Matrix Cascade Equation (MCEq) Documentation

Release 1.2.6

Anatoli Fedynitch

Jul 04, 2022

Contents

Release 1.2.6

Date Jul 04, 2022

Purpose of the code:

This program is a toolkit to compute the evolution of particle densities that evolve as a cascade in the Earth's atmosphere or other target media. Particles are represented by average densities on discrete energy bins. The results are differential energy spectra or total particle numbers. Various models/parameterizations for particle interactions and atmospheric density profiles are packaged with the code.

CHAPTER 1

Installation

The installation via PyPi is the simplest method:

```
pip install MCEq
```

Optionally, one can (and is encouraged to) accelerate calculations with BLAS libraries. For Intel CPUs the Math Kernel Library (MKL) provides quite some speed-up compared to plain numpy. MCEq will auto-detect MKL if it is already installed. To enable MKL by default use:

```
pip install MCEq[MKL]
```

More speed-up can be achieved by using the cuSPARSE library from nVidia's CUDA toolkit. This requires the cupy library. If cupy is detected, MCEq will try to use cuSPARSE as solver. To install MCEq with CUDA 10.1 support:

```
pip install MCEq[CUDA]
```

Alternatively, install cupy by yourself (see **'cupy homepage<https://cupy.chainer.org>'_**).

Supported architectures:

- Linux 32- and 64-bit (x86_64 and AArch64)
- Mac OS X
- Windows

Note:: pip installations of scipy on Windows may be faulty. If scipy throws errors on import, use Miniconda

CHAPTER 2

Upgrading

For installations with pip, upgrading the code and data tables can be done with:

```
pip install MCEq --upgrade
```

In case of major updates the database file will be updated on first import and the old one will be removed. For installations from source, pull the latest release or master branch. The database file will be updated automatically as well.

CHAPTER 3

Building MCEq from source

To modify the code and contribute, the code needs to be installed from the github source:

```
git clone https://github.com/afedynitch/MCEq.git
cd MCEq
pip install -e .
```

This will build and install MCEq in editable mode, and changes to this source directory will be immediately reflected in any code that imports MCEq in the current python enviroment.

CHAPTER 4

Quick start

Open an new python file or jupyter notebook/lab:

```
from MCEq.core import config, MCEqRun
import crflux.models as crf
# matplotlib used plotting. Not required to run the code.
import matplotlib.pyplot as plt


# Initalize MCEq by creating the user interface object MCEqRun
mceq = MCEqRun(

    # High-energy hadronic interaction model
    interaction_model='SIBYLL23C',

    # cosmic ray flux at the top of the atmosphere
    primary_model = (crf.HillasGaisser2012, 'H3a'),

    # zenith angle
    theta_deg = 0.
)

# Solve the equation system
mceq.solve()

# Obtain the result
# Multiply fluxes be E**mag to resolve the features of the steep spectrum
mag = 3
muon_flux = (mceq.get_solution('mu+', mag) +
             mceq.get_solution('mu-', mag))
numu_flux = (mceq.get_solution('numu', mag) +
             mceq.get_solution('antinumu', mag))
nue_flux = (mceq.get_solution('nue', mag) +
            mceq.get_solution('antinue', mag))
```

(continues on next page)

(continued from previous page)

```
# The lines below are for plotting with matplotlib
plt.loglog(mceq.e_grid, muon_flux, label='muons')
plt.loglog(mceq.e_grid, numu_flux, label='muon neutrinos')
plt.loglog(mceq.e_grid, nue_flux, label='electron neutrinos')

plt.xlim(1., 1e9)
plt.xlabel('Kinetic energy (GeV)')
plt.ylim(1e-6, 1.)
plt.ylabel(r'$(E/\text{GeV})^3\,\Phi$ (GeV cm$^{-2}$\,$s$^{-1}\,$sr$^{-1}$) (GeV)')
plt.legend()
plt.show()
```

CHAPTER 5

Examples

Follow the *Tutorial* and/or download and run the notebooks from github.

CHAPTER 6

Citations

If you use MCEq in your scientific publication, please cite the code **AND** the physical models.

The current citation for the MCEq is:

> *Calculation of conventional and prompt lepton fluxes at very high energy*
> A. Fedynitch, R. Engel, T. K. Gaisser, F. Riehn, T. Stanev,
> EPJ Web Conf. 99 (2015) 08001
> arXiv:1503.00544

In *References* is the list for the physical models.

CHAPTER 7

Changes to physics between version 1.1.X and 1.2.X

Due to a bug found in the decay table generation for the "new" MCEq versions there are some larger changes for the lowest energies at tens of GeV. To estimate if your computations may be affected check out *Differences between V 1.2.0 and 1.1.3*.

CHAPTER 8

Main documentation

8.1 Tutorial

The main user interface is the class *MCEq.core.MCEqRun* that requires a reference to a cosmic ray model for the initialization. Any cosmic ray flux model from the crflux package. can be selected:

```
from MCEq.core import MCEqRun
import crflux.models as crf

# Initalize MCEq by creating the user interface object MCEqRun
mceq = MCEqRun(

    # High-energy hadronic interaction model
    interaction_model='SIBYLL23C',

    # cosmic ray flux at the top of the atmosphere
    primary_model = (crf.HillasGaisser2012, 'H3a'),

    # zenith angle
    theta_deg = 0.
)
```

The code will raise an exception of a non-existent hadronic interaction model is selected and will list the currently available models. All models can be changed between calls to the solver.

8.1.1 Solving cascade equations

The solver is launched for the current set of parameters by:

```
mceq.solve()
```

By default MCEq will pick the numpy, MKL or the CUDA solver, depending on the the installed packages. Currently only 'forward-euler' solvers are available, which are fast and stable enough.

The spectrum of each particle species at the surface can be retrieved as numpy array with

```
mceq.get_solution('mu+')
```

List available particle species managed by *MCEq.particlemanager*:

```
mceq.pman.print_particle_tables(0)
```

To multiply the solution automatically with E^{mag} use

```
mceq.get_solution('mu+', mag=3) # for E^3 * flux
```

To obtain a solution along the cascade trajectory in depth X, create a grid and pass it to the solver

```
# A linearly spaced set of points from 0.1 up to the X value corresponding
# to the depth at the surface `max_X` (for the selected zenith angle and atmospheric␣
↪model/season)
n_pts = 100
X_grid = np.linspace(0.1, mceq.density_model.max_X, n_pts)

mceq.solve(int_grid=X_grid)
```

To obtain particle spectra at each depth point:

```
longitudinal_spectrum = []
for idx in range(n_pts):
    print('Reading solution at X = {0:5.2f} g/cm2'.format(x_grid[idx]))
    longitudinal_spectrum.append(mceq.get_solution('mu+', grid_idx=idx))
```

To obtain the solutions at equivalent altitudes one needs to simply map the the values of X to the corresponding altitude for the **current** zenith angle and atmospheric model:

```
h_grid = mceq.density_model.X2h(X_grid)
```

To define a strictly increasing grid in X (=stricktly decreasing in altitude), using the converter function between height and depth:

```
h_grid = np.linspace(50 * 1e3 * 1e2, 0) # altitudes from 50 to 0 km (in cm)
X_grid = mceq.density_model.h2X(h_grid)

mceq.solve(int_grid=X_grid)
```

Particle numbers can be obtained by using predefined functions or by integrating the spectrum. These functions support *grid_idx* (as shown above) and a minimal energy cutoff (larger than the minimal grid energy *mceq_config.e_min*):

```
# Number of muons
n_mu = mceq.n_mu(grid_idx=None, min_energy_cutoff=1e-1)

# Number of electrons
n_e = mceq.n_e(grid_idx=None, min_energy_cutoff=86e-3)

# Number of protons above minimal grid energy
n_p = np.sum(mceq.get_solution('p+', integrate=True))
```

All particles listed by `MCEq.ParticleManager.print_particle_tables(0)()` are available to `MCEq.core.get_solution()`.

8.1.2 Changing geometrical and atmospheric parameters

To change the zenith angle

```
mceq.set_theta_deg(<zenith_angle_in_degrees>)
```

Most geometries support angles between 0 (vertical) and 90 degrees.

To change the density profile

```
mceq.set_density_model(('MSIS00', ('Sudbury', 'June')))
```

Available models are:

- 'CORSIKA' - Linsley-parameterizations from the CORSIKA air-shower MC (see `MCEq.geometry.density_models.CorsikaAtmosphere.init_parameters()`)
- 'MSIS00' and 'MSIS00_IC' - NRLMSISE-00 global static atmospheric model by NASA (_IC = centered on IceCube at the South Pole, where zenith angles > 90 degrees are up-going)
- 'AIRS' - an interface to tabulated satellite data (not provided), extrapolated with MSIS00 at altitudes above 50km
- 'Isothermal' - a simple isothermal model with scale height at 6.3 km
- 'GeneralizedTarget' - a piece-wise homogeneous density (not exponential like the atmosphere)

Refer for more info to *Geometry package*.

After changing the models, the spectra can be recomputed with a *`MCEq.core.MCEqRun.solve()`*.

8.1.3 Changing hadronic interaction models

To change the hadronic interaction model

```
mceq.set_interaction_model('EPOS-LHC')
```

Currently available models are:

- SIBYLL-2.3c
- SIBYLL-2.3
- SIBYLL-2.1
- EPOS-LHC
- QGSJet-II-04
- QGSJet-II-03
- QGSJet-01c
- DPMJET-III-3.0.6
- DPMJET-III-19.1
- SIBYLL-2.3c_pp (for proton-proton collisions)

More models planned. Note that internally the model name string is transformed to upper case, and dashes and points are removed.

MCEq will take care of updating all data structures regenerating the matrices. This call takes some time since data memory needs to be allocated and some numbers crunched. If you use this function in a loop for multiple computations, put it further out.

8.1.4 Changing cosmic ray flux model

The flux of cosmic ray nucleons at the top of the atmosphere (primary flux) is the initial condition. The module `crflux.models` contains a contemporary selection of flux models. Refer to the crflux documentation or the source code.

To change the primary flux use *MCEq.core.MCEqRun.set_primary_model()*

```
import crflux.models as pm

mceq.set_primary_model(pm.HillasGaisser2012, 'H3a')
```

8.1.5 Using MCEq for air-showers

MCEq currently provides solutions of the one-dimensional (longitudinal) cascade equations in the variable X (depth). Therefore, full air-shower calculations including the lateral (transverse) extension of particle densities are not possible. What is possible is the computation of longitudinal profiles of particle numbers or depth dependence of spectra. The only difference between "air-shower mode" and the standard "inclusive flux modes" is the initial condition. For air-showers the initial condition is a single particle of a certain type and fixed energy, instead of an entire spectrum of cosmic ray nucleons as described above. To launch a cascade from a single particle use *MCEq.core.MCEqRun.set_single_primary_particle()*

```
# For a 1 EeV proton
mceq.set_single_primary_particle(1e9, pdg_id=2212)

# Or for a 1 EeV iron nucleus
mceq.set_single_primary_particle(1e9, corsika_id=5626)
```

The zenith angle has to be set as shown above with `MCEq.core.MCEqRun.set_zenith_deg()`.

8.2 References

8.2.1 Cosmic-ray flux parameterizations

The parameterizations of the flux of cosmic rays are provided via the module crflux. The references are given in the module's documentation.

8.2.2 Atmosphere

- **CORSIKA parametrizations**

  CORSIKA: A Monte Carlo Code to Simulate Extensive Air Showers
  D. Heck, J. Knapp, J. Capdevielle, G. Schatz, T. Thouw
  Tech. Rep. FZKA 6019, Karsruhe (1998)

- **NRLMSISE-00**

  NRLMSISE-00 empirical model of the atmosphere: Statistical comparisons and scientific issues

J.M. Picone, A.E. Hedin, D.P. Drob, and A.C. Aikin
J. Geophys. Res., 107(A12), 1468, doi:10.1029/2002JA009430, (2002)

8.2.3 Hadronic interaction models

- **SIBYLL-2.3c:**

  The hadronic interaction model Sibyll 2.3c and Feynman scaling
  F. Riehn, R. Engel, A. Fedynitch, T. K. Gaisser, T. Stanev
  PoS(ICRC2017)301

- **SIBYLL-2.3:**

  A new version of the event generator Sibyll
  F. Riehn, R. Engel, A. Fedynitch, T. K. Gaisser, T. Stanev
  PoS ICRC2015 (2016) 558

- **SIBYLL-2.1:**

  Cosmic ray interaction event generator SIBYLL 2.1
  E.-J. Ahn, R. Engel, T. K. Gaisser, P. Lipari, T.Stanev
  Phys.Rev. D80 (2009) 094003

- **EPOS-LHC:**

  EPOS LHC : test of collective hadronization with LHC data
  T. Pierog, I. Karpenko, J. M. Katzy, E. Yatsenko, and K. Werner
  Phys. Rev. C92 (2015) 034906
  arXiv:1306.0121

- **QGSJET-II-04:**

  Monte Carlo treatment of hadronic interactions in enhanced Pomeron scheme: I. QGSJET-II model
  Sergey Ostapchenko
  Phys.Rev. D83 (2011) 014018,

- **DPMJET-III:**

  The Monte Carlo event generator DPMJET-III
  S. Roesler, R. Engel, J. Ranft
  Advanced Monte Carlo for radiation physics, particle transport simulation and applications. Proceedings, Conference, MC2000, Lisbon, Portugal, October 23-26, 2000

- **DPMJET-III-19.1:**

  Cascade equations and hadronic interactions at very high energies
  A. Fedynitch
  CERN-THESIS-2015-371

8.3 Geometry package

In MCEq, geometry is everything related to the medium in which the particle cascade develops. The very basic geometrical functions for the polar coordinate system of the Earth - no it's not flat, but just azimuth symmetric - are located in `MCEq.geometry.geometry`. The density parameterizations and interfaces are in `MCEq.geometry.density_profiles`

8.3.1 MCEq.geometry.density_profiles

This module includes classes and functions modeling the Earth's atmosphere. Currently, two different types models are supported:

- Linsley-type/CORSIKA-style parameterization
- Numerical atmosphere via external routine (NRLMSISE-00)

Both implementations have to inherit from the abstract class `EarthsAtmosphere`, which provides the functions for other parts of the program. In particular the function `EarthsAtmosphere.get_density()`

Typical interaction:

```
$ atm_object = CorsikaAtmosphere("BK_USStd")
$ atm_object.set_theta(90)
$ print 'density at X=100', atm_object.X2rho(100.)
```

The class `MCEqRun` will only the following routines::

- `EarthsAtmosphere.set_theta()`,
- `EarthsAtmosphere.r_X2rho()`.

If you are extending this module make sure to provide these functions without breaking compatibility.

Example: An example can be run by executing the module:

```
$ python MCEq/atmospheres.py
```

class MCEq.geometry.density_profiles.**AIRSAtmosphere**(*location*, *season*, *extrapolate=True*, **args*, ***kwargs*)

Interpolation class for tabulated atmospheres.

This class is intended to read preprocessed AIRS Satellite data.

Parameters

- **location** (*str*) – see *init_parameters()*
- **season** (*str,optional*) – see *init_parameters()*

get_density(*h_cm*)

Returns the density of air in g/cm**3.

Interpolates table at requested value for previously set year and day of year (doy).

Parameters **h_cm** (*float*) – height in cm

Returns density $\rho(h_{cm})$ in g/cm**3

Return type float

get_temperature(*h_cm*)

Returns the temperature in K.

Interpolates table at requested value for previously set year and day of year (doy).

Parameters **h_cm** (*float*) – height in cm

Returns temperature $T(h_{cm})$ in K

Return type float

init_parameters(*location*, ***kwargs*)

Loads tables and prepares interpolation.

Parameters

- **location** (*str*) – supported is only "SouthPole"
- **doy** (*int*) – Day Of Year

class MCEq.geometry.density_profiles.**CorsikaAtmosphere**(*location*, *season=None*)

Class, holding the parameters of a Linsley type parameterization similar to the Air-Shower Monte Carlo CORSIKA.

The parameters pre-defined parameters are taken from the CORSIKA manual. If new sets of parameters are added to *init_parameters()*, the array _thickl can be calculated using *calc_thickl()* .

_atm_param

(5x5) Stores 5 atmospheric parameters _aatm, _batm, _catm, _thickl, _hlay for each of the 5 layers

Type numpy.array

Parameters

- **location** (*str*) – see *init_parameters()*
- **season** (*str,optional*) – see *init_parameters()*

calc_thickl()

Calculates thickness layers for *depth2height()*

The analytical inversion of the CORSIKA parameterization relies on the knowledge about the depth X, where trasitions between layers/exponentials occur.

Example

Create a new set of parameters in *init_parameters()* inserting arbitrary values in the _thikl array:

```
$ cor_atm = CorsikaAtmosphere(new_location, new_season)
$ cor_atm.calc_thickl()
```

Replace _thickl values with printout.

depth2height(*x_v*)

Converts column/vertical depth to height.

Parameters **x_v** (*float*) – column depth X_v in g/cm**2

Returns height in cm

Return type float

get_density(*h_cm*)

Returns the density of air in g/cm**3.

Uses the optimized module function corsika_get_density_jit().

Parameters **h_cm** (*float*) – height in cm

Returns density $\rho(h_{cm})$ in g/cm**3

Return type float

get_mass_overburden(*h_cm*)

Returns the mass overburden in atmosphere in g/cm**2.

Uses the optimized module function corsika_get_m_overburden_jit()

Parameters **h_cm** (*float*) – height in cm

Returns column depth $T(h_{cm})$ in g/cm**2

Return type float

init_parameters(*location*, *season*)

Initializes *_atm_param*. Parameters from ANTARES/KM3NET are based on the work of T. Heid (see this issue)

location	CORSIKA Table	Description/season
"USStd"	23	US Standard atmosphere
"BK_USStd"	37	Bianca Keilhauer's USStd
"Karlsruhe"	24	AT115 / Karlsruhe
"SouthPole"	26 and 28	MSIS-90-E for Dec and June
"PL_SouthPole"	29 and 30	P. Lipari's Jan and Aug
"ANTARES/KM3NeT-ORCA"	NA	PhD T. Heid
"KM3NeT-ARCA"	NA	PhD T. Heid

Parameters

- **location** (*str*) – see table
- **season** (*str, optional*) – choice of season for supported locations

Raises Exception – if parameter set not available

rho_inv(*X*, *cos_theta*)

Returns reciprocal density in cm**3/g using planar approximation.

This function uses the optimized function `planar_rho_inv_jit()`

Parameters **h_cm** (*float*) – height in cm

Returns $\frac{1}{\rho}(X, \cos\theta)$ cm**3/g

Return type float

class MCEq.geometry.density_profiles.**EarthsAtmosphere**(**args*, ***kwargs*)

Abstract class containing common methods on atmosphere. You have to inherit from this class and implement the virtual method *get_density()*.

Note: Do not instantiate this class directly.

thrad

current zenith angle θ in radiants

Type float

theta_deg

current zenith angle θ in degrees

Type float

max_X

Slant depth at the surface according to the geometry defined in the MCEq.geometry

Type float

geometry
Can be a custom instance of EarthGeometry

Type object

X2h(X)
Returns the height above surface as a function of slant depth for currently selected zenith angle.

The spline *s_lX2h* is used, which was calculated or retrieved from cache during the `set_theta()` call.

Parameters **X** (*float*) – slant depth in g/cm**2

Returns height above surface in cm

Return type float h

X2rho(X)
Returns the density $\rho(X)$.

The spline *s_X2rho* is used, which was calculated or retrieved from cache during the `set_theta()` call.

Parameters **X** (*float*) – slant depth in g/cm**2

Returns ρ in cm**3/g

Return type float

calculate_density_spline(*n_steps=2000*)
Calculates and stores a spline of $\rho(X)$.

Parameters **n_steps** (*int, optional*) – number of X values to use for interpolation

Raises `Exception` – if `set_theta()` was not called before.

gamma_cherenkov_air(*h_cm*)
Returns the Lorentz factor gamma of Cherenkov threshold in air (MeV).

get_density(*h_cm*)
Abstract method which implementation should return the density in g/cm**3.

Parameters **h_cm** (*float*) – height in cm

Returns density in g/cm**3

Return type float

Raises `NotImplementedError`

h2X(h)
Returns the depth along path as function of height above surface.

The spline *s_X2rho* is used, which was calculated or retrieved from cache during the `set_theta()` call.

Parameters **h** (*float*) – vertical height above surface in cm

Returns X slant depth in g/cm**2

Return type float

moliere_air(*h_cm*)
Returns the Moliere unit of air for US standard atmosphere.

nref_rel_air(*h_cm*)
Returns the refractive index - 1 in air (density parametrization as in CORSIKA).

r_X2rho(X)
Returns the inverse density $\frac{1}{\rho}(X)$.

The spline *s_X2rho* is used, which was calculated or retrieved from cache during the `set_theta()` call.

Parameters **X** (*float*) – slant depth in g/cm**2

Returns $1/\rho$ in cm**3/g

Return type float

set_theta(*theta_deg*)
Configures geometry and initiates spline calculation for $\rho(X)$.

If the option 'use_atm_cache' is enabled in the config, the function will check, if a corresponding spline is available in the cache and use it. Otherwise it will call `calculate_density_spline()`, make the function `r_X2rho()` available to the core code and store the spline in the cache.

Parameters **theta_deg** (*float*) – zenith angle θ at detector

theta_cherenkov_air(*h_cm*)
Returns the Cherenkov angle in air (degrees).

max_X
Depth at altitude 0.

max_den
Density at altitude 0.

s_X2rho
Spline for conversion from depth to density.

s_h2X
Spline for conversion from altitude to depth.

s_lX2h
Spline for conversion from depth to altitude.

class MCEq.geometry.density_profiles.**GeneralizedTarget**(*len_target=100000.0*, *env_density=0.001225*, *env_name='air'*)
This class provides a way to run MCEq on piece-wise constant one-dimenional density profiles.

The default values for the average density are taken from config file variables *len_target*, *env_density* and *env_name*. The density profile has to be built by calling subsequently `add_material()`. The current composition of the target can be checked with `draw_materials()` or `print_table()`.

Note: If the target is not air or hydrogen, the result is approximate, since seconray particle yields are provided for nucleon-air or proton-proton collisions. Depending on this choice one has to adjust the nuclear mass in `mceq_config`.

Parameters

- **len_target** (*float*) – total length of the target in meters
- **env_density** (*float*) – density of the default material in g/cm**3
- **env_name** (*str*) – title for this environment

add_material(*start_position_cm*, *density*, *name*)
Adds one additional material to a composite target.

Parameters

- **start_position_cm** (*float*) – position where the material starts counted from target origin l|X = 0 in cm
- **density** (*float*) – density of material in g/cm**3
- **name** (*str*) – any user defined name

Raises Exception – If requested start_position_cm is not properly defined.

draw_materials(*axes=None*, *logx=False*)
Makes a plot of depth and density profile as a function of the target length. The list of materials is printed out, too.

Parameters **axes** (*plt.axes, optional*) – handle for matplotlib axes

get_density(*l_cm*)
Returns the density in g/cm**3 as a function of position l in cm.

Parameters **l** (*float*) – position in target in cm

Returns density in g/cm**3

Return type float

Raises Exception – If requested position exceeds target length.

get_density_X(*X*)
Returns the density in g/cm**3 as a function of depth X.

Parameters **X** (*float*) – depth in g/cm**2

Returns density in g/cm**3

Return type float

Raises Exception – If requested depth exceeds target.

print_table(*min_dbg_lev=0*)
Prints table of materials to standard output.

r_X2rho(*X*)
Returns the inverse density $\frac{1}{\rho}(X)$.

Parameters **X** (*float*) – slant depth in g/cm**2

Returns $1/\rho$ in cm**3/g

Return type float

reset()
Resets material list to defaults.

set_length(*new_length_cm*)
Updates the total length of the target.

Usually the length is set

set_theta(**args*)
This method is not defined for the generalized target. The purpose is to catch usage errors.

Raises NotImplementedError – always

max_X
Maximal depth of target.

s_X2h
Spline for depth at distance.

s_h2X
Spline for distance at depth.

class MCEq.geometry.density_profiles.**IsothermalAtmosphere**(*location*, *season*, *hiso_km=6.3*, *X0=1300.0*)

Isothermal model of the atmosphere.

This model is widely used in semi-analytical calculations. The isothermal approximation is valid in a certain range of altitudes and usually one adjust the parameters to match a more realistic density profile at altitudes between 10 - 30 km, where the high energy muon production rate peaks. Such parametrizations are given in the book "Cosmic Rays and Particle Physics", Gaisser, Engel and Resconi (2016). The default values are from M. Thunman, G. Ingelman, and P. Gondolo, Astropart. Physics 5, 309 (1996).

Parameters

- **location** (*str*) – no effect
- **season** (*str*) – no effect
- **hiso_km** (*float*) – isothermal scale height in km
- **X0** (*float*) – Ground level overburden

get_density(*h_cm*)
Returns the density of air in g/cm**3.

Parameters **h_cm** (*float*) – height in cm

Returns density $\rho(h_{cm})$ in g/cm**3

Return type float

get_mass_overburden(*h_cm*)
Returns the mass overburden in atmosphere in g/cm**2.

Parameters **h_cm** (*float*) – height in cm

Returns column depth $T(h_{cm})$ in g/cm**2

Return type float

class MCEq.geometry.density_profiles.**MSIS00Atmosphere**(*location*, *season=None*, *doy=None*, *use_loc_altitudes=False*)

Wrapper class for a python interface to the NRLMSISE-00 model.

NRLMSISE-00 is an empirical model of the Earth's atmosphere. It is available as a FORTRAN 77 code or as a verson traslated into C by Dominik Borodowski. Here a PYTHON wrapper has been used.

_msis
NRLMSISE-00 python wrapper object handler

Parameters

- **location** (*str*) – see *init_parameters()*
- **season** (*str, optional*) – see *init_parameters()*

get_density(*h_cm*)
Returns the density of air in g/cm**3.

Wraps around ctypes calls to the NRLMSISE-00 C library.

Parameters **h_cm** (*float*) – height in cm

Returns density $\rho(h_{cm})$ in g/cm**3

Return type float

get_temperature(*h_cm*)
Returns the temperature of air in K.

Wraps around ctypes calls to the NRLMSISE-00 C library.

Parameters **h_cm** (*float*) – height in cm

Returns density $T(h_{cm})$ in K

Return type float

init_parameters(*location*, *season*, *doy*, *use_loc_altitudes*)
Sets location and season in `NRLMSISE-00`.

Translates location and season into day of year and geo coordinates.

Parameters

- **location** (*str*) – Supported are "SouthPole" and "Karlsruhe"
- **season** (*str*) – months of the year: January, February, etc.
- **use_loc_altitudes** (*bool*) – If to use default altitudes from location

set_doy(*day_of_year*)
Changes MSIS season by day of year.

Parameters **day_of_year** (*int*) –

1. Jan.=0, 1.Feb=32

set_location(*location*)
Changes MSIS location by strings defined in _msis_wrapper.

Parameters **location** (*str*) – location as defined in `NRLMSISE-00.`

set_location_coord(*longitude*, *latitude*)
Changes MSIS location by longitude, latitude in _msis_wrapper

Parameters

- **longitude** (*float*) – longitude of the location with abs(longitude) <= 180
- **latitude** (*float*) – latitude of the location with abs(latitude) <= 90

set_season(*month*)
Changes MSIS location by month strings defined in _msis_wrapper.

Parameters **location** (*str*) – month as defined in `NRLMSISE-00.`

update_parameters(***kwargs*)
Updates parameters of the density model

Parameters

- **location_coord** (*tuple of str*) – (longitude, latitude)
- **season** (*str*) – months of the year: January, February, etc.
- **doy** (*int*) – day of the year. 'doy' takes precedence over 'season' if both are set

class MCEq.geometry.density_profiles.**MSIS00IceCubeCentered**(*location*, *season*)
Extension of *MSIS00Atmosphere* which couples the latitude setting with the zenith angle of the detector.

Parameters

- **location** (*str*) – see `init_parameters()`
- **season** (*str, optional*) – see `init_parameters()`

latitude (*det_zenith_deg*)
Returns the geographic latitude of the shower impact point.

Assumes a spherical earth. The detector is 1948m under the surface.

Credits: geometry fomulae by Jakob van Santen, DESY Zeuthen.

Parameters **det_zenith_deg** (*float*) – zenith angle at detector in degrees

Returns latitude of the impact point in degrees

Return type float

set_theta (*theta_deg*)
Configures geometry and initiates spline calculation for $\rho(X)$.

If the option 'use_atm_cache' is enabled in the config, the function will check, if a corresponding spline is available in the cache and use it. Otherwise it will call `calculate_density_spline()`, make the function `r_X2rho()` available to the core code and store the spline in the cache.

Parameters **theta_deg** (*float*) – zenith angle θ at detector

8.3.2 MCEq.geometry.geometry

The module contains the geometry for an azimuth symmetric Earth.

class MCEq.geometry.geometry.**EarthGeometry**

A model of the Earth's geometry, approximating it by a sphere. The figure below illustrates the meaning of the parameters.

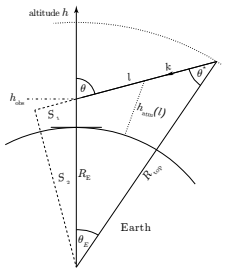


Fig. 1: Curved geometry as it is used in the code (not to scale!).

Example

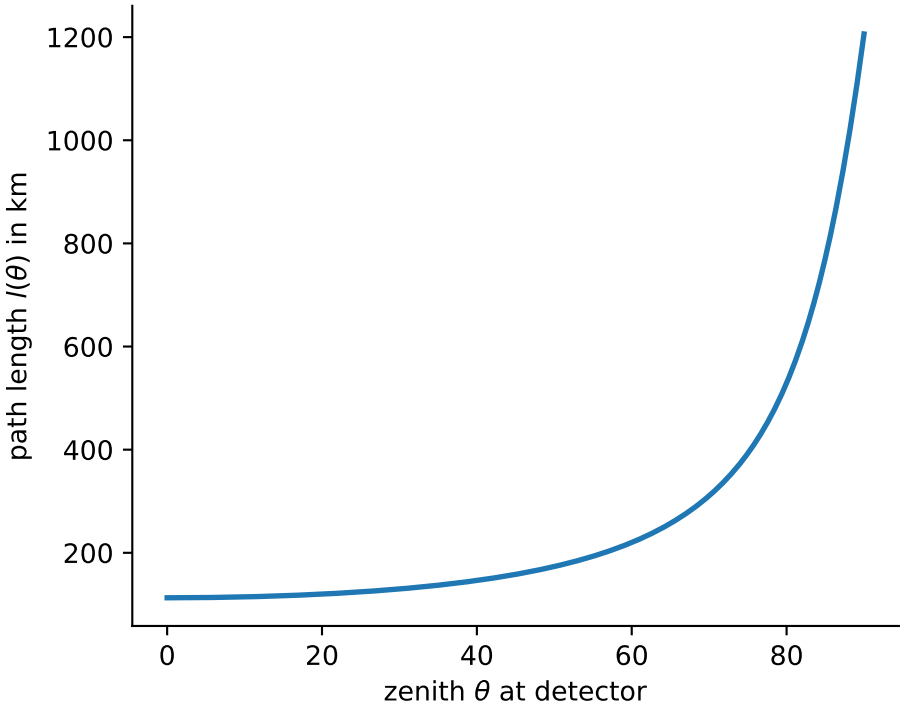
The plots below will be produced by executing the module:

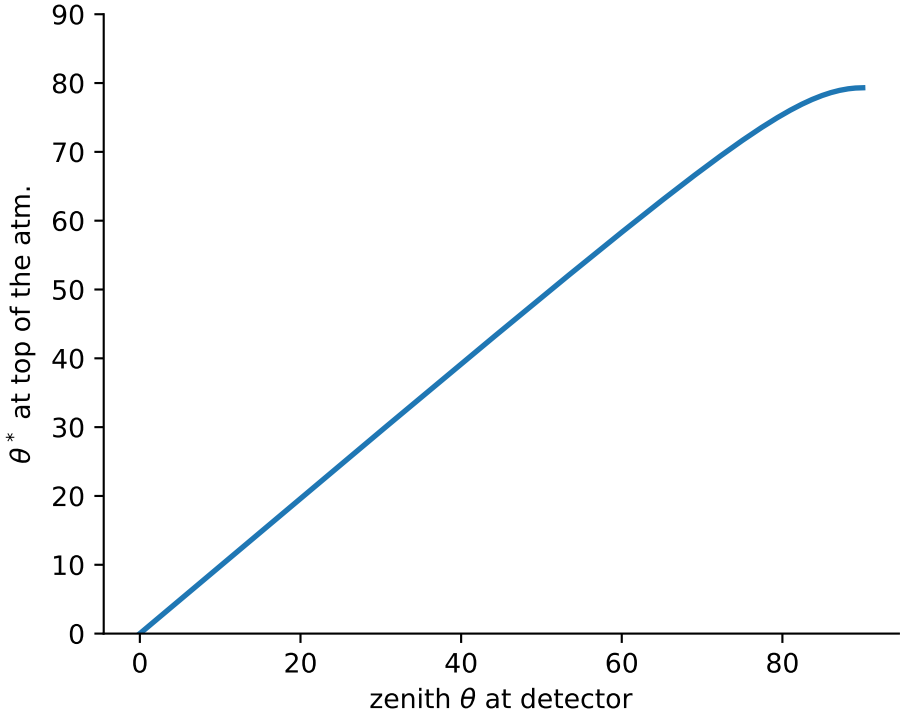
```
$ python geometry.py
```

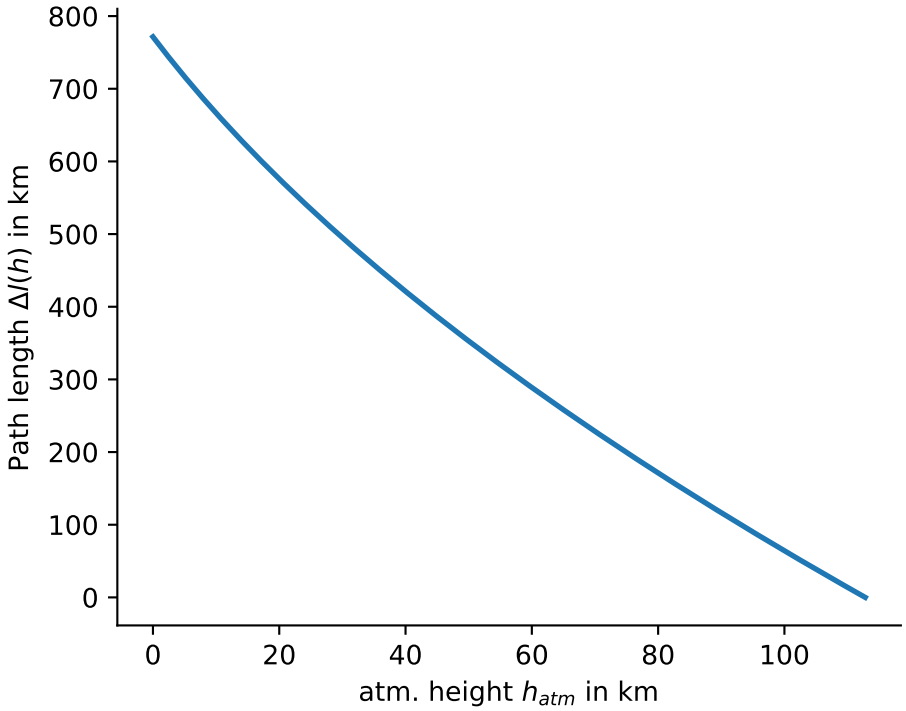
h_obs
observation level height [cm]

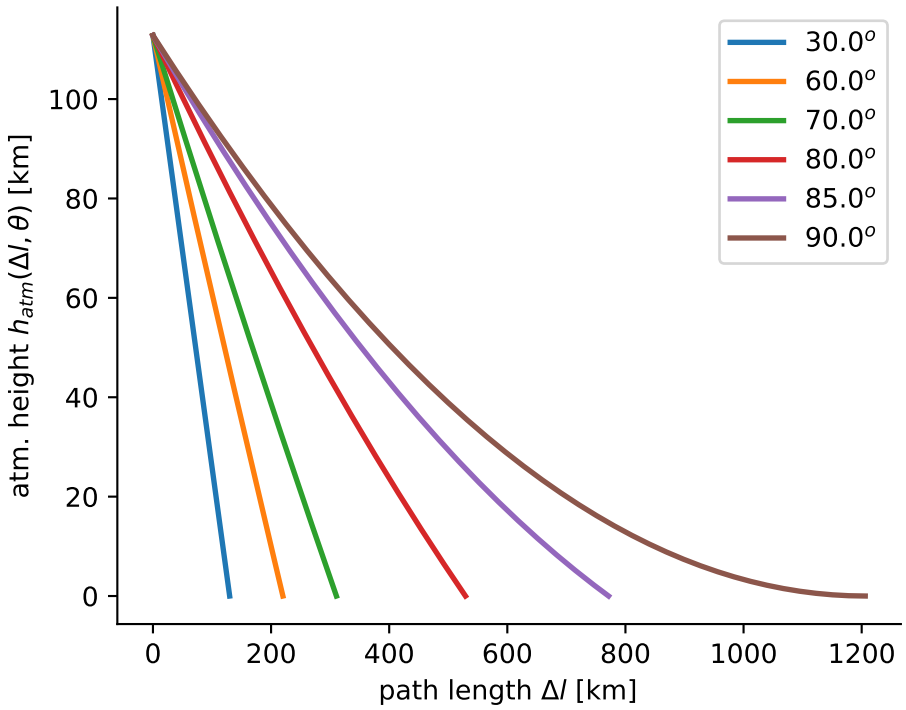
Type float

h_atm
top of the atmosphere [cm]


1200
1000
800
600
400
200
path length $l(\theta)$ in km
0
20
40
60
80
zenith θ at detector


90
80
70
60
50
40
30
20
10
0
θ^* at top of the atm.
0
20
40
60
80
zenith θ at detector


800
700
600
500
400
300
200
100
0
Path length Δl(h) in km
0
20
40
60
80
100
atm. height h_{atm} in km


100
80
60
40
20
0
atm. height h_{atm}(Δl, θ) [km]
0
200
400
600
800
1000
1200
path length Δl [km]
30.0^o
60.0^o
70.0^o
80.0^o
85.0^o
90.0^o

Type float

r_E
radius Earth [cm]

Type float

r_top
radius at top of the atmosphere [cm]

Type float

r_obs
radius at observation level [cm]

Type float

cos_th_star(*theta*)
Returns the zenith angle at atmospheric boarder $\cos(\theta^*)$ in [rad] as a function of zenith at detector.

delta_l(*h*, *theta*)
Distance dl covered along path $l(\theta)$ as a function of current height. Inverse to `h()`.

h(*dl*, *theta*)
Height above surface at distance dl counted from the beginning of path $l(\theta)$ in cm.

l(*theta*)
Returns path length in [cm] for given zenith angle θ [rad].

MCEq.geometry.geometry.**chirkin_cos_theta_star**(*costheta*)
$\cos(\theta^*)$ parameterization.

This function returns the equivalent zenith angle for for very inclined showers. It is based on a CORSIKA study by D. Chirkin, hep-ph/0407078v1, 2004.

Parameters **costheta** (*float*) – $\cos(\theta)$ in [rad]

Returns $\cos(\theta*)$ in [rad]

Return type float

8.3.3 MCEq.geometry.nrlmsise00

CTypes interface to the C-version of the NRLMSISE-00 code, originally developed by Picone et al.. The C-translation is by *Dominik Brodowski <https://www.brodo.de/space/nrlmsise/index.html>_*.

8.3.4 MCEq.geometry.corsikaatm

This set of functions are C implementations of the piecewise defined exponential profiles as used in CORSIKA. An efficient implementation is difficult in plain numpy.

8.4 Advanced documentation

The "advanced documentation" is the almost complete documentation of all modules.

8.4.1 mceq_config – default configuration options

These are all options MCEq accepts. Usually there is no need to change, except for advanced scenarios. Check out the file for a better formatted description and some advanced settings not contained in the list below.

class mceq_config.**FileIntegrityCheck**(*filename*, *checksum=''*)
A class to check a file integrity against provided checksum

filename
path to the file

Type str

checksum
hex of sha256 checksum

Type str

is_passed():
returns True if checksum and calculated checksum of the file are equal

get_file_checksum():
returns checksum of the file

hashlib = <module 'hashlib' from '/home/docs/.pyenv/versions/2.7.18/lib/python2.7/hash

class mceq_config.**MCEqConfigCompatibility**(*namespace*)
This class provides access to the attributes of the module as a dictionary, as it was in the previous versions of MCEq

This method is deprecated and will be removed in future.

mceq_config.**A_target = 14.65672**
Average mass of target (for interaction length calculations) Change parameter only in combination with interaction model setting. By default all particle production matrices are calculated for air targets expect those for models with '_pp' suffix. These are valid for hydrogen targets. <A> = 14.6568 for air as below (source https://en.wikipedia.org/wiki/Atmosphere_of_Earth)

mceq_config.**adv_set = {'allowed_projectiles': [], 'disable_charm_pprod': False, 'disable_**
Advanced settings (some options might be obsolete/not working)

mceq_config.**assume_nucleon_interactions_for_exotics = True**
Assume nucleon, pion and kaon cross sections for interactions of rare or exotic particles (mostly relevant for non-compact mode)

mceq_config.**average_loss_operator = True**
Improve (explicit solver) stability by averaging the continous loss operator

mceq_config.**cuda_fp_precision = 32**
CUDA Floating point precision (default 32-bit 'float')

mceq_config.**cuda_gpu_id = 0**
Select CUDA device ID if you have multiple GPUs

mceq_config.**dXmax = 10.0**
Maximal integration step dX in g/cm2. No limit necessary in most cases, use for debugging purposes when searching for stability issues.

mceq_config.**data_dir = '/home/docs/checkouts/readthedocs.org/user_builds/mceq/checkouts/lat**
Directory where the data files for the calculation are stored

mceq_config.**debug_level = 1**
Debug flag for verbose printing, 0 silences MCEq entirely

mceq_config.**dedx_material = 'air'**
Material for ionization and radiation (=continuous) loss terms Currently available choices: 'air', 'water', 'ice'

mceq_config.**density_model = ('CORSIKA', ('BK_USStd', None))**
(model, (arguments))

Type Atmospheric model in the format

mceq_config.**e_max = 100000000000.0**
The maximal energy is 1e12 GeV, but not all interaction models run at such high energies. If you are interested in lower energies, reduce this value for inclusive calculations to max. energy of interest + 4-5 orders of magnitude. For single primaries the maximal energy is directly limited by this value. Smaller grids speed up the initialization and integration.

mceq_config.**e_min = 0.1**
Minimal energy for grid The minimal energy (technically) is 1e-2 GeV. Currently you can run into stability problems with the integrator with such low thresholds. Use with care and check results for oscillations and feasibility.

mceq_config.**em_db_fname = 'mceq_db_EM_Tsai-Max_Z7.31.h5'**
File name of the MCEq database

mceq_config.**enable_default_tracking = True**
Enable default tracking particles, such as pi_numu, pr_mu+, etc. If only total fluxes are of interest, disable this feature to gain performance since the eqution system becomes smaller and sparser

mceq_config.**enable_em = False**
Enable electromagnetic cascade with matrices from EmCA

mceq_config.**enable_em_ion = False**
enable EM ionization loss

mceq_config.**enable_muon_energy_loss = True**
Muon energy loss according to Kokoulin et al.

mceq_config.**env_density = 0.001225**
density of default material in g/cm^3

mceq_config.**excpt_on_missing_particle = False**
Raise exception when requesting unknown particles from get_solution

mceq_config.**hybrid_crossover = 0.5**
Ratio of decay_length/interaction_length where particle interactions are neglected and the resonance approximation is used 0.5 ~ precision loss <+3% speed gain ~ factor 10 If smoothness and shape accuracy for prompt flux is crucial, use smaller values around 0.1 or 0.05

mceq_config.**integrator = 'euler'**
Selection of integrator (euler/odepack)

mceq_config.**kernel_config = 'numpy'**
euler kernel implementation (numpy/MKL/CUDA). With serious nVidia GPUs CUDA a few times faster than MKL autodetection of fastest kernel below

mceq_config.**leading_process = 'auto'**
The leading process is can be either "decays" or "interactions". This depends on the target density and it is usually chosen automatically. For advanced applications one can force "interactions" to be the dominant process. Essentially this affects how the adaptive step size is computed. There is also the choice of "auto" that takes both processes into account

mceq_config.**len_target = 1000.0**
Default parameters for GeneralizedTarget Total length of the target [m]

mceq_config.**loss_step_for_average = 0.1**
Step size (dX) for averaging

mceq_config.**low_energy_extension = {'he_le_transition': 80, 'nbins_interp': 3, 'use_unkno**
This is not used in the code as before, instead the low energy extension is compiled into the HDF backend files.

mceq_config.**max_density = (0.001225,)**
Approximate value for the maximum density expected. Needed for the resonance approximation. Default value: air at the surface

mceq_config.**mceq_db_fname = 'mceq_db_lext_dpm191_v12.h5'**
File name of the MCEq database

mceq_config.**mkl_threads = 8**
Number of MKL threads (for sparse matrix multiplication the performance advantage from using more than a few threads is limited by memory bandwidth) Irrelevant for GPU integrators, but can affect initialization speed if numpy is linked to MKL.

mceq_config.**muon_helicity_dependence = True**
Helicity dependent muons decays from analytical expressions

mceq_config.**ode_params = {'method': 'bdf', 'name': 'lsoda', 'nsteps': 1000, 'rtol': 0.0**
parameters for the odepack integrator. More details at http://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.ode.html#scipy.integrate.ode

mceq_config.**override_debug_fcn = []**
Override debug prinput for functions listed here (just give the name, "get_solution" for instance) Warning, this option slows down initialization by a lot. Use only when needed.

mceq_config.**override_max_level = 10**
Override debug printout for debug levels < value for the functions above

mceq_config.**pf = 'Linux-5.15.0-1004-aws-x86_64-with-debian-buster-sid'**
Autodetect best solver determine shared library extension and MKL path

mceq_config.**print_module = False**
Print module name in debug output

mceq_config.**prompt_ctau = 0.123**
default ctau < 0.123 cm (that of D0)

Type Definition of prompt

mceq_config.**r_E = 6391000.0**
parameters for EarthGeometry

mceq_config.**return_as = 'kinetic energy'**
The latest versions of MCEq work in kinetic energy not total energy If you want the result to be compatible with the previous choose 'total energy' else 'kinetic energy'

mceq_config.**stability_margin = 0.95**
Stability margin for the integrator. The default 0.95 means that step sizes are chosen 5% away from the stability circle. Usually no need to change, except you know what it does.

mceq_config.**standard_particles = [11, 12, 13, 14, 16, 211, 321, 2212, 2112, 3122, 411, 421,**
Particles for compact mode

mceq_config.**use_isospin_sym = True**
When using modified particle production matrices use isospin symmetries to determine the corresponding modification for neutrons and K0L/K0S

8.4.2 MCEq.core – Core module

This module contains the main program features. Instantiating *MCEq.core.MCEqRun* will initialize the data structures and particle tables, create and fill the interaction and decay matrix and check if all information for the calculation of inclusive fluxes in the atmosphere is available.

class MCEq.core.**MCEqRun**(*interaction_model*, *primary_model*, *theta_deg*, ***kwargs*)
Main class for handling the calculation.

This class is the main user interface for the caclulation. It will handle initialization and various error/configuration checks. The setup has to be accomplished before invoking the integration routine is `MCeqRun.solve()`. Changes of configuration, such as:

- interaction model in *MCEqRun.set_interaction_model()*,
- primary flux in *MCEqRun.set_primary_model()*,
- zenith angle in *MCEqRun.set_theta_deg()*,
- density profile in *MCEqRun.set_density_model()*,
- member particles of the special `obs_` group in `MCEqRun.set_obs_particles()`,

can be made on an active instance of this class, while calling *MCEqRun.solve()* subsequently to calculate the solution corresponding to the settings.

The result can be retrieved by calling *MCEqRun.get_solution()*.

Parameters

- **interaction_model** (*string*) – PDG ID of the particle
- **density_model** (*string*, *sting*, *string*) – model type, location, season
- **primary_model** (*class*, *param_tuple*) – classes derived from `crflux.models.PrimaryFlux` and its parameters as tuple
- **theta_deg** (*float*) – zenith angle θ in degrees, measured positively from vertical direction
- **adv_set** (*dict*) – advanced settings, see *mceq_config*
- **obs_ids** (*list*) – list of particle name strings. Those lepton decay products will be scored in the special `obs_` categories

closest_energy(*kin_energy*)
Convenience function to obtain the nearest grid energy to the *energy* argument, provided as kinetik energy in lab. frame.

decay_z_factor(*parent_pdg*, *child_pdg*)
Energy dependent Z-factor according to Lipari (1993).

get_solution(*particle_name*, *mag=0.0*, *grid_idx=None*, *integrate=False*, *return_as='kinetic energy'*)

Retrieves solution of the calculation on the energy grid.

Some special prefixes are accepted for lepton names:

- the total flux of muons, muon neutrinos etc. from all sources/mothers can be retrieved by the prefix `total_`, i.e. `total_numu`
- the conventional flux of muons, muon neutrinos etc. from all sources can be retrieved by the prefix `conv_`, i.e. `conv_numu`
- correspondigly, the flux of leptons which originated from the decay of a charged pion carries the prefix `pi_` and from a kaon `k_`
- conventional leptons originating neither from pion nor from kaon decay are collected in a category without any prefix, e.g. `numu` or `mu+`

Parameters

- **particle_name** (*str*) – The name of the particle such, e.g. `total_mu+` for the total flux spectrum of positive muons or `pr_antinumu` for the flux spectrum of prompt anti muon neutrinos
- **mag** (*float, optional*) – 'magnification factor': the solution is multiplied by `sol` $= \Phi \cdot E^{mag}$
- **grid_idx** (*int, optional*) – if the integrator has been configured to save intermediate solutions on a depth grid, then `grid_idx` specifies the index of the depth grid for which the solution is retrieved. If not specified the flux at the surface is returned
- **integrate** (*bool, optional*) – return averge particle number instead of
- **flux** (*multiply by bin width*) –

Returns flux of particles on energy grid *e_grid*

Return type (numpy.array)

n_e(*grid_idx=None*, *min_energy_cutoff=0.1*)

Returns the number of electrons plus positrons at a grid step above *min_energy_cutoff*.

Parameters

- **grid_idx** (*int*) – Depth grid index (for profiles)
- **min_energy_cutoff** (*float*) – Energy threshold > mceq_config.e_min

n_mu(*grid_idx=None*, *min_energy_cutoff=0.1*)

Returns the number of positive and negative muons at a grid step above *min_energy_cutoff*.

Parameters

- **grid_idx** (*int*) – Depth grid index (for profiles)
- **min_energy_cutoff** (*float*) – Energy threshold > mceq_config.e_min

n_particles(*label*, *grid_idx=None*, *min_energy_cutoff=0.1*)

Returns number of particles of type *label* at a grid step above an energy threshold for counting.

Parameters

- **label** (*str*) – Particle name
- **grid_idx** (*int*) – Depth grid index (for profiles)

- **min_energy_cutoff** (*float*) – Energy threshold > mceq_config.e_min

regenerate_matrices(*skip_decay_matrix=False*)
Call this function after applying particle prod. modifications aka Barr parameters

set_density_model(*density_config*)
Sets model of the atmosphere.

To choose, for example, a CORSIKA parametrization for the Southpole in January, do the following:

```
mceq_instance.set_density_model(('CORSIKA', ('PL_SouthPole', 'January')))
```

More details about the choices can be found in *MCEq.geometry.density_profiles*. Calling this method will issue a recalculation of the interpolation and the integration path.

From version 1.2 and above, the *density_config* parameter can be a reference to an instance of a density class directly. The class has to be derived either from *MCEq.geometry.density_profiles.EarthsAtmosphere* or *MCEq.geometry.density_profiles.GeneralizedTarget*.

Parameters **density_config** (*tuple of strings*) – (parametrization type, arguments)

set_initial_spectrum(*spectrum*, *pdg_id=None*, *append=False*)
Set a user-defined spectrum for an arbitrary species as initial condition.

This function is an equivalent to *set_single_primary_particle()*. It allows to define an arbitrary spectrum for each available particle species as initial condition for the integration. Set the *append* argument to *True* for subsequent species to define initial spectra combined from different particles.

The (differential) spectrum has to be distributed on the energy grid as dN/dptot, i.e. divided by the bin widths and with the total momentum units in GeV(/c).

Parameters

- **spectrum** (*np.array*) – spectrum dN/dptot
- **pdg_id** (*int*) – PDG ID in case of a particle

set_interaction_model(*interaction_model*, *particle_list=None*, *update_particle_list=True*, *force=False*, *build_matrices=True*)
Sets interaction model and/or an external charm model for calculation.

Decay and interaction matrix will be regenerated automatically after performing this call.

Parameters

- **interaction_model** (*str*) – name of interaction model
- **charm_model** (*str, optional*) – name of charm model
- **force** (*bool*) – force loading interaction model

set_mod_pprod(*prim_pdg*, *sec_pdg*, *x_func*, *x_func_args*, *delay_init=False*)
Sets combination of projectile/secondary for error propagation.

The production spectrum of `sec_pdg` in interactions of `prim_pdg` is modified according to the function passed to `InteractionYields.init_mod_matrix()`

Parameters

- **prim_pdg** (*int*) – interacting (primary) particle PDG ID
- **sec_pdg** (*int*) – secondary particle PDG ID
- **x_func** (*object*) – reference to function

- **x_func_args** (*tuple*) – arguments passed to x_func
- **delay_init** (*bool*) – Prevent init of mceq matrices if you are planning to add more modifications

set_primary_model(*mclass*, *tag*)
Sets primary flux model.

This functions is quick and does not require re-generation of matrices.

Parameters

- **interaction_model** (CRFluxModel.PrimaryFlux) – reference
- **primary model class** (*to*) –
- **tag** (*tuple*) – positional argument list for model class

set_single_primary_particle(*E*, *corsika_id=None*, *pdg_id=None*, *append=False*)
Set type and kinetic energy of a single primary nucleus to calculation of particle yields.

The functions uses the superposition theorem, where the flux of a nucleus with mass A and charge Z is modeled by using Z protons and A-Z neutrons at energy $E_{nucleon} = E_{nucleus}/A$ The nucleus type is defined via CORSIKA ID $= A * 100 + Z$. For example iron has the CORSIKA ID 5226.

Single leptons or hadrons can be defined by specifiying *pdg_id* instead of *corsika_id*.

The *append* argument can be used to compose an initial state with multiple particles. If it is *False* the initial condition is reset to zero before adding the particle.

A continuous input energy range is allowed between $50 * A$ GeV $< E_{\text{nucleus}} < 10^{10} * A$GeV.

Parameters

- **E** (*float*) – kinetic energy of a nucleus in GeV
- **corsika_id** (*int*) – ID of a nucleus (see text)
- **pdg_id** (*int*) – PDG ID of a particle
- **append** (*bool*) – If True, keep previous state and append a new particle.

set_theta_deg(*theta_deg*)
Sets zenith angle θ as seen from a detector.

Currently only 'down-going' angles (0-90 degrees) are supported.

Parameters **theta_deg** (*float*) – zenith angle in the range 0-90 degrees

solve(*int_grid=None*, *grid_var='X'*, ***kwargs*)
Launches the solver.

The setting *integrator* in the config file decides which solver to launch.

Parameters

- **int_grid** (*list*) – list of depths at which results are recorded
- **grid_var** (*str*) – Can be depth *X* or something else (currently only *X* supported)
- **kwargs** (*dict*) – Arguments are passed directly to the solver methods.

unset_mod_pprod(*dont_fill=False*)
Removes modifications from *MCEqRun.set_mod_pprod()*.

Parameters

- **skip_fill** (*bool*) – If *true* do not regenerate matrices

• to be done at a later step by hand) (*(has)* –

z_factor(*projectile_pdg*, *secondary_pdg*, *definition='primary_e'*)
Energy dependent Z-factor according to Thunman et al. (1996)

dim
Energy grid (dimension)

dim_states
Number of cascade particles times dimension of grid (dimension of the equation system)

e_bins
Energy grid (bin edges)

e_grid
Energy grid (bin centers)

e_widths
Energy grid (bin widths)

pman = None
Particle manager (initialized/updated in set_interaction_model)

class MCEq.core.**MatrixBuilder**(*particle_manager*)
This class constructs the interaction and decay matrices.

construct_matrices(*skip_decay_matrix=False*)
Constructs the matrices for calculation.

These are:

- $\boldsymbol{M}_{int} = (-\mathbf{1} + \boldsymbol{C})\boldsymbol{\Lambda}_{int}$,
- $\boldsymbol{M}_{dec} = (-\mathbf{1} + \boldsymbol{D})\boldsymbol{\Lambda}_{dec}$.

For debug_levels >= 2 some general information about matrix shape and the number of non-zero elements is printed. The intermediate matrices $\boldsymbol{C}$ and $\boldsymbol{D}$ are deleted afterwards to save memory.

Set the `skip_decay_matrix` flag to avoid recreating the decay matrix. This is not necessary if, for example, particle production is modified, or the interaction model is changed.

Parameters **skip_decay_matrix** (*bool*) – Omit re-creating D matrix

cont_loss_operator(*pdg_id*)
Returns continuous loss operator that can be summed with appropriate position in the C matrix.

dim
Energy grid (dimension)

dim_states
Number of cascade particles times dimension of grid (dimension of the equation system)

8.4.3 MCEq.particlemanager – Particle manager

The *MCEq.particlemanager.ParticleManager* handles the bookkeeping of MCEq.particlemanager.MCEqParticle`s. It feeds the parameterizations of interactions and decays from :mod:`MCEq.data into the corresponding variables and validates certain relations. The construction of the interaction and decay matrices proceeds by iterating over the particles in *MCEq.particlemanager.ParticleManager*, querying the interaction and decay yields for child particles. Therefore, there is usually no need to directly access any of the classes in *MCEq.data*.

class MCEq.particlemanager.**MCEqParticle**(*pdg_id*, *helicity*, *energy_grid=None*, *cs_db=None*, *init_pdata_defaults=True*)

Bundles different particle properties for simplified availability of particle properties in *MCEq.core.MCEqRun*.

Parameters

- **pdg_id** (*int*) – PDG ID of the particle
- **egrid** (*np.array, optional*) – energy grid (centers)
- **cs_db** (*object, optional*) – reference to an instance of InteractionYields

add_decay_channel(*child*, *dec_matrix*, *force=False*)

Add a decay channel.

The branching ratios are not renormalized and one needs to take care of this externally.

add_hadronic_production_channel(*child*, *int_matrix*)

Add a new particle that is produced in hadronic interactions.

The int_matrix is expected to be in the correct shape and scale as the other interaction (dN/dE(i,j)) matrices. Energy conservation is not checked.

dN_dEkin(*kin_energy*, *sec_pdg*, *verbose=True*, ***kwargs*)

Returns $dN/dE_{\rm Kin}$ in lab frame for an interaction energy close to kin_energy (total) for hadron-air collisions.

The function respects modifications applied via _set_mod_pprod().

Parameters

- **kin_energy** (*float*) – approximate interaction energy
- **prim_pdg** (*int*) – PDG ID of projectile
- **sec_pdg** (*int*) – PDG ID of secondary particle
- **verbose** (*bool*) – print out the closest energy

Returns $x_{\rm Lab}$, $dN/dx_{\rm Lab}$

Return type (numpy.array, numpy.array)

dN_dxf(*energy*, *prim_pdg*, *sec_pdg*, *pos_only=True*, *verbose=True*, ***kwargs*)

Returns $dN/dx_{\rm F}$ in c.m. for interaction energy close to energy (lab. not kinetic) for hadron-air collisions.

The function respects modifications applied via _set_mod_pprod().

Parameters

- **energy** (*float*) – approximate interaction lab. energy
- **prim_pdg** (*int*) – PDG ID of projectile
- **sec_pdg** (*int*) – PDG ID of secondary particle
- **verbose** (*bool*) – print out the closest energy

Returns $x_{\rm F}$, $dN/dx_{\rm F}$

Return type (numpy.array, numpy.array)

dN_dxlab(*kin_energy*, *sec_pdg*, *verbose=True*, ***kwargs*)

Returns $dN/dx_{\rm Lab}$ for interaction energy close to kin_energy for hadron-air collisions.

The function respects modifications applied via _set_mod_pprod().

Parameters

- **kin_energy** (*float*) – approximate interaction kin_energy
- **prim_pdg** (*int*) – PDG ID of projectile
- **sec_pdg** (*int*) – PDG ID of secondary particle
- **verbose** (*bool*) – print out the closest enerkin_energygy

Returns $x_{\rm Lab}$, $dN/dx_{\rm Lab}$

Return type (numpy.array, numpy.array)

dNdec_dxlab(*kin_energy*, *sec_pdg*, *verbose=True*, ***kwargs*)
Returns $dN/dx_{\rm Lab}$ for interaction energy close to `kin_energy` for hadron-air collisions.

The function respects modifications applied via `_set_mod_pprod()`.

Parameters

- **kin_energy** (*float*) – approximate interaction energy
- **prim_pdg** (*int*) – PDG ID of projectile
- **sec_pdg** (*int*) – PDG ID of secondary particle
- **verbose** (*bool*) – print out the closest energy

Returns $x_{\rm Lab}$, $dN/dx_{\rm Lab}$

Return type (numpy.array, numpy.array)

inel_cross_section(*mbarn=False*)
Returns inelastic cross section.

Parameters **mbarn** (*bool*) – if True cross section in mb otherwise in cm**2

Returns $\sigma_{\rm inel}$ in mb or cm**2

Return type (float)

init_custom_particle_data(*name*, *pdg_id*, *helicity*, *ctau*, *mass*, ***kwargs*)
Add custom particle type. (Incomplete and not debugged)

inverse_decay_length(*cut=True*)
Returns inverse decay length (or infinity (np.inf), if particle is stable), where the air density ρ is factorized out.

Parameters

- **E** (*float*) – energy in laboratory system in GeV
- **cut** (*bool*) – set to zero in 'resonance' regime

Returns $\frac{\rho}{\lambda_{dec}}$ in 1/cm

Return type (float)

inverse_interaction_length()
Returns inverse interaction length for A_target given by config.

Returns $\frac{1}{\lambda_{int}}$ in cm**2/g

Return type (float)

is_child(*particle_ref*)
True if this particle decays into *particle_ref*.

is_secondary(*particle_ref*)
True if this projectile and produces particle *particle_ref*.

set_cs(*cs_db*)
Set cross section adn recalculate the dependent variables

set_decay_channels(*decay_db*, *pmanager*)
Populates decay channel and energy distributions

set_hadronic_channels(*hadronic_db*, *pmanager*)
Changes the hadronic interaction model.

Replaces indexing of the yield dictionary from PDG IDs with references from partilcle manager.

A = None
Mass, charge, neutron number

E_crit = None
(float) critical energy in air at the surface

N = None
Mass, charge, neutron number

Z = None
Mass, charge, neutron number

can_interact = None
(bool) can_interact

ctau = None
(float) ctau in cm

dEdX = None
(np.array) continuous losses in GeV/(g/cm2)

decay_dists = None
decay channels if any

hadridx
Returns index range where particle behaves as hadron.

Returns range on energy grid

Return type `tuple()` (int,int)

has_contloss = None
(bool) has continuous losses dE/dX defined

helicity = None
(int) helicity -1, 0, 1 (0 means undefined or average)

is_em = None
(bool) if it's an electromagnetic particle

is_hadron = None
(bool) particle is a hadron

is_lepton = None
(bool) particle is a lepton

is_mixed = None
(bool) particle has both, hadron and resonance properties

is_nucleus = None
(bool) particle is a nucleus (not yet implemented)

is_projectile = None
(bool) particle is interacting projectile

is_resonance = None
(bool) if particle has just resonance behavior

is_stable = None
(bool) particle is stable

is_tracking = None
(bool) is a tracking particle

lidx
Returns lower index of particle range in state vector.

Returns lower index in state vector `MCEqRun.phi`

Return type (int)

mass = None
(float) mass in GeV

mceqidx = None
(int) MCEq ID

name = None
(str) species name in string representation

pdg_id = None
(int) Particle Data Group Monte Carlo particle ID

residx
Returns index range where particle behaves as resonance.

Returns range on energy grid

Return type `tuple()` (int,int)

uidx
Returns upper index of particle range in state vector.

Returns upper index in state vector `MCEqRun.phi`

Return type (int)

unique_pdg_id = None
(int) Unique PDG ID that is different for tracking particles

class MCEq.particlemanager.**ParticleManager**(*pdg_id_list*, *energy_grid*, *cs_db*, *mod_table=None*)

Database for objects of *MCEqParticle*.

Authors: Anatoli Fedynitch (DESY) Jonas Heinze (DESY)

add_tracking_particle(*parent_list*, *child_pdg*, *alias_name*, *from_interactions=False*)
Allows tracking decay and particle production chains.

Replaces previous `obs_particle` function that allowed to track only leptons from decays certain particles. This present feature removes the special PDG IDs 71XX, 72XX, etc and allows to define any channel like:

```
$ particleManagerInstance.add_tracking_particle([211], 14, 'pi_numu')
```

This will store muon neutrinos from pion decays under the alias 'pi_numu'. Multiple parents are allowed:

```
$ particleManagerInstance.add_tracking_particle(
    [411, 421, 431], 14, 'D_numu')
```

Parameters

- **alias** (*str*) – Name alias under which the result is accessible in get_solution
- **parents** (*list*) – list of parent particle PDG ID's
- **child** (*int*) – Child particle
- **from_interactions** (*bool*) – track particles from interactions

keys()
Returns pdg_ids of all particles

set_continuous_losses(*contloss_db*)
Set continuous losses terms to particles with ionization and radiation losses.

set_cross_sections_db(*cs_db*)
Sets the inelastic cross section to each interacting particle.

This applies to most of the hadrons and does not imply that the particle becomes a projectile. parents need in addition defined hadronic channels.

set_decay_channels(*decay_db*)
Attaches the references to the decay yield tables to each unstable particle

set_interaction_model(*cs_db*, *hadronic_db*, *updated_parent_list=None*, *force=False*)
Attaches the references to the hadronic yield tables to each projectile particle

track_leptons_from(*parent_pdg_list*, *prefix*, *exclude_em=True*, *from_interactions=False*, *use_helicities=False*)
Adds tracking particles for all leptons coming from decays of parents in *parent_pdg_list*.

mceqidx2pdg = None
(dict) Converts index in state vector to PDG ID

mceqidx2pref = None
(dict) Converts MCEq index to reference of class:*particlemanager.MCEqParticle*

nspec = None
(int) Total number of species

pdg2mceqidx = None
(dict) Converts PDG ID to index in state vector

pname2mceqidx = None
(dict) Converts particle name to index in state vector

pname2pref = None
(dict) Converts particle name to reference of class:*particlemanager.MCEqParticle*

8.4.4 MCEq.data – Data handling

The tabulated data in MCEq is handled by HDF5Backend. The HDF5 file densly packed data, where matrices are stored as vectors of a sparse CSR data structure. Index dictionaries and other metadata are stored as attributes. The other classes pf this module know how to interact with the backend and provide an intermediate step to the *ParticleManager* that propagates data further to the *MCEqParticle* objects.

class MCEq.data.**ContinuousLosses**(*mceq_hdf_db*, *material='air'*)
Class for managing the dictionary of hadron-air cross-sections.

Parameters

- **mceq_hdf_db** (*object*) – instance of *MCEq.data.HDF5Backend*

- **material** (*str*) – name of the material (not fully implemented)

energy_grid = None
reference to energy grid

index_d = None
Dictionary containing the distribuiton matrices

mceq_db = None
MCEq HDF5Backend reference

parents = None
List of active parents

class MCEq.data.**Decays**(*mceq_hdf_db*, *default_decay_dset='full_decays'*)
Class for managing the dictionary of decay yield matrices.

Parameters **mceq_hdf_db** (*object*) – instance of *MCEq.data.HDF5Backend*

get_matrix(*parent*, *child*)
Returns a DIM x DIM decay matrix.

Parameters

- **parent** (*int*) – PDG ID of parent particle
- **child** (*int*) – PDG ID of final state child particle

Returns decay matrix

Return type numpy.array

mceq_db = None
MCEq HDF5Backend reference

parent_list = None
(list) List of particles in the decay matrices

class MCEq.data.**HDF5Backend**
Provides access to tabulated data stored in an HDF5 file.

The file contains all necessary ingredients to run MCEq, i.e. no other files are required. This database is not maintained in git and it will change infrequently.

class MCEq.data.**InteractionCrossSections**(*mceq_hdf_db*, *interaction_model='SIBYLL2.3c'*)
Class for managing the dictionary of hadron-air cross-sections.

Parameters

- **mceq_hdf_db** (*object*) – instance of *MCEq.data.HDF5Backend*
- **interaction_model** (*str*) – name of the interaction model

get_cs(*parent*, *mbarn=False*)
Returns inelastic parent-air cross-section $\sigma_{inel}^{proj-Air}(E)$ as vector spanned over the energy grid.

Parameters

- **parent** (*int*) – PDG ID of parent particle
- **mbarn** (*bool, optional*) – if True, the units of the cross-section will be $mbarn$, else cm^2

Returns cross-section in $mbarn$ or cm^2

Return type numpy.array

GeV2mbarn = 0.38937930376300284
unit - $\mathrm{GeV}^2 \cdot$ mbarn

GeVcm = 1.9732696312541852e-14
unit - GeV $\cdot$ cm

GeVfm = 0.19732696312541853
unit - GeV $\cdot$ fm

energy_grid = None
reference to energy grid

iam = None
(str) Interaction Model name

index_d = None
Dictionary containing the distribuiton matrices

mbarn2cm2 = 9.999999999999999e-28
unit conversion - mbarn $\rightarrow \mathrm{cm}^2$

mceq_db = None
MCEq HDF5Backend reference

parents = None
List of active parents

class MCEq.data.**Interactions**(*mceq_hdf_db*)
Class for managing the dictionary of interaction yield matrices.

Args: mceq_hdf_db (object): instance of *MCEq.data.HDF5Backend*

get_matrix(*parent*, *child*)
Returns a DIM x DIM yield matrix.

Parameters

- **parent** (*int*) – PDG ID of parent particle
- **child** (*int*) – PDG ID of final state child/secondary particle

Returns yield matrix

Return type numpy.array

print_mod_pprod()
Prints the active particle production modification.

description = None
String containing the desciption of the model

energy_grid = None
reference to energy grid

iam = None
(str) Interaction Model name

index_d = None
Dictionary containing the distribuiton matrices

mceq_db = None
MCEq HDF5Backend reference

mod_pprod = None
(tuple) modified particle combination for error prop.

parents = None
List of active parents

particles = None
List of all known particles

relations = None
Dictionary parent/child relations

8.4.5 MCEq.solvers – ODE solver implementations

The module contains functions which are called by *MCEq.core.MCEqRun.solve()* method.

The implementation is a simple Forward-Euler stepper. The stability is under control since the smallest Eigenvalues are known a priory. The step size is "adaptive", but it is deterministic and known before the integration starts.

The steps that each solver routine does are:

$$\Phi_{i+1} = \Delta X_i \boldsymbol{M}_{int} \cdot \Phi_i + \frac{\Delta X_i}{\rho(X_i)} \cdot \boldsymbol{M}_{dec} \cdot \Phi_i)$$

with

$$\boldsymbol{M}_{int} = (-\mathbf{1} + \boldsymbol{C})\boldsymbol{\Lambda}_{int} \tag{8.1}$$

and

$$\boldsymbol{M}_{dec} = (-\mathbf{1} + \boldsymbol{D})\boldsymbol{\Lambda}_{dec}. \tag{8.2}$$

As one can easily see, each step can be represented by two sparse *gemv* calls and one vector addition. This is what happens in the MKL and CUDA functions below.

The fastest solver is using NVidia's cuSparse library provided via the cupy matrix library. Intel MKL is recommended for Intel CPUs, in particular since MKL is using AVX instructions. The plain numpy solver is for compatibility and hacking, but not recommended for general use.

class MCEq.solvers.**CUDASparseContext**(*int_m*, *dec_m*, *device_id=0*)
This class handles the transfer between CPU and GPU memory, and the calling of GPU kernels. Initialized by *MCEq.core.MCEqRun* and used by *solv_CUDA_sparse()*.

alloc_grid_sol(*dim*, *nsols*)
Allocates memory for intermediate if grid solution requested.

dump_sol()
Saves current solution to a new index in grid solution memory.

get_gridsol()
Downloads grid solution to main memory.

get_phi()
Downloads current solution from GPU memory.

set_matrices(*int_m*, *dec_m*)
Upload sparce matrices to GPU memory

set_phi(*phi*)
Uploads initial condition to GPU memory.

solve_step(*rho_inv*, *dX*)
Makes one solver step on GPU using cuSparse (BLAS)

MCEq.solvers.**solv_CUDA_sparse**(*nsteps*, *dX*, *rho_inv*, *context*, *phi*, *grid_idcs*)
NVIDIA CUDA cuSPARSE implementation of forward-euler integration.

Function requires a working `accelerate` installation.

Parameters

- **nsteps** (*int*) – number of integration steps
- **dX** (*numpy.array[nsteps]*) – vector of step-sizes ΔX_i in g/cm**2
- **rho_inv** (*numpy.array[nsteps]*) – vector of density values $\frac{1}{\rho(X_i)}$
- **int_m** (*numpy.array*) – interaction matrix (8.1) in dense or sparse representation
- **dec_m** (*numpy.array*) – decay matrix (8.2) in dense or sparse representation
- **phi** (*numpy.array*) – initial state vector $\Phi(X_0)$
- **mu_loss_handler** (*object*) – object of type `SemiLagrangianEnergyLosses`

Returns state vector $\Phi(X_{nsteps})$ after integration

Return type numpy.array

MCEq.solvers.**solv_MKL_sparse**(*nsteps*, *dX*, *rho_inv*, *int_m*, *dec_m*, *phi*, *grid_idcs*)
Intel MKL sparse BLAS implementation of forward-euler integration.

Function requires that the path to the MKL runtime library `libmkl_rt.[so/dylib]` defined in the config file.

Parameters

- **nsteps** (*int*) – number of integration steps
- **dX** (*numpy.array[nsteps]*) – vector of step-sizes ΔX_i in g/cm**2
- **rho_inv** (*numpy.array[nsteps]*) – vector of density values $\frac{1}{\rho(X_i)}$
- **int_m** (*numpy.array*) – interaction matrix (8.1) in dense or sparse representation
- **dec_m** (*numpy.array*) – decay matrix (8.2) in dense or sparse representation
- **phi** (*numpy.array*) – initial state vector $\Phi(X_0)$
- **grid_idcs** (*list*) – indices at which longitudinal solutions have to be saved.

Returns state vector $\Phi(X_{nsteps})$ after integration

Return type numpy.array

MCEq.solvers.**solv_numpy**(*nsteps*, *dX*, *rho_inv*, *int_m*, *dec_m*, *phi*, *grid_idcs*)
`numpy` implementation of forward-euler integration.

Parameters

- **nsteps** (*int*) – number of integration steps
- **dX** (*numpy.array[nsteps]*) – vector of step-sizes ΔX_i in g/cm**2
- **rho_inv** (*numpy.array[nsteps]*) – vector of density values $\frac{1}{\rho(X_i)}$
- **int_m** (*numpy.array*) – interaction matrix (8.1) in dense or sparse representation
- **dec_m** (*numpy.array*) – decay matrix (8.2) in dense or sparse representation
- **phi** (*numpy.array*) – initial state vector $\Phi(X_0)$

Returns state vector $\Phi(X_{nsteps})$ after integration

Return type numpy.array

8.4.6 Miscellaneous

Different helper functions.

class MCEq.misc.**energy_grid**(*c*, *b*, *w*, *d*)
Energy grid (centers, bind widths, dimension)

b
Alias for field number 1

c
Alias for field number 0

d
Alias for field number 3

w
Alias for field number 2

MCEq.misc.**caller_name**(*skip=2*)
Get a name of a caller in the format module.class.method

skip specifies how many levels of stack to skip while getting caller name. skip=1 means "who calls me", skip=2 "who calls my caller" etc. An empty string is returned if skipped levels exceed stack height.abs

From https://gist.github.com/techtonik/2151727

MCEq.misc.**corsikaid2pdg**(*corsika_id*)
Conversion of CORSIKA nuclear code to PDG nuclear code

MCEq.misc.**gen_xmat**(*energy_grid*)
Generates x_lab matrix for a given energy grid

MCEq.misc.**getAZN**(*pdg_id*)
Returns mass number A, charge Z and neutron number N of pdg_id.

Note:

```
PDG ID for nuclei is coded according to 10LZZZAAAI.
For iron-52 it is 1000260520.
```

Parameters **pdgid** (*int*) – PDG ID of nucleus/mass group

Returns (Z,A) tuple

Return type (int,int,int)

MCEq.misc.**getAZN_corsika**(*corsikaid*)
Returns mass number A, charge Z and neutron number N of corsikaid.

Parameters **corsikaid** (*int*) – corsika id of nucleus/mass group

Returns (Z,A) tuple

Return type (int,int,int)

MCEq.misc.**info**(*min_dbg_level*, **message*, ***kwargs*)
Print to console if *min_debug_level* <= *config.debug_level*

The fuction determines automatically the name of caller and appends the message to it. Message can be a tuple of strings or objects which can be converted to string using *str()*.

Parameters

- **min_dbg_level** (*int*) – Minimum debug level in config for printing
- **message** (*tuple*) – Any argument or list of arguments that casts to str
- **condition** (*bool*) – Print only if condition is True
- **blank_caller** (*bool*) – blank the caller name (for multiline output)
- **no_caller** (*bool*) – don't print the name of the caller

Authors: Anatoli Fedynitch (DESY) Jonas Heinze (DESY)

MCEq.misc.**is_charm_pdgid**(*pdgid*)
Returns True if particle ID belongs to a heavy (charm) hadron.

MCEq.misc.**pdg2corsikaid**(*pdg_id*)
Conversion from nuclear PDG ID to CORSIKA ID.

Note:

```
PDG ID for nuclei is coded according to 10LZZZAAAI.
For iron-52 it is 1000260520.
```

MCEq.misc.**print_in_rows**(*min_dbg_level*, *str_list*, *n_cols=5*)
Prints contents of a list in rows *n_cols* entries per row.

MCEq.misc.**theta_deg**(*cos_theta*)
Converts $\cos\theta$ to θ in degrees.

MCEq.misc.**theta_rad**(*theta*)
Converts θ from rad to degrees.

8.5 Differences between V 1.2.0 and 1.1.3

8.5.1 Lepton fluxes

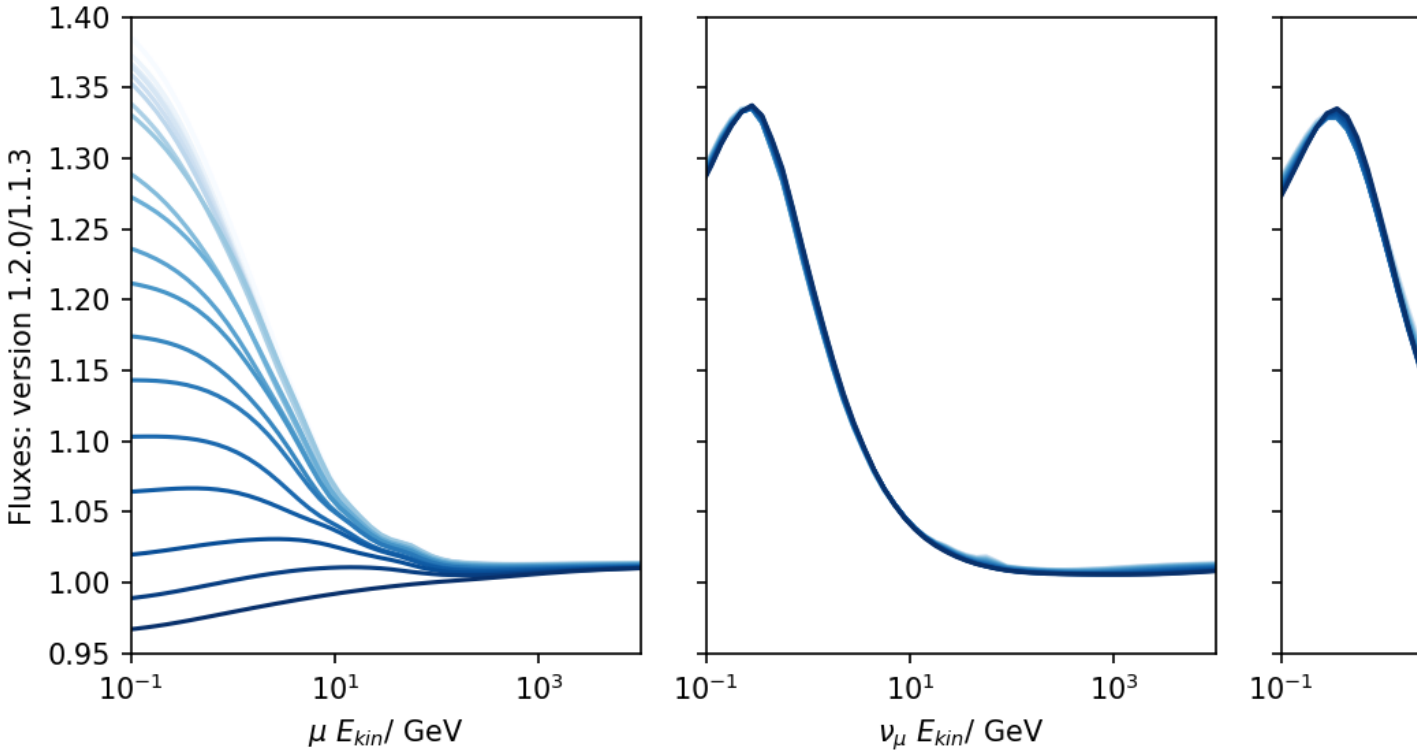
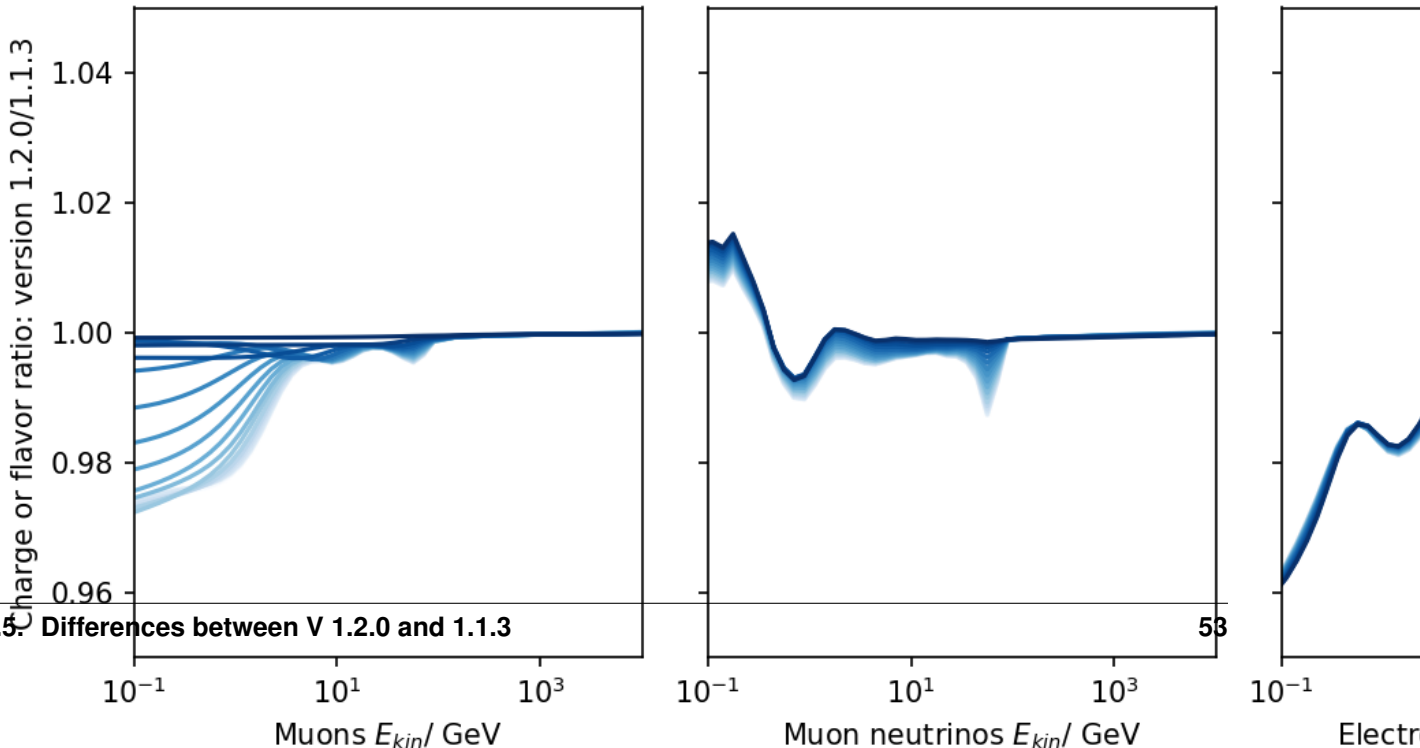


Fig. 2: Ratio of fluxes generated with the H3a primary model and EPOS-LHC between the versions 1.2.0 and 1.1.3 (from left to right fo
are used for different zenith angles.



mula for the boost discovered by Matthias Huber, thx. The effect is strongest at low energies as seen in the plots. At high energies there are no changes. For fluxes the changes are most striking in the zenith distribution of muons. For neutrinos the effect is mostly related to the spectral index. For electron neutrinos there is some effect for the zenith distribution at tens of GeV and will affect predictions made for IceCube DeepCore or KM3Net-ORCA. Update and recomputation of expectations is therefore recommended. For high energies, i.e. IceCube/P-ONE/ARCA recomputation is not necessary.

8.5.2 Muons in air showers

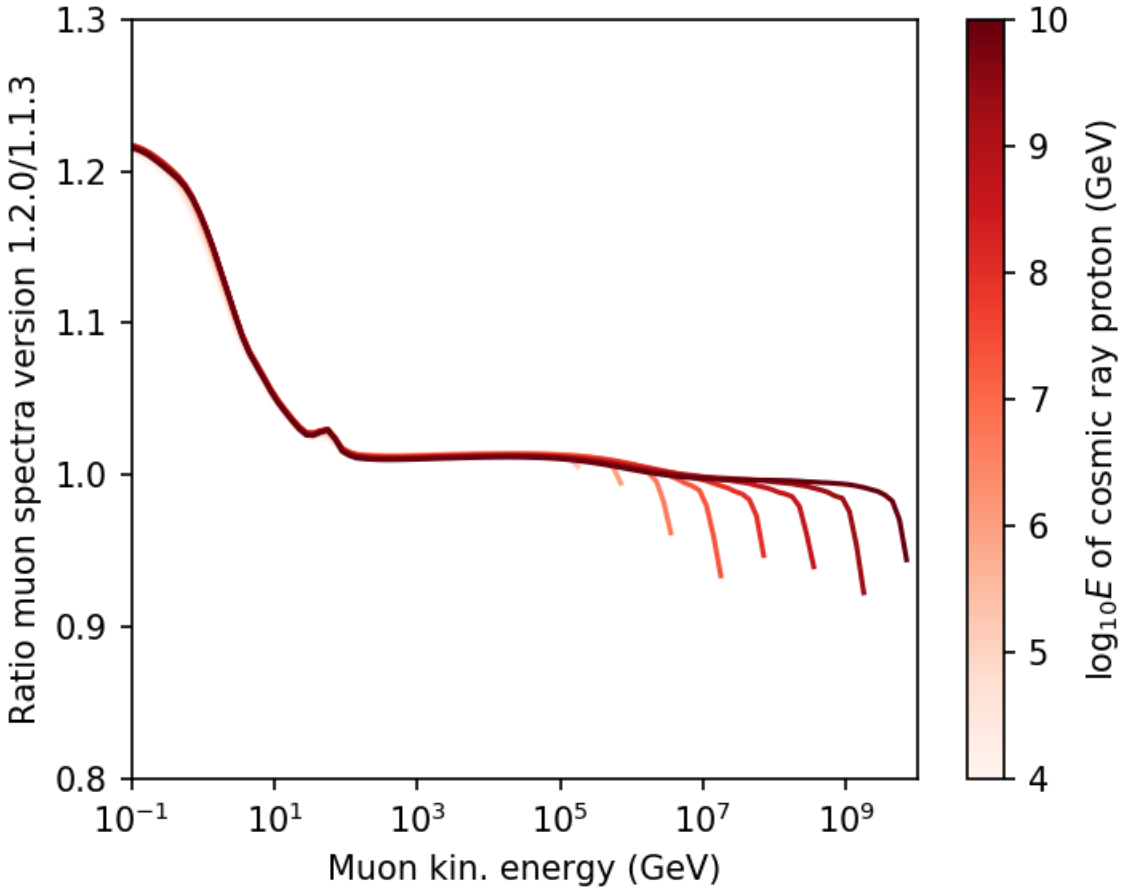
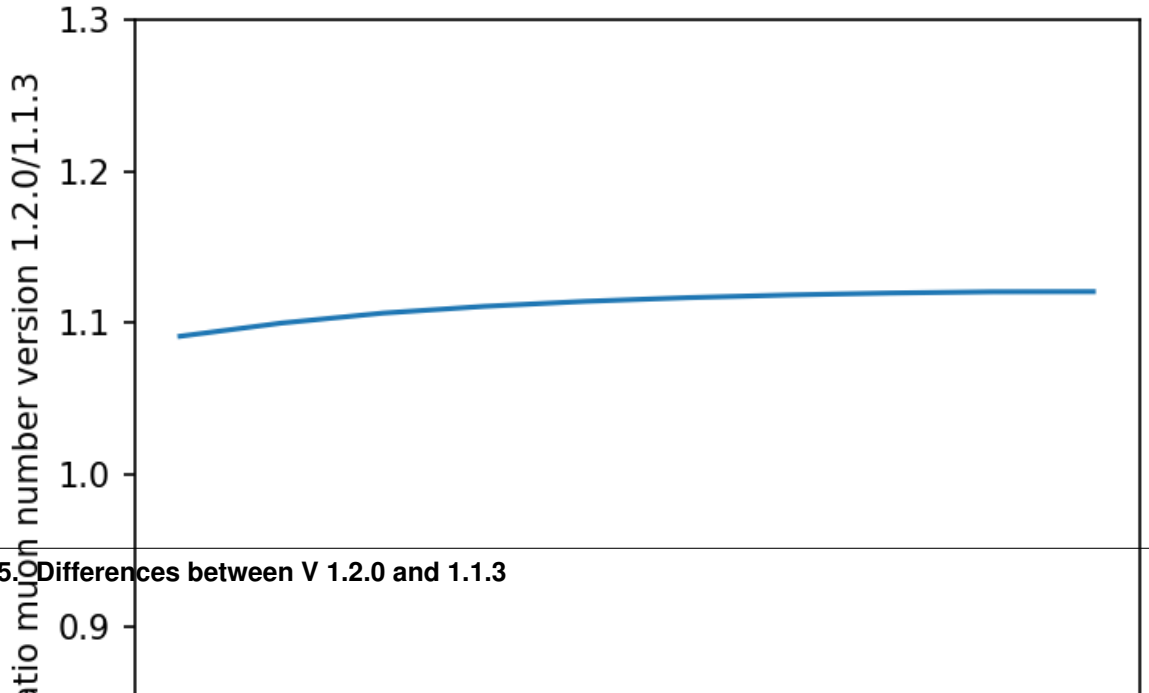


Fig. 4: Ratio of muon spectra for single primaries generated with EPOS-LHC between the versions 1.2.0 and 1.1.3 for vertical zenith angle. Shades are used for different primary energies.



For MCEq computations for a single primary, the resulting spectra

constitute the spectrum of particles in air showers. There is a ~10% change for the muon number.

8.5.3 Other changes

Some particle "mappings" have been updated and synchronized between the development version of CORSIKA 8. This has minor impact on the hadron ratios within the cascade and can lead to percent/sub-percent changes here and there.

8.5.4 Acknowledgements

The author is grateful to Matthias Huber (TUM) for helping with the discovery of the "decay bug". And to Maximilian Reininghaus and Ralf Ulrich (KIT) for making thorough and very detailed cross checks with various versions of CORSIKA.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

X

Z