.set('mode_file','mode0004.dat')
ai0004.set('format','fmt_tp')

# Defines a linearised walladia boundary condition
# An <aelboundary> object must be defined instead of a <boundary> object
b0004 = aelboundary('Blk0001','w0004',name='b0004')
b0004.set('type','walladia')
b0004.set('ael_interface','ai0004')
...

#.. Aeroelastic Problem definition
ael1=aeroelastics(name='ael1')
ael1.set('mesh_deformation_technique','tfi')
# It is mandatory to set mesh_deformation_type='interpolated'
ael1.set('mesh_deformation_type','interpolated')
ael1.set('fluid_density_unit',RoadimF)
ael1.set('fluid_length_unit',LadimF)
ael1.set('fluid_velocity_unit',QadimF)
ael1.set('fluid_temperature_unit',TadimF)
ael1.set('structure_mass_unit',1.)
ael1.set('structure_length_unit',1.)
ael1.set('structure_time_unit',1.)
ael1.set('structure_temperature_unit',1.)
# Select a linearised aeroelastic simulation type
ael1.set('simulation_type','linearized')
#.. kind of structural model
ael1.set('structural_model','modal')
#.. size of modal basis
ael1.set('modal_basis_size',2)
#.. frequency for linearised forced motion
ael1.set('frequency',34.4)
#.. selected mode for linearised forced motion
ael1.set('mode_number',1)
#.. selected amplitude for linearised forced motion
ael1.set('modal_amplitude',1.)
#.. initial phase for linearised forced motion
ael1.set('initial_phase',1.57)
ael1.set('dynamic_pressure_ref',Pdyn)
ael1.set('surface_ref',1.)

#.. Initial steady state field
Ini0001 = aelinit('Dom0001',name='Ini0001')
Ini0001.set('steady_field_file','fac0001')
Ini0001.set('format','bin_v3d')
...
```

```
problem.set('global_numerics','num1')
problem.set('global_model','mod1')
problem.set('global_aeroelastics','ael1')
problem.compute()
problem.extract()
```

INDEX

Empty page

DIFFUSION SCHEME

Software Secretariat Archives (DSNA)

Redactors

At users' request

END of LIST