StdAtm

Release unknown

unknown

Feb 12, 2024

CONTENTS

StdAtm provides a numpy-oriented Standard Atmosphere model.

See class *Atmosphere* for usage.

CHAPTER

ONE

CONTENTS

1.1 stdatm

1.1.1 stdatm package

Subpackages

Submodules

stdatm.atmosphere module

Simple implementation of International Standard Atmosphere.

class stdatm.atmosphere.**Atmosphere**(*altitude: Union[float, Sequence[float]]*, *delta_t: Union[float, Sequence[float]] = 0.0*, *altitude_in_feet: bool = True*)

Bases: object

Simple implementation of International Standard Atmosphere for troposphere and stratosphere.

Atmosphere properties are provided in the same "shape" as provided altitude:

- if altitude is given as a float, returned values will be floats
- if altitude is given as a sequence (list, 1D numpy array, ...), returned values will be 1D numpy arrays
- if altitude is given as nD numpy array, returned values will be nD numpy arrays

Usage:

Also, after instantiating this class, setting one speed parameter allows to get value of other ones. Provided speed values should have a shape compatible with provided altitudes.

```
>>> atm1 = Atmosphere(30000)
>>> atm1.true_airspeed = [100.0, 250.0]
>>> atm1.mach
array([0.32984282, 0.82460705])

>>> atm2 = Atmosphere([0, 1000, 35000])
>>> atm2.equivalent_airspeed = 200.0
>>> atm2.true_airspeed
array([200.        , 202.95792913, 359.28282052])

>>> atm2.mach = [1.0, 1.5, 2.0]
>>> atm2.true_airspeed
```

(continues on next page)

(continued from previous page)

```
array([340.29526405, 508.68507243, 593.0730464 ])

>>> atm2.equivalent_airspeed = [[300, 200, 100],[50, 100, 150]]
>>> atm2.true_airspeed
array([[300.        , 202.95792913, 179.64141026],
       [ 50.        , 101.47896457, 269.46211539]])
```

Parameters

- **altitude** – altitude (units decided by altitude_in_feet)
- **delta_t** – temperature increment (°C) applied to whole temperature profile
- **altitude_in_feet** – if True, altitude should be provided in feet. Otherwise, it should be provided in meters.

get_altitude(*altitude_in_feet: bool = True*) → Union[float, Sequence[float]]

Parameters **altitude_in_feet** – if True, altitude is returned in feet. Otherwise, it is returned in meters

Returns altitude provided at instantiation

property delta_t: Union[float, Sequence[float]]
Temperature increment applied to whole temperature profile.

property temperature: Union[float, numpy.ndarray]
Temperature in K.

property pressure: Union[float, numpy.ndarray]
Pressure in Pa.

property density: Union[float, numpy.ndarray]
Density in kg/m3.

property speed_of_sound: Union[float, numpy.ndarray]
Speed of sound in m/s.

property dynamic_viscosity: Union[float, numpy.ndarray]
Dynamic viscosity in kg/m/s.

property kinematic_viscosity: Union[float, numpy.ndarray]
Kinematic viscosity in m2/s.

property mach: Union[float, numpy.ndarray]
Mach number.

property true_airspeed: Union[float, numpy.ndarray]
True airspeed (TAS) in m/s.

property equivalent_airspeed: Union[float, numpy.ndarray]
Equivalent airspeed (EAS) in m/s.

property unitary_reynolds: Union[float, numpy.ndarray]
Unitary Reynolds number in 1/m.

property dynamic_pressure: Union[float, numpy.ndarray]
Theoretical (true) dynamic pressure in Pa.

It is given by q = 0.5 * mach**2 * gamma * static_pressure.

property impact_pressure: Union[float, numpy.ndarray]
Compressible dynamic pressure (AKA impact pressure) in Pa.

property calibrated_airspeed: Union[float, numpy.ndarray]
Calibrated airspeed in m/s.

class stdatm.atmosphere.**AtmosphereSI**(*altitude: Union[float, Sequence[float]]*, *delta_t: float = 0.0*)
Bases: *stdatm.atmosphere.Atmosphere*

Same as *Atmosphere* except that altitudes are always in meters.

Parameters

- **altitude** – altitude in meters
- **delta_t** – temperature increment (°C) applied to whole temperature profile

property altitude
Altitude in meters.

stdatm.atmosphere_partials module

Implementation of International Standard Atmosphere with partial derivatives of state parameters with respect to altitude.

class stdatm.atmosphere_partials.**AtmosphereWithPartials**(*altitude: Union[float, Sequence[float]]*, *delta_t: float = 0.0*, *altitude_in_feet: bool = True*)
Bases: *stdatm.atmosphere.Atmosphere*

Implementation of International Standard Atmosphere for troposphere and stratosphere with derivatives of state parameters with respect to altitude.

Atmosphere properties and partials are provided in the same "shape" as provided altitude:

- if altitude is given as a float, returned values will be floats
- if altitude is given as a sequence (list, 1D numpy array, ...), returned values will be 1D numpy arrays
- if altitude is given as nD numpy array, returned values will be nD numpy arrays

The AtmosphereWithPartials class inherits from the Atmosphere class and thus retains its usages. It however adds the computation of the partial derivatives of all state properties with respect to the altitude.

Usage:

```
>>> from stdatm import AtmosphereWithPartials
>>> pressure = AtmosphereWithPartials(30000).pressure # pressure at 30,000 feet,␣
↪dISA = 0 K
>>> # pressure at 30,000 feet, dISA = 0 K
>>> partials_pressure_altitude = AtmosphereWithPartials(30000).partial_pressure_
↪altitude

>>> # init for alt. 0, 10,000 and 30,000 feet
>>> atm = AtmosphereWithPartials([0.0,10000.0,30000.0])
>>> # derivative of pressures with respect to altitude for all defined altitudes
>>> atm.partial_pressure_altitude
array([-3.66160356, -2.70401861, -1.36992549])
>>> # derivative of dynamic viscosities with respect to altitude for all defined␣
↪altitudes
```

(continues on next page)

(continued from previous page)

```
>>> atm.partial_dynamic_viscosity_altitude
array([-9.55961630e-11, -9.88873356e-11, -1.06349854e-10])
```

Parameters

- **altitude** – altitude (units decided by altitude_in_feet)
- **delta_t** – temperature increment (°C) applied to whole temperature profile
- **altitude_in_feet** – if True, altitude should be provided in feet. Otherwise, it should be provided in meters.

property partial_temperature_altitude: Union[float, numpy.ndarray]
Partial derivative of the temperature in K with respect to the altitude in the unit provided.

property partial_pressure_altitude: Union[float, numpy.ndarray]
Partial derivative of the pressure in Pa with respect to the altitude in the unit provided.

property partial_density_altitude: Union[float, numpy.ndarray]
Partial derivative of the density in kg/m**3 with respect to the altitude in the unit provided.

property partial_speed_of_sound_altitude: Union[float, numpy.ndarray]
Partial derivative of the speed of sound in m/s with respect to the altitude in the unit provided.

property partial_dynamic_viscosity_altitude: Union[float, numpy.ndarray]
Partial derivative of the dynamic viscosity in kg/m/s with respect to the altitude in the unit provided.

property partial_kinematic_viscosity_altitude: Union[float, numpy.ndarray]
Partial derivative of the kinematic viscosity in m**2/s with respect to the altitude in the unit provided.

stdatm.partials_state_parameters module

Functions for computation of the partial derivative of atmosphere state parameters.

stdatm.partials_state_parameters.**compute_partial_temperature**(*altitude*) → numpy.ndarray

Parameters **altitude** – in m

Returns Partial of temperature in K with respect to altitude in m

stdatm.partials_state_parameters.**compute_partial_pressure**(*altitude*) → numpy.ndarray

Parameters **altitude** – in m

Returns Partial of pressure in Pa with respect to altitude in m

stdatm.partials_state_parameters.**compute_partial_density**(*temperature: Union[numpy.ndarray, numbers.Real]*, *pressure: Union[numpy.ndarray, numbers.Real]*, *partial_temperature_altitude: Union[numpy.ndarray, numbers.Real]*, *partial_pressure_altitude: Union[numpy.ndarray, numbers.Real]*) → Union[numpy.ndarray, numbers.Real]

Parameters

- **temperature** – in K
- **pressure** – in Pa
- **partial_temperature_altitude** – derivative of the temperature in K with respect to the altitude
- **partial_pressure_altitude** – derivative of the pressure in Pa with respect to the altitude

Returns Partial of density in kg/m**3 with respect to altitude

stdatm.partials_state_parameters.**compute_partial_speed_of_sound**(*temperature: Union[numpy.ndarray, numbers.Real]*, *partial_temperature_altitude: Union[numpy.ndarray, numbers.Real]*) → Union[numpy.ndarray, numbers.Real]

Parameters

- **temperature** – in K
- **partial_temperature_altitude** – derivative of the temperature in K with respect to the altitude

Returns Partial of speed of sound in m/s with respect to altitude

stdatm.partials_state_parameters.**compute_partial_dynamic_viscosity**(*temperature: Union[numpy.ndarray, numbers.Real]*, *partial_temperature_altitude: Union[numpy.ndarray, numbers.Real]*) → Union[numpy.ndarray, numbers.Real]

Parameters

- **temperature** – in K
- **partial_temperature_altitude** – derivative of the temperature in K with respect to the altitude

Returns Partial of dynamic viscosity in kg/m/s with respect to altitude

stdatm.partials_state_parameters.**compute_partial_kinematic_viscosity**(*dynamic_viscosity: Union[numpy.ndarray, numbers.Real]*, *density: Union[numpy.ndarray, numbers.Real]*, *partial_dynamic_viscosity_altitude: Union[numpy.ndarray, numbers.Real]*, *partial_density_altitude: Union[numpy.ndarray, numbers.Real]*) → Union[numpy.ndarray, numbers.Real]

Parameters

- **dynamic_viscosity** – in kg/m/s
- **density** – in kg/m**3
- **partial_dynamic_viscosity_altitude** – derivative of the dynamic viscosity in kg/m/s with respect to the altitude
- **partial_density_altitude** – derivative of the density in kg/m**3 with respect to the altitude

Returns Partial of kinematic viscosity in m**2/s with respect to altitude

stdatm.speed_parameters module

Functions for computation of atmosphere speed parameters.

stdatm.speed_parameters.**compute_tas_from_mach**(*mach*, *speed_of_sound*)

Parameters

- **mach** – no unit
- **speed_of_sound** – in m/s

Returns true airspeed in m/s

stdatm.speed_parameters.**compute_tas_from_eas**(*equivalent_airspeed*, *density*)

Parameters

- **equivalent_airspeed** – in m/s
- **density** – in kg/m**3

Returns true airspeed in m/s

stdatm.speed_parameters.**compute_tas_from_unit_re**(*unitary_reynolds*, *kinematic_viscosity*)

Parameters

- **unitary_reynolds** – in 1/m
- **kinematic_viscosity** – in m**2/s

Returns true airspeed in m/s

stdatm.speed_parameters.**compute_tas_from_pdyn**(*dynamic_pressure*, *density*)

Parameters

- **dynamic_pressure** – in Pa
- **density** – in kg/m**3

Returns true airspeed in m/s

stdatm.speed_parameters.**compute_mach**(*true_airspeed*, *speed_of_sound*)

Parameters

- **true_airspeed** – in m/s
- **speed_of_sound** – in m/s

Returns Mach number (no unit)

stdatm.speed_parameters.**compute_equivalent_airspeed**(*true_airspeed*, *density*)

Parameters

- **true_airspeed** – in m/s
- **density** – in kg/m**3

Returns equivalent airspeed in m/s

stdatm.speed_parameters.**compute_unitary_reynolds**(*true_airspeed*, *kinematic_viscosity*)

Parameters

- **true_airspeed** – in m/s
- **kinematic_viscosity** – in m**2/s

Returns unitary_reynolds in 1/m

stdatm.speed_parameters.**compute_dynamic_pressure**(*true_airspeed*, *density*)

Parameters

- **true_airspeed** – in m/s
- **density** – in kg/m**3

Returns incompressible dynamic pressure in Pa

stdatm.speed_parameters.**compute_impact_pressure**(*mach*, *pressure*)

Parameters

- **mach** – no unit
- **pressure** – in Pa

Returns impact pressure in Pa

stdatm.speed_parameters.**compute_calibrated_airspeed**(*impact_pressure*)

Parameters **impact_pressure** – in Pa

Returns calibrated airspeed in m/s

stdatm.state_parameters module

Functions for computation of atmosphere state parameters.

stdatm.state_parameters.**compute_temperature**(*altitude*, *delta_t*) → numpy.ndarray

Parameters

- **altitude** – in m
- **delta_t** – in K

Returns Temperature in K

stdatm.state_parameters.**compute_pressure**(*altitude*) → numpy.ndarray

Parameters **altitude** – in m

Returns pressure in Pa

stdatm.state_parameters.**compute_density**(*pressure: Union[numpy.ndarray, numbers.Real]*, *temperature: Union[numpy.ndarray, numbers.Real]*) → Union[numpy.ndarray, numbers.Real]

Parameters

- **pressure** – in Pa
- **temperature** – in K

Returns air density in kg/m**3

stdatm.state_parameters.**compute_speed_of_sound**(*temperature: Union[numpy.ndarray, numbers.Real]*) → Union[numpy.ndarray, numbers.Real]

Parameters **temperature** – in K

Returns in m/s

stdatm.state_parameters.**compute_dynamic_viscosity**(*temperature: Union[numpy.ndarray, numbers.Real]*) → Union[numpy.ndarray, numbers.Real]

Parameters **temperature** – in K

Returns in kg/m/s

stdatm.state_parameters.**compute_kinematic_viscosity**(*dynamic_viscosity: Union[numpy.ndarray, numbers.Real]*, *density: Union[numpy.ndarray, numbers.Real]*) → Union[numpy.ndarray, numbers.Real]

Parameters

- **dynamic_viscosity** – in kg/m/s
- **density** – in kg/m**3

Returns in m**2/s

Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

M

P

S

T

U