USSA1976

Yvan Nollet

Jan 31, 2023

CONTENTS

The U.S. Standard Atmosphere 1976 model.

This package implements the atmosphere thermophysical model provided by the National Aeronautics and Space Administration technical report NASA-TM-X-74335 published in 1976 and entitled *U.S. Standard Atmosphere, 1976*.

CHAPTER

ONE

FEATURES

- Run the U.S. Standard Atmosphere 1976 model on your custom altitude grid
- **Compute all 14 atmospheric variables of the model as a function of altitude:**
  - air temperature
  - air pressure
  - number density (of individual species)
  - air number density
  - air density
  - air molar volume
  - air pressure scale height
  - air particles mean speed
  - air particles mean free path
  - air particles mean collision frequency
  - speed of sound in air
  - air dynamic viscosity
  - air kinematic viscosity
  - air thermal conductivity coefficient
- Results stored in NetCDF format
- Command-line interface
- Python interface

CHAPTER

TWO

REQUIREMENTS

- Python 3.8+

CHAPTER

THREE

INSTALLATION

You can install *USSA1976* via pip from PyPI:

```
$ pip install ussa1976
```

CHAPTER

FOUR

USAGE

- For the Command-line interface, please see the Command-line Reference for details.
- For the Python interface, refer to the User Guide.

CHAPTER

FIVE

CONTRIBUTING

Contributions are very welcome. To learn more, see the Contributor Guide.

CHAPTER

SIX

LICENSE

Distributed under the terms of the MIT license, *USSA1976* is free and open source software.

CHAPTER

SEVEN

ISSUES

If you encounter any problems, please file an issue along with a detailed description.

CHAPTER

EIGHT

CREDITS

This project was generated from @cjolowicz's Hypermodern Python Cookiecutter template.

8.1 Usage

8.1.1 ussa1976

Compute the U.S. Standard Atmosphere 1976.

```
ussa1976 [OPTIONS]
```

Options

-z, --zstart <zstart>
: Start altitude [m]

  Default
  : 0.0

-Z, --zstop <zstop>
: Stop altitude [m]

  Default
  : 1000000.0

-n, --znum <znum>
: Number of altitude points

  Default
  : 1001

-f, --filename <filename>
: Output file name

  Default
  : ussa1976.nc

--version
: Show the version and exit.

8.2 User Guide

This page presents the Python interface of `ussa1976`.

For details on how to use the command-line interface to `ussa1976`, refer to the *usage page*.

8.2.1 Getting started

Compute the U.S. Standard Atmosphere 1976 model:

```
import ussa1976

ds = ussa1976.compute()
```

The output is a `Dataset` object that tabulates the values of the different atmospheric variables as a function of altitude.

By default, the U.S. Standard Atmosphere 1976 model is computed on a piece-wise linearly spaced altitude mesh, specified in the table below.

Table 1: Default altitude mesh

Altitude range	Altitude step
$[0, 11]$ km	50 m
$[11, 32]$ km	100 m
$[32, 50]$ km	200 m
$[50, 100]$ km	500 m
$[100, 300]$ km	1000 m
$[300, 500]$ km	2000 m
$[500, 1000]$ km	5000 m

8.2.2 Inspect the output data set

You can easily access the values of the different computed variables for further manipulation.

For example, pressure values can be accessed with:

```
ds["p"].values
```

Please refer to the xarray documentation for more details.

8.2.3 Plot a variable

Note: The matplotlib library must be installed for plotting.

Plotting variables is made very convenient with xarray.

For example, the code below plots the pressure as a function of altitude:

```
import matplotlib.pyplot as plt

plt.figure(dpi=100)
```

(continues on next page)

(continued from previous page)

```
ds.p.plot(y="z", xscale="log")
plt.grid()
plt.show()
```

Please refer to the xarray documentation for more details.

8.2.4 Work with a custom altitude mesh

You can compute the U.S. Standard Atmosphere 1976 model on any altitude mesh oy your liking as long as the altitude bounds are within $[0, 1000]$ km.

For example, you can compute the model on a regular altitude mesh between 0 kilometer and 100 kilometer with a 1-meter altitude step, with:

```
import numpy as np

ds = ussa1976.compute(z=np.arange(0.0, 100001.0, 1.0))
```

Note: Altitude units are meter.

8.2.5 Compute specific variables

You might not be interested in computing all 14 variables of the U.S. Standard Atmosphere 1976 model. You can select only the variables that are relevant for your application using the `variables` parameters.

For example, to compute only the air temperature (`t`), air pressure (`p`), air number density (`n_tot`) and the species number density (`n`), use:

```
ds = ussa1976.compute(variables=["t", "p", "n_tot", "n"])
```

The table below indicates what symbol is used for each variable.

Table 2: Variables symbol

Symbol	Variable name
`t`	air temperature
`p`	air pressure
`n`	number density
`n_tot`	air number density
`rho`	air density
`mv`	air molar volume
`hp`	air pressure scale height
`v`	air particles mean speed
`mfp`	air particles mean free path
`f`	air particles mean collision frequency
`cs`	speed of sound in air
`mu`	air dynamic viscosity
`nu`	air kinematic viscosity
`kt`	air thermal conductivity coefficient

By default, all 14 variables are computed.

8.3 Reference

8.3.1 constants

Constants module.

As much as possible, constants' names are chosen to be as close as possible to the notations used in [NNU76].

Notes

Constants' values are evaluated in the following set of units: * length: meter * time: second * mass: kilogram * temperature: kelvin * quantity of matter: mole

Note the following derived units: * 1 Pa = 1 kg * m^-1 * s^-2 * 1 Joule = 1 kg * m^2 * s^-2

ussa1976.constants.**A = {'Ar': 4.487e+20, 'H': 3.305e+21, 'He': 1.7e+21, 'O': 6.986e+20, 'O2': 4.863e+20}**

Thermal diffusion coefficients [m * s^-1].

ussa1976.constants.**ALPHA = {'Ar': 0.0, 'H': -0.25, 'He': -0.4, 'N2': 0.0, 'O': 0.0, 'O2': 0.0}**

Thermal diffusion constants above 86 km [dimensionless].

ussa1976.constants.**AR_7 = 1.3514e+18**

Argon number density at altitude Z7 [m^-3].

ussa1976.constants.**B = {'Ar': 0.87, 'H': 0.5, 'He': 0.691, 'O': 0.75, 'O2': 0.75}**

Thermal diffusion constants [dimensionless].

ussa1976.constants.**BETA = 1458000.0**

β constant in eq. 51 of [NNU76] [kg * m^-1 * s^-1 * K^-0.5].

ussa1976.constants.**F = {'Ar': 0.00934, 'CH4': 2e-06, 'CO2': 0.000314, 'H2': 5e-07, 'He': 5.24e-06, 'Kr': 1.14e-06, 'N2': 0.78084, 'Ne': 1.818e-05, 'O2': 0.209476, 'Xe': 8.7e-08}**

Sea level volume fractions below 86 km [dimensionless].

ussa1976.constants.**G0 = 9.80665**

Sea level gravity [m / s^-2].

ussa1976.constants.**GAMMA = 1.4**

Ratio of specific heat of air at constant pressure to the specific heat of air at constant volume [dimensionless].

ussa1976.constants.**H: ndarray[Any, dtype[float64]] = array([0. , 11000. , 20000. , 32000. , 47000. , 51000. , 71000. , 84852.05])**

Geopotential altitudes of the layers' boundaries (below 86 km) [m].

ussa1976.constants.**HE_7 = 7581700000000000.0**

Helium number density at altitude Z7 [m^-3].

Notes

Assumes typo at page 13.

ussa1976.constants.**H_11 = 80000000000.0**

Hydrogen number density at altitude Z7 [m^-3].

ussa1976.constants.**K = 1.380622e-23**

Boltzmann constant [J * K^-1].

ussa1976.constants.**K_7 = 120.0**

Eddy diffusion coefficients [m^2 * s^-1].

ussa1976.constants.**LAMBDA = 1.875e-05**

λ constant in eq. 32 of [NNU76] [m^-1].

ussa1976.constants.**LK: ndarray[Any, dtype[float64]] = array([-0.0065, 0. , 0.001 , 0.0028, 0. , -0.0028, -0.002])**

Temperature gradients in the seven layers (below 86 km) [K * m^-1].

ussa1976.constants.**LK7 = 0.0**

Temperature gradient in the 8th layer [K * m^-1].

ussa1976.constants.**LK9 = 0.012**

Temperature gradient in the 10th layer [K * m^-1].

ussa1976.constants.**M = {'Ar': 0.039948, 'CH4': 0.01604303, 'CO2': 0.04400995, 'H': 0.00100797, 'H2': 0.00201594, 'He': 0.0040026, 'Kr': 0.0838, 'N2': 0.0280134, 'Ne': 0.020183, 'O': 0.01599939, 'O2': 0.0319988, 'Xe': 0.1313}**

Molar masses of the individual species [kg * mole^-1].

ussa1976.constants.**M0 = 0.028964425278793997**

Sea level mean air molar mass [kg * mole^-1].

ussa1976.constants.**N2_7 = 1.129794e+20**

Molecular nitrogen number density at altitude Z7 [m^-3].

ussa1976.constants.**NA = 6.022169e+23**

Avogadro number [mole^-1].

ussa1976.constants.**O2_7 = 3.030898e+19**

Molecular oxygen number density at altitude Z7 [m^-3].

ussa1976.constants.**O_7 = 8.6e+16**

Atomic oxygen number density at altitude Z7 [m^-3].

ussa1976.constants.**P0 = 101325.0**

Pressure at sea level [Pa].

ussa1976.constants.**PHI = 720000000000.0**

Vertical air particles flux [m^2 * s^-1].

ussa1976.constants.**Q1 = {'Ar': 9.434079e-14, 'He': -2.457369e-13, 'O': -5.809644e-13, 'O2': 1.366212e-13}**

Vertical transport constants above 86 km [m^-3].

ussa1976.constants.**Q2 = {'Ar': 0.0, 'He': 0.0, 'O': -3.416248e-12, 'O2': 0.0}**

Vertical transport constants above 86 km [m^-3].

ussa1976.constants.**R = 8.31432**
Universal gas constant [J * K^-1 * mole^-1].

ussa1976.constants.**R0 = 6356766.0**
Effective Earth radius [m].

ussa1976.constants.**S = 110.4**
Sutherland constant in eq. 51 of [NNU76] [K].

ussa1976.constants.**SIGMA = 3.65e-10**
Mean effective collision diameter [m].

ussa1976.constants.**T0 = 288.15**
Temperature at sea level [K].

ussa1976.constants.**T10 = 360.0**
Temperature at altitude Z10 [K].

ussa1976.constants.**T11 = 999.2356**
Temperature at altitude Z11 [K].

ussa1976.constants.**T7 = 186.8673**
Temperature at altitude Z7 [K].

ussa1976.constants.**T9 = 240.0**
Temperature at altitude Z9 [K].

ussa1976.constants.**TINF = 1000.0**
Exospheric temperature [K].

ussa1976.constants.**U1 = {'Ar': 86000.0, 'He': 86000.0, 'O': 56903.11, 'O2': 86000.0}**
Vertical transport constants above 86 km [m].

ussa1976.constants.**U2 = {'O': 97000.0}**
Vertical transport constants above 86 km [m].

ussa1976.constants.**W1 = {'Ar': 8.333333e-14, 'He': 6.666667e-13, 'O': 2.70624e-14, 'O2': 8.333333e-14}**
Vertical transport constants above 86 km [m^-3].

ussa1976.constants.**W2 = {'O': 5.008765e-13}**
Vertical transport constants above 86 km [m^-3].

ussa1976.constants.**Z10 = 120000.0**
Top altitude of the 10th layer [m].

ussa1976.constants.**Z12 = 1000000.0**
Top altitude of the 12nd layer [m].

ussa1976.constants.**Z7 = 86000.0**
Top altitude of the 7th layer [m].

ussa1976.constants.**Z8 = 91000.0**
Top altitude of the 8th layer [m].

ussa1976.constants.**Z9 = 110000.0**
Top altitude of the 9th layer [m].

8.3.2 core

U.S. Standard Atmosphere 1976 thermophysical model.

The U.S. Standard Atmosphere 1976 model [NNU76] divides the atmosphere into two altitude regions:

1. the low-altitude region, from 0 to 86 kilometers
2. the high-altitude region, from 86 to 1000 kilometers.

A number of computational functions hereafter are specialised for one or the other altitude region and is valid only in that altitude region, not in the other. Their name include a `low_altitude` or a `high_altitude` part to reflect that they are valid only in the low altitude region and high altitude region, respectively.

ussa1976.core.**compute**(*z=array([0.00e+00, 5.00e+01, 1.00e+02, ..., 9.90e+05, 9.95e+05, 1.00e+06])*, *variables=None*)

Compute U.S. Standard Atmosphere 1976 data set on specified altitude grid.

Parameters

- **z** (ndarray) – Altitude [m].
- **variables** (list, *optional*) – Names of the variables to compute.

Returns
Data set holding the values of the different atmospheric variables.

Return type
Dataset

Raises
ValueError – When altitude is out of bounds, or when variables are invalid.

ussa1976.core.**compute_gravity**(*z*)

Compute gravity.

Parameters
z (ndarray) – Altitude [m].

Returns
Gravity [m * s^-2].

Return type
ndarray

ussa1976.core.**compute_high_altitude**(*data_set*, *mask=None*, *inplace=False*)

Compute U.S. Standard Atmosphere 1976 in high-altitude region.

Parameters

- **data_set** (Dataset) – Data set to compute.
- **mask** (DataArray, *optional*) – Mask to select the region of the data set to compute. By default, the mask selects the entire data set.
- **inplace** (bool, *default* False) – If True, modifies data_set in place, else returns a copy of data_set.

Returns
If inplace is True, returns nothing, else returns a copy of data_set.

Return type
Dataset

ussa1976.core.**compute_levels_temperature_and_pressure_low_altitude**()

Compute temperature and pressure at low-altitude region' levels.

Returns
Levels temperatures [K] and pressures [Pa].

Return type
tuple of arrays

ussa1976.core.**compute_low_altitude**(*data_set*, *mask=None*, *inplace=False*)

Compute U.S. Standard Atmosphere 1976 in low-altitude region.

Parameters

- **data_set** (Dataset) – Data set to compute.
- **mask** (DataArray, *optional*) – Mask to select the region of the data set to compute. By default, the mask selects the entire data set.
- **inplace** (bool, *default* False) – If True, modifies data_set in place, else returns a copy of data_set.

Returns
If inplace is True, returns nothing, else returns a copy of data_set.

Return type
Dataset

ussa1976.core.**compute_mean_molar_mass_high_altitude**(*z*)

Compute mean molar mass in high-altitude region.

Parameters
z (ndarray) – Altitude [m].

Returns
Mean molar mass [kg/mole].

Return type
ndarray

ussa1976.core.**compute_number_densities_high_altitude**(*altitudes*)

Compute number density of individual species in high-altitude region.

Parameters
altitudes (ndarray) – Altitudes [m].

Returns
Number densities of the individual species and total number density at the given altitudes.

Return type
DataArray

Notes

A uniform altitude grid is generated and used for the computation of the integral as well as for the computation of the number densities of the individual species. This gridded data is then interpolated at the query `altitudes` using a linear interpolation scheme in logarithmic space.

ussa1976.core.**compute_pressure_low_altitude**(*h*, *pb*, *tb*)

Compute pressure in low-altitude region.

Parameters

- **h** (ndarray) – Geopotential height [m].
- **pb** (ndarray) – Levels pressure [Pa].
- **tb** (ndarray) – Levels temperature [K].

Returns
Pressure [Pa].

Return type
ndarray

ussa1976.core.**compute_pressure_low_altitude_non_zero_gradient**(*h*, *hb*, *pb*, *tb*, *lkb*)

Compute pressure in low-altitude non-zero temperature gradient region.

Parameters

- **h** (ndarray) – Geopotential height [m].
- **hb** (float) – Geopotential height at the bottom of the layer [m].
- **pb** (float) – Pressure at the bottom of the layer [Pa].
- **tb** (float) – Temperature at the bottom of the layer [K].
- **lkb** (float) – Temperature gradient in the layer [K * m^-1].

Returns
Pressure [Pa].

Return type
ndarray

ussa1976.core.**compute_pressure_low_altitude_zero_gradient**(*h*, *hb*, *pb*, *tb*)

Compute pressure in low-altitude zero temperature gradient region.

Parameters

- **h** (ndarray) – Geopotential height [m].
- **hb** (float) – Geopotential height at the bottom of the layer [m].
- **pb** (float) – Pressure at the bottom of the layer [Pa].
- **tb** (float) – Temperature at the bottom of the layer [K].

Returns
Pressure [Pa].

Return type
ndarray

ussa1976.core.**compute_temperature_gradient_high_altitude**(*z*)

Compute temperature gradient in high-altitude region.

Parameters
z (ndarray) – Altitude [m].

Returns
Temperature gradient [K/m].

Return type
ndarray

ussa1976.core.**compute_temperature_high_altitude**(*z*)

Compute temperature in high-altitude region.

Parameters
z (ndarray) – Altitude [m].

Returns
Temperature [K].

Return type
ndarray

ussa1976.core.**compute_temperature_low_altitude**(*h*, *tb*)

Compute temperature in low-altitude region.

Parameters

- **h** (ndarray) – Geopotential height [m].
- **tb** (ndarray) – Levels temperature [K].

Returns
Temperature [K].

Return type
ndarray

ussa1976.core.**eddy_diffusion_coefficient**(*z*)

Compute Eddy diffusion coefficient in high-altitude region.

Parameters
z (ndarray) – Altitude [m].

Returns
Eddy diffusion coefficient [m^2 * s^-1].

Return type
ndarray

Notes

Valid in the altitude region $86 \leq z \leq 150$ km.

ussa1976.core.**f_above_115_km**(*g*, *t*, *dt_dz*, *mi*, *alpha*)

Evaluate function f above 115 km altitude.

Evaluate the function f defined by equation (36) in [NNU76] in the altitude region $115\ z \leq 1000$ km.

Parameters

- **g** (ndarray) – Gravity at the different altitudes [m * s^-2].

- **t** (ndarray) – Temperature at the different altitudes [K].
- **dt_dz** (ndarray) – Temperature gradient at the different altitudes [K * m^-1].
- **mi** (float) – Species molar masses [kg * mole^-1].
- **alpha** (float) – Alpha thermal diffusion constant [dimensionless].

Returns
Function f at the different altitudes.

Return type
ndarray

ussa1976.core.**f_below_115_km**(*g*, *t*, *dt_dz*, *m*, *mi*, *alpha*, *d*, *k*)

Evaluate function f below 115 km altitude.

Evaluates the function f defined by equation (36) in [NNU76] in the altitude region 86 km $\leq z \leq$ 115 km.

Parameters

- **g** (ndarray) – Gravity values at the different altitudes [m * s^-2].
- **t** (ndarray) – Temperature values at the different altitudes [K].
- **dt_dz** (ndarray) – Temperature gradient values at the different altitudes [K * m^-1].
- **m** (ndarray) – Molar mass [kg * mole^-1].
- **mi** (float) – Species molar masses [kg * mole^-1].
- **alpha** (float) – Alpha thermal diffusion constant [dimensionless].
- **d** (ndarray) – Thermal diffusion coefficient values at the different altitudes [m^2 * s^-1].
- **k** (ndarray) – Eddy diffusion coefficient values at the different altitudes [m^2 * s^-1].

Returns
Function f at the different altitudes.

Return type
ndarray

ussa1976.core.**init_data_set**(*z*)

Initialise data set.

Parameters
z (ndarray) – Altitudes [m].

Returns
Initialised data set.

Return type
Dataset

ussa1976.core.**log_interp1d**(*x*, *y*)

Compute linear interpolation of $y(x)$ in logarithmic space.

Parameters

- **x** (ndarray) – 1-D array of real values.
- **y** (ndarray) – N-D array of real values. The length of y along the interpolation axis must be equal to the length of x.

Returns
Interpolating function.

Return type
callable()

ussa1976.core.**tau_function**(*z_grid*, *below_500=True*)

Compute τ function.

Compute integral given by equation (40) in [NNU76] at each point of an altitude grid.

Parameters

- **z_grid** (ndarray) – Altitude grid (values sorted by ascending order) to use for integration [m].
- **below_500** (bool, *default* True) – True if altitudes in z_grid are lower than 500 km, False otherwise.

Returns
Integral evaluations [dimensionless].

Return type
ndarray

Notes

Valid for 150 km $\leq z \leq$ 500 km.

ussa1976.core.**thermal_diffusion_coefficient**(*nb*, *t*, *a*, *b*)

Compute thermal diffusion coefficient values in high-altitude region.

Parameters

- **nb** (ndarray) – Background number density [m^-3].
- **t** (ndarray) – Temperature [K].
- **a** (float) – Thermal diffusion constant a [m^-1 * s^-1].
- **b** (float) – Thermal diffusion constant b [dimensionless].

Returns
Thermal diffusion coefficient [m^2 * s^-1].

Return type
ndarray

ussa1976.core.**thermal_diffusion_term**(*s*, *z_grid*, *g*, *t*, *dt_dz*, *m*, *d*, *k*)

Compute thermal diffusion term of given species in high-altitude region.

Parameters

- **s** (str) – Species.
- **z_grid** (ndarray) – Altitude grid [m].
- **g** (ndarray) – Gravity values on the altitude grid [m * s^-2].
- **t** (ndarray) – Temperature values on the altitude grid [K].
- **dt_dz** (ndarray) – Temperature gradient values on the altitude grid [K * m^-1].
- **m** (ndarray) – Values of the mean molar mass on the altitude grid [kg * mole^-1].
- **d** (ndarray) – Molecular diffusion coefficient values on the altitude grid, for altitudes strictly less than 115 km [m^2 * s^-1].

- **k** (ndarray) – Eddy diffusion coefficient values on the altitude grid, for altitudes strictly less than 115 km [m^2 * s^-1].

Returns
Thermal diffusion term [m^-1].

Return type
ndarray

ussa1976.core.**thermal_diffusion_term_atomic_oxygen**(*z_grid*, *g*, *t*, *dt_dz*, *d*, *k*)

Compute oxygen thermal diffusion term in high-altitude region.

Parameters

- **z_grid** (ndarray) – Altitude grid [m].
- **g** (ndarray) – Gravity values on the altitude grid [m * s^-2].
- **t** (ndarray) – Temperature values on the altitude grid [K].
- **dt_dz** (ndarray) – Temperature values gradient on the altitude grid [K * m^-1].
- **d** (ndarray) – Thermal diffusion coefficient on the altitude grid [m^2 * s^-1].
- **k** (ndarray) – Eddy diffusion coefficient values on the altitude grid [m^2 * s^-1].

Returns
Thermal diffusion term [-1].

Return type
ndarray

ussa1976.core.**to_altitude**(*h*)

Convert geopotential height to (geometric) altitude.

Parameters
h (ndarray) – Geopotential altitude [m].

Returns
Altitude [m].

Return type
ndarray

ussa1976.core.**to_geopotential_height**(*z*)

Convert altitude to geopotential height.

Parameters
z (ndarray) – Altitude [m].

Returns
Geopotential height [m].

Return type
ndarray

ussa1976.core.**velocity_term**(*s*, *z_grid*)

Compute velocity term of a given species in high-altitude region.

Parameters

- **s** (str) – Species.
- **z_grid** (ndarray) – Altitude grid [m].

Returns
Velocity term [m^-1].

Return type
ndarray

Notes

Not valid for atomic oxygen. See *velocity_term_atomic_oxygen()*.

ussa1976.core.**velocity_term_atomic_oxygen**(*grid*)

Compute velocity term of atomic oxygen in high-altitude region.

Parameters
grid (ndarray) – Altitude grid [m].

Returns
Velocity term [m^-1].

Return type
ndarray

ussa1976.core.**velocity_term_hump**(*z*, *q1*, *q2*, *u1*, *u2*, *w1*, *w2*)

Compute transport term.

Compute the transport term given by equation (37) in [NNU76].

Parameters

- **z** (ndarray) – Altitude [m].
- **q1** (float) – Q constant [m^-3].
- **q2** (float) – q constant [m^-3].
- **u1** (float) – U constant [m].
- **u2** (float) – u constant [m].
- **w1** (float) – W constant [m^-3].
- **w2** (float) – w constant [m^-3].

Returns
Transport term [m^-1].

Return type
ndarray

Notes

Valid in the altitude region: 86 km $\leq z \leq$ 150 km.

ussa1976.core.**velocity_term_no_hump**(*z*, *q1*, *u1*, *w1*)

Compute transport term.

Compute the transport term given by equation (37) in [NNU76] where the second term is zero.

Parameters

- **z** (ndarray) – Altitude.
- **q1** (float) – Q constant [m^-3].

- **u1** (float) – U constant [m].
- **w1** (float) – W constant [m^-3].

Returns
Transport term [m^-1].

Return type
ndarray

Notes

Valid in the altitude region $86 \text{ km} \leq z \leq 150 \text{ km}$.

8.3.3 ussa1976

Command-line interface.

See also:

Command-line interface usage page

8.4 Contributor Guide

Thank you for your interest in improving this project. This project is open-source under the MIT license and welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- Source Code
- Documentation
- Issue Tracker
- Code of Conduct

8.4.1 How to report a bug

Report bugs on the Issue Tracker.

When filing an issue, make sure to answer these questions:

- Which operating system and Python version are you using?
- Which version of this project are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

8.4.2 How to request a feature

Request features on the Issue Tracker.

8.4.3 How to set up your development environment

You need Python 3.8+ and the following tools:

- Poetry
- Nox
- nox-poetry

Install the package with development requirements:

```
$ poetry install
```

You can now run an interactive Python session, or the command-line interface:

```
$ poetry run python
$ poetry run ussa1976
```

8.4.4 How to test the project

Run the full test suite:

```
$ nox
```

List the available Nox sessions:

```
$ nox --list-sessions
```

You can also run a specific Nox session. For example, invoke the unit test suite like this:

```
$ nox --session=tests
```

Unit tests are located in the `tests` directory, and are written using the pytest testing framework.

8.4.5 How to submit changes

Open a pull request to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests. This project maintains 100% code coverage.
- If your changes add functionality, update the documentation accordingly.

Feel free to submit early, though—we can always iterate on this.

To run linting and code formatting checks before commiting your change, you can install pre-commit as a Git hook by running the following command:

```
$ nox --session=pre-commit -- install
```

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

8.5 Contributor Covenant Code of Conduct

8.5.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

8.5.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

8.5.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

8.5.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

8.5.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at yvan.nollet@rayference.eu. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

8.5.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

8.5.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

8.6 MIT License

BIBLIOGRAPHY

[NNU76] NASA, NOAA, and USAF. U.S. Standard Atmosphere, 1976. techreport NASA-TM-X-74335, National Aeronautics and Space Administration, 1976. URL: https://ntrs.nasa.gov/search.jsp?R=19770009539 (visited on 2019-10-25).

PYTHON MODULE INDEX

u

INDEX

Symbols

A

B

C

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z