TITAN

Release 0.1

Fabio Morgado

Jan 30, 2023

CONTENTS:

TransatmospherIc flighT simulAtioN - A python code for multi-fidelity and multi-physics simulations of access-to-space and re-entry

For further info, check *Usage*. For installation, check *Installation*.

Warning: This library is still under development.

CHAPTER

ONE

USAGE

1.1 Installation

To install TITAN, it is required to use an Anaconda environment. The required libraries are listed in the requirements.txt file. In order to install the required packages, the Anaconda environment can be created using

```
$ conda create --name myenv --file requirements.txt
```

If the packages are not found, the user can append a conda channel to retrieve the packages, by running

```
$ conda config --append channels conda-forge
```

To activate the Conda environment:

```
$ conda activate myenv
```

After activation, the user needs to install other packages that were not possible (as GMSH) using the conda requirements. To do so, the user can use the pip manager to install the packages listed in the pip_requirements.txt

```
(.venv) $ pip install -r pip_requirements.txt
```

To install pymap3D, you can clone the following github page https://github.com/geospace-code/pymap3d/ into the Executables foldar and install using

```
(.venv) $ pip install -e pymap3d
```

1.1.1 Optional

Mutation++

The mutation++ package is an optional method to compute the freestream conditions. It can be installed by following the instructions in https://github.com/mutationpp/Mutationpp Once the mutation++ has been compiled, you can install by:

1. In the github link, go to the thirdpary folder and clone the Pybind repository into your thirdparty folder in mutationpp
2. In the Mutationpp root folder, run

```
(.venv) $ python setup.py build
(.venv) $ python setup.py install
```

AMGio

AMGio is a library that is required to perform mesh adaptation when running high-fidelity simulations. To install the AMGio library, one must clone the following github page to the TITAN/Executables folder: https://github.com/bmunguia/amgio. THe user can the proceed to the installation using

```
(.venv) $ pip install -e amgio/su2gmf/
```

1.1.2 GRAM model

TITAN has the capability to use the NASA-GRAM https://software.nasa.gov/software/MFS-33888-1 to retrieve the atmospheric properties of Earth, Neptune and Uranus. The user needs to request NASA to use the atmospheric model.

Once the GRAM tool is compiled, the user needs to link the binaries, and place them in the Executables folder

1.1.3 Troubleshooting

If mpirun is not working, the user may require to reinstall openmpi and/or mpi4py using pip, by following the steps:

```
(.venv) $ mamba uninstall mpi4py
(.venv) $ mamba uninstall openmpi
(.venv) $ pip install openmpi
(.venv) $ pip install mpi4py
```

1.2 Setting up the Configuration file

An explanation of the Configuration file can be found in the Config_temmplate.cfg file, in the root folder.

TITAN will read the configuration file using the config parser package. The file is divided into several subsections:

1.2.1 Options

- **Num_iters** - Maximum number of iterations
- **Load_State** - Load the last simulation state
- **Fidelity** - Select the level of the aerothermodynamics in the simulation (Low/High/Multi)
- **Output_folder** - Folder where the simulation solution is stored
- **Load_mesh** - Flag to indicate if the mesh should be loaded (if already pre-processed in previous simulation)
- **Load_state** - Flag to resume the simulation (overrules the flag Load_mesh)

1.2.2 Trajectory

- **Altitude** - Initial altitude [meters]
- **Velocity** - Initial Velocity [meters/second]
- **Flight_path_angle** - Initial FLight Path Angle [degree]
- **Heading_angle** - Initial Heading Angle [degree]
- **Latitude** - Initial Latitude [degree]
- **Longitude** - Initial Longitude [degree]

1.2.3 Model

- **Planet** - Name of the planel (Earth, Neptune, Uranus)
- **Vehicle** - Flag for use of custom vehicle parameters (Mass, Nose radius, Area of reference)
- **Drag** - Flag for use of drag model (if Vehicle = True)

1.2.4 Vehicle

- **Mass** - Mass of the vehicle [kg]
- **Nose_radius** - Nose radius of the vehicle [meters]
- **Area_reference** - Area of reference for coefficient computation [meters^2]
- **Drag_file** - Name of the Drag model containing the Mach vs drag coefficient information in TITAN/Model/Drag

1.2.5 Freestream

- **method** - Method used for the computation of the freestream (Standard, Mutationpp, GRAM)
- **model** - Atmospheric model (Earth - NRLMSISE00,GRAM ; Neptune - GRAM; Uranus - GRAM)

1.2.6 GRAM

- **MinMaxFactor** - Value of the MinMaxFactor for the NeptuneGRAM
- **ComputeMinMaxFactor** - Automatic computation of the MinMaxFactor for the NeptuneGRAM (see Neptune-GRAM manual. 0 = False, 1 = True)
- **SPICE_Path** - Path for the SPICE database
- **GRAM_Path** - Path for GRAM software (required for Earth GRAM)

1.2.7 Time

- **Time_step** - Value of the time step [second]

1.2.8 SU2

- **Solver** - Solver to be used in CFD simulation (EULER/NAVIER_STOKES or NEMO_EULER/NEMO_NAVIER_STOKES)
- **Num_iters** - Number of CFD iterations
- **Conv_method** - Convective scheme (Default = AUSM)
- **Adapt_iter** - Number of mesh adaptations
- **Num_cores** - Number of cores to run CFD simulation
- **Muscl** - Flag for MUSCL reconstruction (Yes/No)
- **Cfl** - CFL number

1.2.9 Bloom

- **Flag** - Flag to activate Bloom (True/False)
- **Layers** - Number of layers in the boundary layer
- **Spacing** - Spacing of the initial layer
- **Growth_Rate** - Growth rate between layers

1.2.10 AMG

- **Flag** - Flag to activate AMG
- **P** - Norm of the error estimate for the Hessian computation
- **C** - Correction for metric complexity
- **Sensor** - Name of the computational field used to compute the metric tensor for mesh adaptation

1.2.11 Assembly

- **Path** - Path for the geometry files
- **Connectivity** - Linkage information for the specified components in the Objects section
- **Angle_of_attack** - Angle of attack of the assembly [degree]
- **Sideslip** - Angle of sideslip of the assembly [degree]

1.2.12 Objects

- **Primitive used in the Assembly** - name_Marker = (NAME, TYPE, MATERIAL)
- **Joints used in the Assembly** - name_Marker = (NAME, TYPE, MATERIAL, TRIGGER_TYPE, TRIGGER_VALUE)
  - NAME -> Name of the geometry file in stl format
  - TYPE -> Type of the object (Primitive/Joint)
  - MATERIAL -> Material of the object, needs to one specified in the material database
  - TRIGGER_TYPE -> The criteria for the joint fragmentation (Altitude, time, iteration, Temperature)
  - TRIGGER_VALUE -> The value to trigger the fragmentation

1.3 Running a simulation

TITAN is called in the conda environment using

```
(.venv) $ python TITAN.py -c config.cfg
```

The solution is stored in the specifed output folder. The structure in the output folder is as **SPECIFY HERE**

After obtaining the solution of the simulation, the data can be postprocessed by introducing a new flag to the instruction, refering to the Postprocess method that can be **WIND** or **ECEF**. The following command does not run a new simulation, but it postprocess the already obtained solutions in the **Output_folder** specified field.

```
(.venv) $ python TITAN.py -c config.cfg -pp WIND
```

1.4 Geometry modelling

The frame convention in the geometry modelling are such that the X axis is the longitudinal axis pointing ahead, Z axis is the vertical axis pointing downwards, and the Y axis is the lateral one, pointing in such a way that the frame is right-handed.

In case of multiple components, if the components are in contact with each other, the respective meshes need to be identical in the interface (i.e. same node positioning and same facets).

CHAPTER

TWO

MODULES

2.1 TITAN

`TITAN.main(filename='', postprocess='')`

TITAN main function

Parameters

- **filename** (*str*) – Name of the configuration file
- **postprocess** (*str*) – Postprocess method. If specified, TITAN will only perform the post-process of the already obtained solution in the specified output folder. The config fille still needs to be specified.

`TITAN.loop(options=[], titan=[])`

Simulation loop for time propagation

The function calls the different modules to perform dynamics propagation, thermal ablation, fragmentation assessment and structural dynamics for each time iteration. The loop finishes when the iteration number is higher than the one the user specified.

Parameters

- **options** (Options) – object of class *configuration.Options*
- **titan** (Assembly_list) – object of class Assembly_list

2.2 Configuration

`class configuration.Trajectory(altitude=0, gamma=0, chi=0, velocity=0, latitude=0, longitude=0)`

Class Trajectory

A class to store the user-defined trajectory information

altitude

[meters] Altitude value.

chi

[radians] Heading Angle value.

gamma

[radians] Flight Path Angle value.

`latitude`
[radians] Latitude value.

`longitude`
[radians] Longitude value.

`velocity`
[meters/second] Velocity value.

`class configuration.Fenics(`*E=68000000000.0*, *FENICS=False*, *FE_MPI=False*, *FE_MPI_cores=12*, *FE_verbose=False*`)`

FEniCS class

Class to store the user-defined information for the structural dynamics simulation using FEniCS

`E`
[Pa] Young Modulus

`FE_MPI`
[bool] Flag value indicating if MPI is to be used for FEniCS

`FE_MPI_cores`
[int] Flag value indicating the number of cores if **FE_MPI=True**

`FE_verbose`
[bool] Flag value indicating the verbosity of FEniCS solver

`class configuration.Dynamics(`*time_step=0*, *time=0*, *propagator='EULER'*, *adapt_propagator=False*, *manifold_correction=True*`)`

Dynamics class

A class to store the user-defined dynamics options for the simulation

`adapt_propagator`
[bool] Flag value indicating time-step adaptation

`manifold_correction`
[bool] Flag value indicating manifold correction

`propagator`
[str] Name of the propagator to be used in the dynamics (options - EULER).

`time`
[seconds] Physical time of the simulation.

`time_step`
[seconds] Value of the time-step.

`class configuration.Aerothermo(`*heat_model='vd'*, *knc_pressure=0.0001*, *knc_heatflux=0.005*, *knf=100*`)`

Aerothermo class

A class to store the user-defined aerothemo model options

`heat_model`
[str] Name of the heatflux model to be used

`knc_heatflux`
[float] Value of the continuum knudsen for the heatflux computation

`knc_pressure`
[float] Value of the continuum knudsen for the pressure computation

`knf`
[float] Value of the free-molecular knudsen

`class configuration.Freestream`

Freestream class

A class to store the user-defined freestream properties per time iteration

`R`
[??] Value of the Gas constant

`Temperature`
[kelvin] Value of the freestream temperature

`Velocity`
[m/s] Value of the freestream velocity

`gamma`
[float] Value of the freestream specific heat ratio

`knudsen`
[float] Value of the freestream knudsen

`mach`
[float] Value of the freestream Mach number

`method`
Selection of freestream calculation method (Mutationpp, default = Standard)

`mfp`
[??]

`model`
[??]

`muEC`
[??]

`muSu`
[??]

`ninf`
[??]

`omega`
[??]

`percent_gas`
[??]

`prandtl`
[float] Value of the freestream prandtl

`pressure`
[Pa] Value of the freestream Pressure

rho

[kg/m^3] Value of the freestream density

class configuration.**Options**(*iters=1*, *time_step=0.1*, *fidelity='Low'*, *SPARTA=False*, *SP_NUM=1*, *sp_iters=0*, *SPARTA_MPI_cores=4*, *Opti_Flag='OFF'*, *fenics=False*, *FE_MPI=False*, *FE_MPI_cores=12*, *FE_verbose=False*, *case='benchmark'*, *E=68000000000.0*, *output_folder='TITAN_sol'*, *propagator='Euler'*, *adapt_propagator=False*, *assembly_rotation=[]*, *manifold_correction=True*, *adapt_time_step=False*, *rerr_tol=0.001*, *num_joints=0*, *frame_for_writing='W'*, *max_time_step=0.5*, *save_displacement=False*, *save_vonMises=False*)

Options class

A class that keeps the information of the selected user-defined options for all the disciplinary areas and methods required to run the simulation

aerothermo

[*Aerothermo*] Object of class Aerothermo

clean_up_folders()

Cleans the simulation output folder specified in the configuration file

create_output_folders()

Creates the folder structure to save the soluions

current_iter

[int] Current iteration

dynamics

[*Dynamics*] Object of class Dynamics

fenics

[*Fenics*] Object of class Fenics

fidelity

[str] Fidelity of the aerothermo calculation (Low - Default, High, Hybrid)

freestream

[*Freestream*] Object of class Freestream

iters

[int] Number of dynamic iterations

read_state()

Load last state of the TITAN object

Returns

titan – Object of class Assembly_list

Return type

Assembly_list

save_state(*titan*, *i=0*)

Saves the TITAN object state

Parameters

- **titan** (Assembly_list) – Object of class Assebly_list
- **i** (*int*) – Iteration number.

structural_dynamics
[boolean] Flag to perform structural dynamics

`configuration.`**`read_trajectory`**`(`*configParser*`)`
Read the Trajectory specified in the config file

Parameters
configParser (*configParser*) – Object of Config Parser

Returns
trajectory – Object of class Trajectory

Return type
Trajectory

`configuration.`**`read_geometry`**`(`*configParser*`)`
Geometry pre-processing

Reads the specified configuration file and creates a list with the information of the objects and assemblies

Parameters
configParser (*configParser*) – Object of Config Parser

Returns
titan – Object of class Assembly_list

Return type
Assembly_list

`configuration.`**`read_config_file`**`(`*configParser*, *postprocess=''*`)`
Read the config file

Parameters

- **configParser** (*configParser*) – Object of Config Parser
- **postprocess** (*str*) – Postprocess method of the solution. If not None, only returns output_folder

Returns

- **options** (*Options*) – Object of class Options
- **titan** (*Assembly_list*) – List of objects of class Assembly_list

2.3 Aerothermo

`aerothermo.`**`compute_aerothermo`**`(`*titan*, *options*`)`
Fidelity selection for aerothermo computation

Parameters

- **titan** (Assembly_list) – Object of class Assembly_list
- **options** (Options) – Object of class Options

`aerothermo.`**`compute_low_fidelity_aerothermo`**`(`*assembly*, *options*`)`
Low-fidelity aerothermo computation

Function to compute the aerodynamic and aerothermodynamic using low-fidelity methods. It can compute from free-molecular to continuum regime. For the transitional regime, it uses a bridging methodology.

Parameters

- **assembly** (Assembly_list) – Object of class Assembly_list
- **options** (Options) – Object of class Options

aerothermo.**backfaceculling**(*body*, *nodes*, *nodes_normal*, *free_vector*, *npix*)

Backface culling function

This function detects the facets that are impinged by the flow

Parameters

- **body** (Assembly) – Object of Assembly class
- **free_vector** (*np.array*) – Array with the freestream direction with respect to the Body frame
- **npix** (*int*) – Resolution of the matrix used for the facet projection methodology (pixels)

Returns
node_points – Array of IDs of the visible nodes

Return type
np.array

aerothermo.**bridging**(*free*, *Kn_cont*, *Kn_free*)

Computation of the bridging factor for the aeordynamic computation

Parameters

- **free** (*Assembly.Freestream*) – Freestream object
- **Kn_cont** (*float*) – Knudsen limit for the continuum regime
- **Kn_free** (*float*) – Knudsen limit for the free-molecular regime

Returns
AeroBridge – Bridging factor

Return type
float

aerothermo.**compute_aerodynamics**(*assembly*, *obj*, *index*, *flow_direction*, *options*)

Low-fidelity computation of the aerodynamics (pressure, friction)

Parameters

- **assembly** (Assembly_list) – Object of class Assembly_list
- **obj** (Component) – Object of class Component
- **index** (*np.array(int)*) – Indexing list indicating nodes facing the flow (backface culling)
- **flow_direction** (*np.array(float)*) – Array indicating direction of the flow in the body frame
- **options** (Options) – Object of class Options

aerothermo.**compute_aerothermodynamics**(*assembly*, *obj*, *index*, *flow_direction*, *options*)

Low-fidelity computation of the aerothermodynamics (heat-flux)

Parameters

- **assembly** (Assembly_list) – Object of class Assembly_list
- **obj** (Component) – Object of class Component

- **index** (*np.array(int)*) – Indexing list indicating nodes facing the flow (backface culling)
- **flow_direction** (*np.array(float)*) – Array indicating direction of the flow in the body frame
- **options** (*Options*) – Object of class Options

aerothermo.**aerodynamics_module_continuum**(*nodes_normal*, *free*, *p*, *flow_direction*)

Pressure computation for continuum regime

Function uses the Modified Newtonian Theory

Parameters

- **nodes_normal** (*np.array*) – List of the normals of each vertex on the surface
- **free** (*Assembly.Freestream*) – Freestream object
- **p** (*np.array*) – List of vertex IDs that are visible to the flow
- **flow_direction** (*np.array*) – Vector containing the flow_direction in the Body frame

Returns
Pressure – Vector with pressure values

Return type
np.array

aerothermo.**aerodynamics_module_bridging**(*nodes_normal*, *free*, *p*, *aerobridge*, *flow_direction*, *wall_temperature*)

Pressure computation for Transitional regime

Parameters

- **nodes_normal** (*np.array*) – List of the normals of each vertex on the surface
- **free** (*Assembly.Freestream*) – Freestream object
- **p** (*np.array*) – List of vertex IDs that are visible to the flow
- **aerobridge** (*float*) – Bridging value between 0 and 1
- **flow_direction** (*np.array*) – Vector containing the flow_direction in the Body frame
- **body_temperature** (*float*) – Temperature of the body

Returns

- **Pressure** (*np.array*) – Vector with pressure values
- **Shear** (*np.array*) – Vector with skin friction values

aerothermo.**aerodynamics_module_freemolecular**(*nodes_normal*, *free*, *p*, *flow_direction*, *body_temperature*)

Pressure computation for Free-molecular regime

Function uses the Schaaf and Chambre theory

Parameters

- **nodes_normal** (*np.array*) – List of the normals of each vertex on the surface
- **free** (*Assembly.Freestream*) – Freestream object
- **p** (*np.array*) – List of vertex IDs that are visible to the flow
- **flow_direction** (*np.array*) – Vector containing the flow_direction in the Body frame
- **body_temperature** (*float*) – Temperature of the body

Returns

- **Pressure** (*np.array*) – Vector with pressure values
- **Shear** (*np.array*) – Vector with skin friction values

`aerothermo.`**`aerothermodynamics_module_continuum`**(*nodes_normal*, *nodes_radius*, *free*, *p*, *body_temperature*, *flow_direction*, *hf_model*)

Heatflux computation for continuum regime

Function uses the Scarab equation (sc) or the Van Driest equation (vd)

Parameters

- **nodes_normal** (`np.array`) – List of the normals of each vertex on the surface
- **nodes_radius** (`np.array`) – Local radius of each vertex
- **free** (`Assembly.Freestream`) – Freestream object
- **p** (`np.array`) – List of vertex IDs that are visible to the flow
- **body_temperature** (`float`) – Temperature of the body
- **flow_direction** (`np.array`) – Vector containing the flow_direction in the Body frame
- **hf_model** (`str`) – Heatflux model to be used (default = ??, sc = Scarab, vd = Van Driest)

Returns
Stc – Vector with Stanton number

Return type
np.array

`aerothermo.`**`aerothermodynamics_module_bridging`**(*nodes_normal*, *nodes_radius*, *free*, *p*, *wall_temperature*, *flow_direction*, *atm_data*, *hf_model*, *Kn_cont*, *Kn_free*, *lref*, *assembly*, *options*)

Heatflux computation for the heat-flux regime

Parameters

- **nodes_normal** (`np.array`) – List of the normals of each vertex on the surface
- **nodes_radius** (`np.array`) – Local radius of each vertex
- **free** (`Assembly.Freestream`) – Freestream object
- **p** (`np.array`) – List of vertex IDs that are visible to the flow
- **wall_temperature** (`float`) – Temperature of the body
- **flow_direction** (`np.array`) – Vector containing the flow_direction in the Body frame
- **atm_data** (`str`) – Atmospheric model
- **hf_model** (`str`) – Heatflux model to be used (default = ??, sc = Scarab, vd = Van Driest)
- **Kn_cont** (`float`) – Knudsen limit for the continuum regime
- **Kn_free** (`float`) – Knudsen limit for the free-molecular regime
- **lref** (`float`) – Reference length
- **options** (`Options`) – Object of class Options

Returns
St – Vector with Stanton number

Return type
np.array

aerothermo.**aerothermodynamics_module_freemolecular**(*nodes_normal*, *free*, *p*, *flow_direction*, *Wall_Temperature*)

Heatflux computation for free-molecular regime

Function uses the Schaaf and Chambre Theory Based on book of Wallace Hayes - Hypersonic Flow Theory

Parameters

- **nodes_normal** (*np.array*) – List of the normals of each vertex on the surface
- **free** (*Assembly.Freestream*) – Freestream object
- **p** (*np.array*) – List of vertex IDs that are visible to the flow
- **Wall_temperature** (*float*) – Temperature of the body
- **flow_direction** (*np.array*) – Vector containing the flow_direction in the Body frame

Returns
Stfm – Vector with Stanton number

Return type
np.array

2.4 Dynamics

class dynamics.**DerivativesAngle**(*droll=0*, *dpitch=0*, *dyaw=0*, *ddroll=0*, *ddpitch=0*, *ddyaw=0*)

Class DerivativesAngle

A class to store the derivatives information regarding the angular dynamics in the body frame

ddpitch
[float] Second derivative of the pitch angle

ddroll
[float] Second derivative of the roll angle

ddyaw
[float] Second derivative of the yaw angle

dpitch
[float] Derivative of the pitch angle

droll
[float] Derivative of the roll angle

dyaw
[float] Derivative of the yaw angle

class dynamics.**DerivativesCartesian**(*dx=0*, *dy=0*, *dz=0*, *du=0*, *dv=0*, *dw=0*)

Class DerivativesCartesian

A class to store the derivatives information of position and velocity in the cartesian (ECEF) frame

du
[float] Derivative of the X-velocity

dv

[float] Derivative of the Y-velocity

dw

[float] Derivative of the Z-velocity

dx

[float] Derivative of the X-position

dy

[float] Derivative of the Y-position

dz

[float] Derivative of the Z-position

dynamics.**integrate**(*titan*, *options*)

Time integration

This function calls a time integration scheme

Parameters

- **titan** (Assembly_list) – Object of class Assembly_list
- **options** (Options) – Object of class Options

dynamics.**compute_angular_derivatives**(*assembly*)

Computation of the angular derivatives in the Body frame

This function computes the angular dericatives taking into consideration the euler and aerodynamic moments

Parameters

assembly (Assembly) – Object of Assembly class

dynamics.**compute_cartesian_derivatives**(*assembly*, *options*)

Computation of the cartesian derivatives

This function computes the cartesian derivatives of the position and velocity It uses the gravity, aerodynamic, centrifugal and coriolis forces for the acceleration computation.

Parameters

- **assembly** (Assembly) – Object of class Assembly
- **options** (Options) – Object of class Options

dynamics.**compute_cartesian**(*assembly*, *options*)

Computation of the cartesian dynamics

This function computes the cartesian position and velocity of the assembly

Parameters

- **assembly** (Assembly) – Object of class Assembly
- **options** (Options) – Object of class Options

dynamics.**compute_quaternion**(*assembly*)

Computation of the quaternion

This function computes the quaternion value of the body frame with respect to the ECEF frame The quaternion will give the rotation matrix that will allow to pass from Body to ECEF

Parameters
`assembly` (`Assembly`) – Object of Assembly class

`euler.compute_Euler(titan, options)`

Euler integration

Parameters

- `titan` (`Assembly_list`) – Object of class Assembly_list
- `options` (`Options`) – Object of class Options

`euler.update_position_cartesian(assembly, cartesianDerivatives, angularDerivatives, options)`

Update position and attitude of the assembly

Parameters

- `assembly` (`Assembly`) – Object of class Assembly
- `cartesianDerivatives` (`DerivativesCartesian`) – Object of class DerivativesCartesian
- `angularDerivatives` (`DerivativesAngle`) – Object of class DerivativesAngle
- `options` (`Options`) – Object of class Options

2.5 Forces

`forces.compute_aerodynamic_forces(titan, options)`

Computes the aerodynamic forces in the wind frame

Parameters

- `titan` (`Assembly_list`) – Object of class Assembly_list
- `options` (`Options`) – Object of class Options

`forces.compute_aerodynamic_moments(titan, options)`

Computes the aerodynamic moments in the wind Body frame

Parameters

- `titan` (`Assembly_list`) – Object of class Assembly_list
- `options` (`Options`) – Object of class Options

`forces.compute_inertial_forces(assembly, options)`

Computes the inertial forces in the Body Frame

This functions computes the inertial forces that will be used for the Structurla dynamics

Parameters

- `assembly` (`Assembly`) – Object of class Assembly
- `options` (`Options`) – Object of class Options

2.6 Fragmentation

fragmentation.**fragmentation**(*titan*, *options*)

Check if components meet the specified criteria to be removed from the simulation. At the moment, only altitude, iteration number, time and total ablation are specified.

Parameters

- **titan** (Assembly_list) – Object of class Assembly_list
- **options** (Options) – Object of class Options

fragmentation.**demise_components**(*titan*, *assembly_pos*, *joints_id*, *options*)

Computes the inertial forces in the Body Frame

This functions computes the inertial forces that will be used for the Structurla dynamics

Parameters

- **titan** (Assembly_list) – Object of class Assembly_list
- **assembly_pos** (*array*) – Array containing the index position of the assemblies that will undergo fragmentation
- **joints_id** (*array*) – Array containing the index of the joints that demised (index in relation to each assembly that will undergo fragmentation), to be removed from the simulation
- **options** (Options) – Object of class Options

2.7 Freestream

atmosphere.**load_atmosphere**(*name*)

This function loads the atmosphere model with respect to the user specification

Parameters

name (*str*) – Name of the atmospheric model

Returns

- **f** (*scipy.interpolate.interp1d*) – Function interpolation of the atmopshere atributes with respect to altitude
- **spacies_index** (*array*) – Array with the species used in the model

mix_mpp.**mixture_mpp**(*species*, *density*, *temperature*)

Retrieve the mixture object of the Mutation++ library

Parameters

- **species** (*array*) – Species used for the mixture
- **density** (*array*) – Density of each species
- **temperature** (*float*) – Temperature of the mixture

Returns

mix – Object of the mpp Mixture

Return type

mpp.Mixture()

mix_properties.**compute_freestream**(*model*, *altitude*, *velocity*, *lref*, *freestream*, *assembly*, *options*)

Compute the freestream properties

The user needs to specify the method for the freestream computation (Standard, Mutationpp)

Parameters

- **model** (*str*) – Name of the atmospheric model
- **altitude** (*float*) – Altitude value in meters
- **velocity** (*float*) – Velocity value in meters
- **lref** (*float*) – Refence length in meters
- **freesteam** (Freestream) – Object of class assembly.freestream
- **options** (Options) – Object of class Options

mix_properties.**compute_stagnation**(*free*, *options*)

Compute the post-shock stagnation values

Parameters

- **free** (Freestream) – Object of class assembly.freestream
- **options** (Options) – Object of class Options

2.8 Geometry

2.8.1 Assembly

class assembly.**Assembly_list**(*objects*)

Class Assembly list

A class to store a list of assemblies and respective information, as well as the number of iterations and simulation time

assembly

[array] List of assemblies

connectivity

[array] List of the linkage information between the different components

create_assembly(*connectivity*, *aoa=0.0*, *slip=0.0*, *roll=0.0*)

Creates the assembly list

Parameters

- **connectivty** (*array*) – array containing the user-defined connectivity between the different components
- **aoa** (*float*) – Angle of attack in degrees
- **slip** (*float*) – Slip angle in degrees
- **roll** (*float*) – Roll angle in degrees

id

[array] Number ID to identify the assembly. Whenever an assembly is generated (i.e. due to fragmentation/ablation or during the preprocessing phase), it will have this number ID.

iter

[int] Iteration

objects

[array] List of components

time

[float] simulation physical time

class assembly.**Dynamics**(*roll=0*, *pitch=0*, *yaw=0*, *vel_roll=0*, *vel_pitch=0*, *vel_yaw=0*)

Class Dynamics

A class to store the dynamics information of the assembly

pitch

[float] Pitch angle in radians

roll

[float] Roll angle in radians

vel_pitch

[float] Pitch angular velocity in rad/s

vel_roll

[float] Roll angular velocity in rad/s

vel_yaw

[float] Yaw angular velocity in rad/s

yaw

[float] Yaw angle in radians

class assembly.**Body_force**(*force=array([[0.0], [0.0], [0.0]])*, *moment=array([[0.0], [0.0], [0.0]])*)

Class Body_force

A class to store the force and moment information that the assembly experiences at each iteration in the body frame

force

[np.array] Force array (3x1)

moment

[np.array] Moment array (3x1)

class assembly.**Wind_force**(*lift=0*, *drag=0*, *crosswind=0*)

Class Wind_force

A class to store the force information that the assembly experiences at each iteration in the wind frame

crosswind

[float] Crosswind force

drag

[float] Drag force

lift

[float] Lift force

class assembly.**Freestream**(*pressure=0*, *mach=0*, *gamma=0*, *knudsen=0*, *prandtl=0*, *temperature=0*, *density=0*, *velocity=0*, *R=0*, *mfp=0*, *omega=0*, *diameter=0*, *mu=0*, *cp=0*, *cv=0*)

Class Freestream

A class to store freestream information with respect to the position and velocity of each assembly

P1_s
[float] Pressure at the stagnation point

R
[float] Gas constant

T1_s
[float] Temperature at the stagnation point

cp
[float] Heat capacity at constant pressure

cv
[float] Heat capacity at constant volume

density
[float] Freestream density [kg/m^3]

diameter
[float] Mean diameter

gamma
[float] Freestream specific heat ratio

h1_s
[?float?] Specific enthalpy at the stagnation point

knudsen
[float] Freestream knudsen

mach
[float] Freestream mach

mfp
[float] mean free path in meters

mu
[float] Mean viscosity

mu_s
[float] Viscosity at the stagnation point

omega
[float] Mean viscosity coefficient

percent_gas
[array (float)] percentage of species in the mixture with respect to moles

percent_mass
[array (float)] percentage of species in the mixture with respect to mass

prandtl
[float] Freestream prandtl

pressure
[float] Freestream pressure [Pa]

rho_s
[float] Density at the stagnation point

species_index
[array (str)] list of species in the mixture

temperature
[float] Freestream temperature [K]

velocity
[float] Freestream velocity [m/s]

class assembly.**Aerothermo**(*n_points*)

Class Aerothermo

A class to store the surface quantities

heatflux
[np.array] Heatflux [W]

pressure
[np.array] Pressure [Pa]

shear
[np.array] Skin friction [Pa]

class assembly.**Assembly**(*objects=[]*, *id=0*, *aoa=0.0*, *slip=0.0*, *roll=0.0*)

Class Assembly

A class to store the information respective to each assemly at every time iteration

Aref
[float] Area of reference [meters^2]

COG
[array] CYZ coordinates of the center of mass in the body frame [meters]

Lref
[float] Length of reference [meters]

aerothermo
[Aerothermo] Object of class Aerothermo to store the surface quantities

aoa
[float] Angle of attack [radians]

body_force
[Body_force] Object of class Body_force to store the force and moment information in the body frame

compute_mass_properties()

Computes the inertial properties

Function to compute the inertial properties using the 3D domain information.

dynamics
[Dynamics] Object of class Dynamics to store the dynamics information

freestream
[Freestream] Object of class Freestream to store the freestream information

generate_inner_domain(*write=False*, *output_folder=''*, *output_filename=''*, *bc_ids=[]*)
Generates the 3D structural mesh

Generates the tetrahedral inner domain using the GMSH software

Parameters

- **write** (*bool*) – Flag to output the 3D domain
- **output_folder** (*str*) – Directory of the output folder when writing the 3D domain
- **output_filename** (*str*) – Name of the file

id
[int] ID of the assembly

inertia
[array] Inertia matrix in the body frame [kg/m^2]

mass
[float] Mass of the assembly [kg]

mesh
[Mesh] Object of class Mesh containing the grid information

objects
[array] List of the components that are part of the assembly

slip
[float] Slip angle [radians]

wind_force
[Body_force] Object of class Wind_force to store the force information in the wind frame

assembly.**create_assembly_flag**(*list_bodies*, *Flags*)
Generates the assembly connectivity matrix

Creates a flag m*n where m is the number of assemblies and n is the sum of all components used in the simulation. For every component belonging to a Body, the flag is True on that position. The assemblies are created according to the generated matrix

Parameters

- **list_bodies** (*array of components*) – array containing the used-defined components
- **Flags** (*np.array*) – numpy array containing the linkage information of each component

Returns
assembly_flag – numpy array containing information on how to generate the assemblies with respect to the components introduced in the simulation

Return type
np.array

2.8.2 Component

class component.**Component**(*filename*, *file_type*, *inner_stl=''*, *id=0*, *binary=True*, *temperature=300*, *trigger_type='Indestructible'*, *trigger_value=0*, *fenics_bc_id=-1*, *material='Unittest'*)

Component class

Class to store the information of a singular component.

COG

[meters] Center of mass in XYZ coordinates

compute_mass_properties(*coords*, *elements*)

Compute the inertia properties

Uses the volumetric grid information, along with the material density to compute the mass, Center of mass and inertia matrix using tetras

Parameters

- **coords** (*np.array*) – numpy array containing the XYZ coordinates of the vertex of each tetrahedral element
- **elements** (*np.array*) – numpy array containing the connectivity information of each tetrahedral element

id

[int] ID of the component

inertia

[kg/m^2] Inertia matrix

mass

[kg] Mass of the component

material

[Material] Object of class Material to store the material properties

mesh

[Mesh] Object of class mesh that stores the mesh information

name

[str] Name of the file where the mesh is stores

temperature

[K] Temperature

trigger_type

[str] Type of trigger for type joint (Altitude, Temperature, Stress)

trigger_value

[float] Value of the trigger criteria

type

[str] Type of the component (joint, primitive). Several sub-components can be used to form a larger component

2.9 Material

class material.**Material**(*name*)

Class Material

A class to store the material properties for each user-defined component

density

[float] Density of the material

emissivity

[float] Emissivity value

heatConductivity

[float] Heat conductivity value

material_density(*index*)

Function to retrieve the material density

Returns
density – Return material density

Return type
float

material_emissivity(*index*)

Function to retrieve the emissivity value

Returns
emissivity – Return emissivity value

Return type
float

material_heatConductivity(*index*)

Function to retrieve the material heat conductivity

Returns
heatConductivity – Return interpolation function for the heat conductivity

Return type
scipy.interpolate.interp1d

material_meltingHeat(*index*)

Function to retrieve the melting Heat value

Returns
meltingHeat – Return melting heat value

Return type
float

material_meltingTemperature(*index*)

Function to retrieve the melting temperature value

Returns
meltingTemperature – Return melting temperature value

Return type
float

`material_name`(*index*)

Function to retrieve the material name

Returns
name – Return material name

Return type
str

`material_oxideActivationTemperature`(*index*)

Function to retrieve the oxide activation Temperatire

Returns
oxideActivationTemperature – Return oxide activation temperature value

Return type
float

`material_oxideEmissivity`(*index*)

Function to retrieve the material oxide emissivity

Returns
oxideEmissivity – Return interpolation function for the oxide emissivity

Return type
scipy.interpolate.interp1d

`material_oxideHeatOfFormation`(*index*)

Function to retrieve the oxide heat of formation

Returns
oxideHeatofFormation – Return oxide heat of formation value

Return type
float

`material_oxideReactionProbability`(*index*)

Function to retrieve the oxide reaction probability

Returns
oxideReactionProbability – Return oxide reaction probability

Return type
float

`material_specificHeatCapacity`(*index*)

Function to retrieve the material specific heat capacity

Returns
specificHeatCapacity – Return interpolation function for the specific heat capacity

Return type
scipy.interpolate.interp1d

`material_yieldStress`(*index*)

Function to retrieve the material yield stress

Returns
yieldStress – Return interpolation function for the yield Stress

Return type
scipy.interpolate.interp1d

`material_youngModulus(index)`
Function to retrieve the young Modulus

Returns
youngModulus – Return interpolation function for the young Modulus

Return type
scipy.interpolate.interp1d

`meltingHeat`
[float] Melting Heat value of the material

`meltingTemperature`
[float] Melting Temperature value

`name`
[str] Name of the material

`oxideActivationTemperature`
[float] Oxidation activation temperature value

`oxideEmissivity`
[float] Oxidation emissivity value

`oxideHeatOfFormation`
[float] Oxidation Formation Heat value

`oxideReactionProbability`
[float] Oxidation reaction probability value

`specificHeatCapacity`
[float] Specific Heat Capacity value of the material

`yieldStress`
[float] Yield Stress value

`youngModulus`
[float] Young Modulus value

2.10 SU2

`class su2.Solver(restart, su2, freestream)`
Class Solver

A class to store the solver information. The class in the su2.py file is hardcoded to work with SU2.

`fluid_model`
[str] Fluid Model (Fluid model = MUTATIONPP -> Uses the Mutationpp Library)

`gas_composition`
[str] Gas Composition

`gas_model`
[str] Gas Model (Gas to be used in the simulation)

`kind_turb_model`
[str] Turbulence Model (Default = NONE)

`restart`
: [str] Restart boolean (if YES, the CFD simulation will restart from previous solution)

`solver`
: [str] Solver (EULER, NAVIER-STOKES, NEMO_EULER, NEMO_NAVIER_STOKES)

`transport_coeff`
: [str] Transport Coefficient

class `su2.Solver_Freestream_Conditions`(*freestream*)

Class Solver Freestream Conditions

A class to store the freestream conditions used in the CFD simulation The class in the su2.py file is hardcoded to work with SU2.

`aoa`
: [str] Angle of attack in deg

`init_option`
: [str] Initialization option to be used to compute the freestream (Default = TD_CONDITIONS)

`mach`
: [str] Mach number

`pressure`
: [str] Freestream Pressure in Pa

`temperature`
: [str] Freestream Temperature in K

class `su2.Solver_Reference_Value`

Class Solver Reference value

A class to store the reference values for the coefficient and moment computation The class in the su2.py file is hardcoded to work with SU2.

`origin_moment_x`
: [str] x-coordinate to which the moment is computed

`origin_moment_y`
: [str] y-coordinate to which the moment is computed

`origin_moment_z`
: [str] z-coordinate to which the moment is computed

`ref_area`
: [str] Area of reference

`ref_length`
: [str] Length of reference

class `su2.Solver_BC`(*assembly*, *su2*)

Class Solver Boundary conditions

A class to store the applied boundary conditions The class in the su2.py file is hardcoded to work with SU2.

`euler`
: [str] Euler Marker

farfield
[str] Farfield Marker

iso
[str] Isothermal Marker

monitor
[str] Monitoring Marker

outlet
[str] Outlet Marker

plot
[str] Plot Marker

class su2.Solver_Numerical_Method(*su2*)
Class Solver Numerical Method

A class to store the solver numerical methods The class in the su2.py file is hardcoded to work with SU2.

class su2.Flow_Numerical_Method(*su2*)
Class Flow Numerical Method

A class to store the flow numerical methods The class in the su2.py file is hardcoded to work with SU2.

conv_method
[str] Convective method (AUSM, AUSMPLUSUP2)

limiter
[str] Limiter method (Default = VENKATAKRISHNAN_WANG)

limiter_coeff
[str] Limiter coefficiet (Default = 0.01)

muscl
[str] MUSCL activation boolean (Default = YES)

time
[str] Time discretization (Default = EULER_EXPLICIT)

class su2.Solver_Convergence
Class Solver convergence

A class to store the convergence criteria The class in the su2.py file is hardcoded to work with SU2.

cauchy_elems
[str] Number of elements to be used in the Cauchy convergence window

cauchy_eps
[str] Residual for convergence using the Cauchy Window

field
[str] Fields to look for convergence

res_min
[str] Minimum residual for convergence

start_iter
[str] Start iteration for the convergence assessment

class su2.**Solver_Input_Output**(*it*, *iteration*, *output_folder*, *cluster_tag*)

Class Solver Input Output

A class to store the IO information. The class in the su2.py file is hardcoded to work with SU2.

mesh_filename
[str] Name of the mesh to be used in the simulation

mesh_format
[str] Mesh format (Default = SU2)

output_files
[str] Generated output files

output_freq
[str] Frequency for the output file generation

output_surf
[str] Name of the surface solution filename to write the simulation data

output_vol
[str] Name of the volume solution filename to write the simulation data

screen
[str] Screen output

solution_input
[str] Solution filename to read

solution_output
[str] Solution filename to write

tabular_format
[str] Solution format

class su2.**SU2_Config**(*freestream*, *assembly*, *restart*, *it*, *iteration*, *su2*, *options*, *cluster_tag*)

Class SU2 Configuration

A class to store all the information required write the SU2 configuration file The class in the su2.py file is hardcoded to work with SU2.

bc
[Solver_BC] Object of class Solver_BC

convergence
[Solver_Convergence] Object of class Solver_Convergence

flow
[Flow_Numerical_Method] Object of class Flow_Numerical_Method

free_cond
[Solver_Freestream_Conditions] Object of class Solver_Freestream_Conditions

inout
[Solver_Input_Output] Object of class Solver_Input_Output

name
[str] Name of the configuration file

num

[Solver_Numerical_Method] Object of class Solver_Numerical_Method

ref

[Solver_Reference_Value] Object of class Solver_Reference_Value

solver

[Solver] Object of class Solver

`su2.`**`write_SU2_config`**(*freestream*, *assembly*, *restart*, *it*, *iteration*, *su2*, *options*, *cluster_tag*, *input_grid*, *output_grid=''*, *interpolation=False*, *bloom=False*, *interp_to_BL=False*)

Write the SU2 configuration file

Generates a configuration file to run a SU2 CFD simulation according to the position of the object and the user-defined parameters.

Parameters

- **freestream** (`Freestream`) – Object of class Freestream
- **assembly** (`Assembly_list`) – Object of class Assembly_list
- **restart** (*bool*) – Boolean value to indicate if CFD simulation is restarting from previous solution
- **it** (*int*) – Value of adaptive iteration
- **iteration** (*int*) – Value of time iteration
- **su2** (*CFD*) – Object of class CFD
- **options** (`Options`) – Object of class Options
- **cluster_tag** (*int*) – Value of the cluster tag number for simulation parallelization
- **input_grid** (*str*) – Name of the input mesh file
- **output_grid** (*str*) – Name of the output file

`su2.`**`retrieve_index`**(*SU2_type*)

Retrieve index to retrieve solution fields

Returns the index to read the correct fields in the solution file, according to the user-specified solver

Parameters

SU2_type (*str*) – Solver used in the CFD simulation

Returns

index – Array of index with the position of solution fields of interest

Return type

np.array()

`su2.`**`read_vtk_from_su2_v2`**(*filename*, *assembly_coords*, *idx_inv*, *options*)

Read the VTK file solution

Reads and retrieves the solution stored in the VTK file format

Parameters

- **filename** (*str*) – Name and location of the VTK solution file
- **assembly_coords** (*np.array()*) – Coordinates of the mesh nodes
- **idx_inv** (*np.array*) – Sort indexing such that the VTK retrieved solution corresponds to the stored mesh nodes positioning

- **options** (Options) – Object of class Options

Returns
aerothermo – object of class Aerothermo

Return type
Aerothermo

su2.**split_aerothermo**(*total_aerothermo*, *assembly*)
Split the solution into the different assemblies used in the CFD simulation

Parameters

- **total_aerothermo** (Aerothermo) – Object of class Aerothermo
- **assembly** (*List_Assembly*) – Object of class List_Assembly

su2.**run_SU2**(*n*, *options*)
Calls the SU2 executable and run the simulation

Parameters

- **n** (*int*) – Number of cores
- **options** (Options) – Object of class Options

su2.**generate_BL**(*assembly*, *options*, *it*, *cluster_tag*)
Generates a Boundary Layer

Parameters

- **assembly** (*List_Assembly*) – Object of class List_Assembly
- **options** (Options) – Object of class Options
- **it** (*int*) – Value of adaptive iteration
- **cluster_tag** (*int*) – Value of Cluster tag

su2.**adapt_mesh**(*assembly*, *options*, *it*, *cluster_tag*)
Anisotropically adapts the mesh

Parameters

- **assembly** (*List_Assembly*) – Object of class List_Assembly
- **options** (Options) – Object of class Options
- **it** (*int*) – Value of adaptive iteration
- **cluster_tag** (*int*) – Value of Cluster tag

su2.**compute_cfd_aerothermo**(*assembly_list*, *options*, *cluster_tag=0*)
Compute the aerothermodynamic properties using the CFD software

Parameters

- **assembly_list** (*List_Assembly*) – Object of class List_Assembly
- **options** (Options) – Object of class Options
- **cluster_tag** (*int*) – Value of Cluster tag

2.11 Multi-fidelity switch

switch.**compute_aerothermo**(*titan*, *options*)

Aerothermo computation using a multi-fidelity approach (i.e. can use both low- and high-fidelity methodology)

The function uses the Billig formula to assess the shock envelope criteria, used to determine wether to use low- or high-fidelity methods

Parameters

- **titan** (*List_Assembly*) – Object of class List_Assembly
- **options** (Options) – Object of class Options

switch.**compute_billig**(*M*, *theta*, *center*, *sphere_radius*, *index_assembly*, *assembly*, *list_assembly*, *index_object*, *i*, *true_assembly*)

Computation of the shock envelope using the Billing formula

if the object is inside the shock envelope generated by an upstream body, the framework will use the high-fidelity methodology to compute the aero thermodynamics. Else, it will use low-fidelity methodology

Parameters

- **M** (*float*) – Freestream Mach number
- **theta** (*float*) – Shockwave inclination angle
- **center** (*np.array()*) – Coordinates of the sphere center
- **sphere_radius** (*float*) – Radius of the sphere
- **index_assembly** (*int*) – Index of the assembly producing the shockwave
- **assembly** (*List_Assembly*) – Object of List_Assembly
- **list_assembly** (*np.array()*) – Index of the remaining assemblies to check if they are inside or outside the shock envelope

Returns

computational_domain_bodies – List of bodies inside the shock envelope

Return type

List

CHAPTER THREE

EXAMPLE

Examples of configuration files can be found in the folder TITAN/Examples and can be run by the user.

INDEX

D

E

F

G

H

I

K

L

M

S

T

U

V

W

Y