PriNCe Documentation

Release 0.6.0

Anatoli Fedynitch

Sep 12, 2023

Contents

Release 0.6.0

Date Sep 12, 2023

PriNCe is derived from **Pr**opagation **i**ncluding **N**uclear **C**ascade **e**quations. This code is to be a numerical solver of the transport equations for Ultra-High Energy Cosmic Rays through the radiation fields of the intergalactic space. Its specific features are:

- **Time dependent UHECR transport equation solver**; efficient enough to compute a single spectrum within seconds,
- **Fast and easy variation of input parameters**; such as cross section models and extragalactic photon backgrounds,
- **Accessibility and modularity**; ability to easily modify and extend specific parts of the code through interfaces.

To achieve these goals, PriNCe is written in pure Python using vectorized expressions for the performance intensive parts. It accelerates those using libraries like Numpy and Scipy.

Data and other requirements

The code requires data files, which are bundled with this package in binary form. On first import around ~150MB of data will be automatically downloaded and placed in the data directory.

These data are constructed from a variety of sources. The code is publicly available in a different repository called PriNCe-data-utils.

CHAPTER 1

System requirements

- ~16GB RAM
- several GB disk space

The package is portable and will run on any flavor of Linux, Mac OS X and Windows that satisfy the dependencies below. Due to memory requirements 32-bit architectures are not recommended but may work under certain circumstances.

The code is pure Python relies on vectorization via numpy/scipy, and optionally Intel's MKL and nVidia's CUDA (via cupy).

Dependencies:

- Python 3 (2.7 is not supported.)
- numpy
- scipy
- matplotlib
- tqdm
- jupyter (optional, but needed for examples)

Optional:

- mkl (Intel's MKL runtime from pip/anaconda works)
- cupy (For GPU acceleration with CUDA. Support is experimental and memory requirements are not yet understood. Tested on RTX 2080TI/11GB, but typical memory requirements should not exceed 3-4 GB.)

CHAPTER 2

Installation

All you need is a python distribution including *pip*. For scientific computing [Anaconda/Miniconda](https://www.anaconda.com/products/individual/) may be good choice, but not necessary.

The installation via PyPi is the simplest method:

```
pip install prince-cr
```

To install from source:

```
git clone https://github.com/joheinze/PriNCe
cd PriNCe
pip install -e .
```

with the option *-e* (or *–editable*) this will link the local foulder to your pip installation, such that any local code edits will take effect. Use this for testing and development.

To test an installation:

    pip install pytest python -m pytest –pyargs prince_cr

CHAPTER 3

Examples

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

CHAPTER 4

Citations

If you use PriNCe in your scientific publications, please cite the code **AND** the physical models. Have a look at the *References* for instructions how to cite.

CHAPTER 5

Main documentation

5.1 Tutorial

The main interface to the cde is provided by the class *prince_cr.core.PriNCeRun*. To create an instance type:

```
from prince_cr.core import PriNCeRun
prince_run = PriNCeRun()
```

This assume will assume the Talys disintegration model at low energies and a superposition model based on Sophia at for photo-meson production at high energies. The photon field will be the CMB and the Gilmore et. al. CIB model.

prince_run will take a few minutes to instanciate. This oject can be saved using *pickle* to skip this step.

5.1.1 Changing model assumption

Either of these assumptions can be changed by manually creating objects:

```
from prince_cr import photonfields
pf = photonfields.CombinedPhotonField(
                [photonfields.CMBPhotonSpectrum,
                photonfields.CIBDominguez2D])

from prince_cr import cross_sections
cs = cross_sections.CompositeCrossSection([(0., cross_sections.TabulatedCrossSection,␣
↪('peanut_IAS',)),
                                           (0.14, cross_sections.SophiaSuperposition,␣
↪())])
```

The numbers in the arguments to `prince_cr.cross_sections.CompositeCrossSection` define the energies (in GeV) above which the models are joined. In this case Peanut will be used at low energies (greater then 0) and Sophia for energies above 0.14 GeV

These objects then need to be passed to *prince_cr.core.PriNCeRun*:

```
from prince_cr.core import PriNCeRun
prince_run_talys = core.PriNCeRun(max_mass = 56, photon_field=pf, cross_sections=cs)
```

Can be used to set a maximal nucleus mass. All heavier nuclei will be ignored, which can lead to a very significant speedup. Setting this option is equivalent to setting the same option in the config:

```
from prince_cr.config import
config.max_mass = 14
```

The config is a simple dictionary, see *prince/config.py* for all available options. Most settings need to be set before creating other objects, such as *PriNCeRun*

5.1.2 Solving the Transport equation

To solve the transport equation, create an instance of `UHECRPropagationSolverBDF`:

```
from prince_cr.solvers import UHECRPropagationSolverBDF
solver = UHECRPropagationSolverBDF(initial_z=1., final_z = 0.,prince_run=prince_run,
                                   enable_pairprod_losses = True, enable_adiabatic_
↪losses = True)
```

This will use Backward Differentiation to solve the transport equation from redshift 1 to 0. The Switches can be used to enable or disable interactions. There are no UHECRs injected into the system yet. Add a source class by:

```
rmax = 1e11
gamma = 1.2

from prince_cr.cr_sources import AugerFitSource,
solver.add_source_class(
    AugerFitSource(prince_run, norm = 1e-50, params={1407: (gamma, rmax, 1.)}))
```

This will inject pure nitrogen with maximal rigidity 1e11 GV and a spectral index of E^-1.2. You can add multiple sources by repeated calls to *add_source_class*. However note that this will get slow for a large number of sources.

Finally solve the system by calling:

```
solver.solve(dz=1e-3,verbose=False,progressbar=True)
```

The result is available as *solver.res*, which is an instance of `UHECRPropagationResults` and contains several convienience functions for accessing the propagated spectra (see full documentation or examples).

5.1.3 Custom UHECR sources

In the above examples we injected a pure nitrogen source. To inject a mix of elements, provide more keys to the *params* argument:

```
rmax = 1e11
gamma = 1.2

f_hydrogen = 0.
f_helium = 67.3
f_nitrogen = 28.1
f_silicon = 4.6
f_iron = 0.
```

(continues on next page)

(continued from previous page)

```
from prince_cr.cr_sources import AugerFitSource,
solver.add_source_class(prince_run, norm = total_norm,
                params={101: (gamma, rmax, f_hydrogen),
                        402: (gamma, rmax, f_helium),
                        1407: (gamma, rmax, f_nitrogen),
                        2814: (gamma, rmax, f_silicon),
                        5626: (gamma, rmax, f_iron)}))
```

gamma and *rmax* can also be defined separately for each element.

The spectral shape is defined by the source class (in this case *prince_cr.cr_sources.AugerFitSource*) *prince_cr.cr_sources* contains several other predefined classes. You can also define your own source class by subclassing `prince_cr.cr_sources.CosmicRaySource` and implementing `CosmicRaySource.injection_spectrum()`.

5.2 References

5.2.1 The PriNCe code

Whenever publishing results, we would like ask for a citation of the original paper that contains the description of the code in the appendix.

> *A new view on Auger data and cosmogenic neutrinos in light of different nuclear disintegration and air-shower models*
> J. Heinze, A. Fedynitch, D. Boncioli and W. Winter
> Astrophys.J. 873 (2019) no.1, 88
> https://doi.org/10.3847/1538-4357/ab05ce

In case the code mentioned on your slides or internal notes, please advertise the GitHub page https://github.com/joheinze/PriNCe.

5.2.2 Physical models

The models for nuclear interactions and decays are parameterizations of the models that we did not make. The default settings will use a combination of the TALYS code for nuclear interactions and the SOPHIA event generator to run.

- **Nuclear photo-disintegration models:**
  - *TALYS: Comprehensive Nuclear Reaction Modeling*
    A.J. Koning, S. Hilaire, M.C. Duijvestijn
    AIP Conf.Proc. 769 (2005) 1, 1154
    International Conference on Nuclear Data for Science and Technology (ND2004), 1154
    DOI: 10.1063/1.1945212
  - *PEANUT* is part of FLUKA:
    FLUKA: A multi-particle transport code (Program version 2005)
    Alfredo Ferrari, Paola R. Sala, Alberto Fasso, Johannes Ranft
    Report number: CERN-2005-010, SLAC-R-773, INFN-TC-05-11, CERN-2005-10
    DOI: 10.2172/877507

  - *PSB: Photonuclear Interactions of Ultrahigh-Energy Cosmic Rays and their Astrophysical Consequences*
    Puget, J. L. and Stecker, F. W. and Bredekamp, J. H.
    Astrophys. J. 205 (1976)

- Photo-hadronic interactions and photo-pion production:
  - *SOPHIA: Monte Carlo simulations of photohadronic processes in astrophysics*
    A. Mucke, Ralph Engel, J.P. Rachen, R.J. Protheroe, Todor Stanev
    Comput.Phys.Commun. 124 (2000) 290-314
    astro-ph/9903478 [astro-ph]
    DOI: 10.1016/S0010-4655(99)00446-4
  - *Improved photomeson model for interactions of cosmic ray nuclei*
    Leonel Morejon, Anatoli Fedynitch, Denise Boncioli, Daniel Biehl, Walter Winter
    JCAP 11 (2019) 007
    arXiv:1904.07999
    DOI: 10.1088/1475-7516/2019/11/007 (publication)
    AstroPhoMes code
- **Air-shower models (if used in comparisons with Xmax):**

  The hadronic interaction model Sibyll 2.3c and extensive air showers
  Felix Riehn, Ralph Engel, Anatoli Fedynitch, Thomas K. Gaisser, Todor Stanev
  arXiv:1912.03300

  Monte Carlo treatment of hadronic interactions in enhanced Pomeron scheme: I. QGSJET-II model
  Sergey Ostapchenko
  Phys.Rev. D83 (2011) 014018, arXiv:1010.1869

5.3 Advanced documentation

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

- `prince_cr.config` – *default configuration options*
- `prince_cr.core` – *Core module*
- `prince_cr.data` – *(particle) Species management*
- `prince_cr.solvers` – *PDE solver implementations*
- `prince_cr.cross_sections` – *Cross section data management*
- `prince_cr.interaction_rates` – *Computation of Interaction Matrices*
- `prince_cr.photonfields` – *EBL and CMB photon fields*
- `prince_cr.cr_sources` – *UHECR source class definitions*

5.3.1 prince_cr.config – default configuration options

The module contains all static options of PriNCe that are not meant to be changed in run time.

PriNCe configuration module.

prince_cr.config.**E_CMB = 2.34823e-13**
CMB energy kB*T0 [GeV]

prince_cr.config.**H_0 = 70.5**
Hubble constant

prince_cr.config.**Omega_Lambda = 0.73**
Omega_Lambda

prince_cr.config.**Omega_m = 0.27**
Omega_m

prince_cr.config.**cosmic_ray_grid = (3, 14, 8)**
Cosmic ray energy grid (defines system size for solver) Number of bins in multiples of 4 recommended for maximal vectorization efficiency for 256 bit AVX or similar Format (log10(E_min), log10(E_max), nbins/decade of energy)

prince_cr.config.**data_dir = '/home/docs/checkouts/readthedocs.org/user_builds/prince/checko**
Directory where the data files for the calculation are stored

prince_cr.config.**db_fname = 'prince_db_05.h5'**
PrinceDB file name

prince_cr.config.**debug_level = 1**
Debug flag for verbose printing, 0 silences PriNCe entirely

prince_cr.config.**grid_scale = 'E'**
Scale of the energy grid 'E': logarithmic in energy E_i = E_min * (Delta)^i 'logE': linear grid in x = log_10(E): x_i = x_min + i * Delta

prince_cr.config.**override_debug_fcn = []**
Printout debug info only for functions in this list (just give the name, "get_solution" for instance) Warning, this option slows down initialization by a lot. Use only when needed.

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

prince_cr.config.**pf = 'Linux-5.19.0-1028-aws-x86_64-with-debian-buster-sid'**
determine shared library extension and MKL path

prince_cr.config.**photon_grid = (-15, -6, 8)**
Photon grid of target field, only for calculation of rates

prince_cr.config.**print_module = False**
Print module name in debug output

prince_cr.config.**redist_fname = 'sophia_redistribution_logbins.npy'**
Model file for redistribution functions (from SOPHIA or similar)

prince_cr.config.**redist_threshold_ID = 101**
Particle ID for which redistribution functions are needed to be taken into account. The default value is 101 (proton). All particles with smaller IDs, i.e. neutrinos, pions, muons etc., will have energy redistributions. For larger IDs (nuclei) the boost conservation is employed.

prince_cr.config.**semi_lagr_method = '5th_order'**
Order of semi-lagrangian for energy derivative

prince_cr.config.**tau_dec_threshold = inf**
Threshold lifetime value for explicit transport of particles of this type. It means that if a particle is unstable with lifetime smaller than this threshold, it will be decayed until all final state particles of this chain are stable. In other words: short intermediate states will be integrated out

prince_cr.config.**x_cut = 0.0001**
Cut on energy redistribution functions Resitribution below this x value are set to 0. "x_cut" : 0., "x_cut_proton" : 0.,

prince_cr.config.**y_cut = inf**
cut on photon energy, cross section above y = E_cr e_ph / m_cr does not contribute

5.3.2 prince_cr.core – Core module

This module contains the main program features. Instantiating *prince_cr.core.PriNCeRun* 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.

Provides user interface and runtime management.

class prince_cr.core.**PriNCeRun**(**args*, ***kwargs*)
This is a draft of the main class.

This class is supposed to interprete the config options and initialize all necessary stuff in order. This class is meant to keep all separate ingredients together in one place, and it is inteded to be passed to further classes via *self*.

5.3.3 prince_cr.data – (particle) Species management

The *prince_cr.data.SpeciesManager* handles the bookkeeping of :class:'prince_cr.data.PrinceSpecies's.

Module inteded to contain some prince-specific data structures.

class prince_cr.data.**EnergyGrid**(*lower*, *upper*, *bins_dec*)
Class for constructing a grid for discrete distributions.

Since we discretize everything in energy, the name seems appropriate. All grids are log spaced.

Parameters

- **lower** (*float*) – log10 of low edge of the lowest bin
- **upper** (*float*) – log10 of upper edge of the highest bin
- **bins_dec** (*int*) – bins per decade of energy

class prince_cr.data.**PrinceDB**
Provides access to data stored in an HDF5 file.

The file contains all tables for runnin PriNCe. Currently the only still required file is the particle database. The tools to generate this database are publicly available in PriNCe-data-utils.

class prince_cr.data.**PrinceSpecies**(*ncoid*, *princeidx*, *d*)
Bundles different particle properties for simplified availability of particle properties in *prince_cr.core.PriNCeRun*.

Parameters

- **pdgid** (*int*) – PDG ID of the particle
- **particle_db** (*object*) – a dictionary with particle properties

- **d** (*int*) – dimension of the energy grid

static calc_AZN(*nco_id*)
Returns mass number A, charge Z and neutron number N of `nco_id`.

indices(*grid_tag='default'*)
Returns a list of all indices in the state vector.

Returns array of indices in state vector `PrinceRun.phi`

Return type (numpy.array)

lbin(*grid_tag='default'*)
Returns lower bin of particle range in state vector.

Returns lower bin in state vector `PrinceRun.phi`

Return type (int)

lidx(*grid_tag='default'*)
Returns lower index of particle range in state vector.

Returns lower index in state vector `PrinceRun.phi`

Return type (int)

ubin(*grid_tag='default'*)
Returns upper bin of particle range in state vector.

Returns upper bin in state vector `PrinceRun.phi`

Return type (int)

uidx(*grid_tag='default'*)
Returns upper index of particle range in state vector.

Returns upper index in state vector `PrinceRun.phi`

Return type (int)

A = None
Mass, charge, neutron number

N = None
Mass, charge, neutron number

Z = None
Mass, charge, neutron number

decay_channels = None
decay channels if any

has_redist = None
(bool) particle has an energy redistribution

is_alias = None
(bool) particle is an alias (PDG ID encodes special scoring behavior)

is_baryon = None
(bool) particle is a baryon

is_charged = None
(bool) particle is a lepton

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

is_hadron = None
(bool) particle is a hadron (meson or baryon)

is_lepton = None
(bool) particle is a lepton

is_meson = None
(bool) particle is a meson

is_nucleus = None
(bool) particle is a nucleus

is_stable = None
(bool) particle is stable

lifetime = None
(float) lifetime

mass = None
Mass in atomic units or GeV

ncoid = None
Neucosma ID of particle

princeidx = None
(int) Prince index (in state vector)

sl

Return the slice for this species on the grid can be used as spec[s.sl]

Returns a slice object pointing to the species in the state vecgtor

Return type (slice)

sname = None
(str) species name in string representation

class prince_cr.data.**SpeciesManager**(*ncoid_list*, *ed*)
Provides a database with particle and species.

add_grid(*grid_tag*, *dimension*)
Defines additional grid dimensions under a certain tag.

Propagates changes to this variable to all known species.

ncoid2princeidx = None
(dict) Converts Neucosma ID to index in state vector

nspec = None
(int) Total number of species

princeidx2ncoid = None
(dict) Converts index in state vector to Neucosma ID

princeidx2sref = None
(dict) Converts prince index to reference of class:*data.PrinceSpecies*

sname2princeidx = None
(dict) Converts particle name to index in state vector

sname2sref = None
(dict) Converts particle name to reference of class:*data.PrinceSpecies*

prince_cr.data.**db_handler = <prince_cr.data.PrinceDB object>**
db_handler is the HDF file interface

5.3.4 prince_cr.solvers – PDE solver implementations

Contains solvers to solve the coupled differential equation system

The steps performed by the solver are:

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

Contains solvers, i.e. integrators, kernels, steppers, for PriNCe.

class prince_cr.solvers.propagation.**UHECRPropagationResult**(*state*, *egrid*, *spec_man*)
Reduced version of solver class, that only holds the result vector and defined add and multiply

get_lnA(*nco_ids*, *egrid=None*)
Return the average ln(A) as a function of total energy for all elements in the range

get_solution(*nco_id*)
Returns the spectrum in energy per nucleon

get_solution_group(*nco_ids*, *epow=3*, *egrid=None*)
Return the summed spectrum (in total energy) for all elements in the range

get_solution_scale(*nco_id*, *epow=0*)
Returns the spectrum scaled back to total energy

class prince_cr.solvers.partial_diff.**SemiLagrangianSolver**(*cr_grid*)
Contains routines to project spectra from shifted grids back to old grid

interpolate(*conloss*, *state*)
Uses linear interpolation to find the new state old grid

interpolate_4thorder_weights(*conloss*, *state*)
Uses quadratic interpolation with lagrange polynomials

interpolate_5thorder_weights(*conloss*, *state*)
Uses cubic interpolation with lagrange polynomials

interpolate_cubic_weights(*conloss*, *state*)
Uses cubic interpolation with lagrange polynomials

interpolate_gradient(*conloss*, *state*)
Uses a linear approximation arround x_i to find the new state old grid

interpolate_linear_weights(*conloss*, *state*)
Uses linear interpolation with lagrange polynomials (same as interpolate, but directly implemented for testing)

interpolate_quadratic_weights(*conloss*, *state*)
Uses quadratic interpolation with lagrange polynomials

5.3.5 prince_cr.cross_sections – Cross section data management

Contains function to load and combined cross section models

5.3.6 prince_cr.interaction_rates – Computation of Interaction Matrices

Contains function to precompute the sparse Interaction matrices efficiently

The module contains classes for computations of interaction rates

class prince_cr.interaction_rates.**ContinuousAdiabaticLossRate**(*prince_run*, *energy='grid'*, **args*, ***kwargs*)

Implementation of continuous pair production loss rates.

loss_vector(*z*, *energy=None*)
Returns all continuous losses on dim_states grid

single_loss_length(*pid*, *z*)
Returns energy loss length in cm (convenience function for plotting)

spec_man = None
Reference to species manager

class prince_cr.interaction_rates.**ContinuousPairProductionLossRate**(*prince_run*, *energy='grid'*, **args*, ***kwargs*)

Implementation of continuous pair production loss rates.

loss_vector(*z*)
Returns all continuous losses on dim_states grid

photon_vector(*z*)
Returns photon vector at redshift *z* on photon grid.

This vector is in fact a matrix of vectors of the interpolated photon field with dimensions (dim_cr, xi_steps).

Parameters **z** (*float*) – redshift

Return value from cache if redshift value didn't change since last call.

single_loss_length(*pid*, *z*, *pfield=None*)
Returns energy loss length in cm (convenience function for plotting)

photon_field = None
Reference to PhotonField object

spec_man = None
Reference to species manager

class prince_cr.interaction_rates.**PhotoNuclearInteractionRate**(*prince_run=None*, *with_dense_jac=True*, **args*, ***kwargs*)

Implementation of photo-hadronic/nuclear interaction rates. This Version directly writes the data into a CSC-matrix and only updates the data each time.

get_hadr_jacobian(*z*, *scale_fac=1.0*, *force_update=False*)
Returns the nonel rate vector and coupling matrix.

photon_vector(*z*)
Returns photon vector at redshift *z* on photon grid.

This vector is in fact a matrix of vectors of the interpolated photon field with dimensions (dim_cr, xi_steps).

Parameters **z** (*float*) – redshift

Return value from cache if redshift value didn't change since last call.

single_interaction_length(*pid*, *z*, *pfield=None*)
Returns energy loss length in cm (convenience function for plotting)

cross_sections = None
Reference to CrossSection object

photon_field = None
Reference to PhotonField object

spec_man = None
Reference to species manager

5.3.7 prince_cr.photonfields – EBL and CMB photon fields

Contains different EBL models as a function of redshift

Created on Feb 22, 2017

@author: Anatoli Fedynitch

class prince_cr.photonfields.**CIBDominguez2D**(*model='base'*, *simple_scaling=False*)
CIB model "3" by Gilmore et al.

CIB photon distribution for z = 0...2. Requires availability of an *scipy.interp2d* object file *data/CIB_dominguez_int2D.ppo*.

Note: The class contains an interpolators for the upper and lower limits, which are not yet accessable through a function

Ref.: R.C. Gilmore et al., MNRAS 410, 2556 (2011) [arXiv:1104.0671]

class prince_cr.photonfields.**CIBFranceschini2D**(*simple_scaling=False*)
CIB model "1" by Fraceschini et al.

CIB photon distribution for z = 0...2. Requires availability of an *scipy.interp2d* object file *data/CIB_franceschini_int2D.ppo*.

Ref.:

A. Franceschini et al., Astron. Astrphys. 487, 837 (2008) [arXiv:0805.1841]

class prince_cr.photonfields.**CIBFranceschiniZ0**
CIB model "1" by Fraceschini et al.

CIB photon distribution at z=0.

Ref.:

A. Franceschini et al., Astron. Astrphys. 487, 837 (2008) [arXiv:0805.1841]

get_photon_density(*E*, *z*)
Returns the redshift-scaled number density of CMB photons

Parameters

- **z** (*float*) – redshift
- **E** (*float*) – photon energy (GeV)

Returns CMB photon spectrum in $\mathrm{GeV}^{-1}\mathrm{cm}^{-3}$

Return type float

class prince_cr.photonfields.**CIBGilmore2D**(*model='fiducial'*, *simple_scaling=False*)
CIB model "3" by Gilmore et al.

CIB photon distribution for z = 0...7. Requires availability of an *scipy.interp2d* object file *data/CIB_gilmore_int2D.ppo*.

Note: Currently uses the fixed model from the reference as standard, for the fiducial model, change the 'model' keyword

Ref.: R.C. Gilmore et al., MNRAS Soc. 422, 3189 (2012) [arXiv:1104.0671]

class prince_cr.photonfields.**CIBInoue2D**(*model='base'*, *simple_scaling=False*)
CIB model "2" by Inoue et al.

CIB photon distribution for z = 0...10. Requires availability of an *scipy.interp2d* object file *data/CIB_inoue_int2D.ppo*. A low and high variation of the "third-population" component are also available, by passing

Ref.:

Y. Inoue et al. [arXiv:1212.1683]

class prince_cr.photonfields.**CIBSteckerZ0**
CIB model "1" by Stecker et al.

CIB photon distribution at z=0.

Ref.: F.W. Stecker et al., Astrophys. J. 648, 774 (2006) [astro-ph/0510449]

get_photon_density(*E*, *z*)
Returns the redshift-scaled number density of CMB photons

Parameters

- **z** (*float*) – redshift
- **E** (*float*) – photon energy (GeV)

Returns CMB photon spectrum in $\mathrm{GeV}^{-1}\mathrm{cm}^{-3}$

Return type float

class prince_cr.photonfields.**CMBPhotonSpectrum**
Redshift-scaled number density of CMB photons

In the CMB frame (equivalent to the observer's frame). Normalisation from Planck's spectrum. The scaling goes as $n(E, z) = (1 + z)^3 n(E/(1 + z), z = 0)$. The CMB spectrum is a blackbody spectrum with the present-day temperature T0 = 2.725 K.

Ref.:

M. Ahlers, L.A. Anchordoqui, and S. Sarkar, Phys. Rev. D 79, 083009 (2009) [0902.3993]

get_photon_density(*E*, *z*)
Returns the redshift-scaled number density of CMB photons

Parameters

- **z** (*float*) – redshift
- **E** (*float*) – photon energy (GeV)

Returns CMB photon spectrum in $\mathrm{GeV}^{-1}\mathrm{cm}^{-3}$

Return type float

class prince_cr.photonfields.**CombinedPhotonField**(*list_of_classes_and_args*)
Class to combine (sum) several models, which inherit from *PhotonField*.

This class is useful when constructing a realistic photon spectrum, which is typically a superposition of CMB and CIB. The list of models can be passed to the constructor or each model can be added separately using the *add_model()*.

Parameters **list_of_classes_and_args** – Can be either list of classes or list of tuples (class, args)

add_model(*model_class*, *model_args=()*)
Adds a model class to the combination.

get_photon_density(*E*, *z*)
Returns the redshift-scaled number density of photons as a sum of different models.

Parameters

- **z** (*float*) – redshift
- **E** (*float*) – photon energy (GeV)

Returns CMB photon spectrum in $\mathrm{GeV}^{-1}\mathrm{cm}^{-3}$

Return type float

class prince_cr.photonfields.**EBLSplined2D**

get_photon_density(*E*, *z*)
Returns the redshift-scaled number density of CIB photons

Accepts scalar, vector and matrix arguments.

Parameters

- **z** (*float*) – redshift
- **E** (*float*) – photon energy (GeV)

Returns CMB photon spectrum in $\mathrm{GeV}^{-1}\mathrm{cm}^{-3}$

Return type float

class prince_cr.photonfields.**FlatPhotonSpectrum**
Constant photon density for testing.

get_photon_density(*E*, *z*)
Returns the redshift-scaled number density of CMB photons

Parameters

- **z** (*float*) – redshift
- **E** (*float*) – photon energy (GeV)

Returns CMB photon spectrum in $\mathrm{GeV}^{-1}\mathrm{cm}^{-3}$

Return type float

class prince_cr.photonfields.**PhotonField**
Base class for constructing target photon densities.

Derived classes have to implement the method `PhotonField.get_photon_densities()`.

5.3.8 `prince_cr.cr_sources` – UHECR source class definitions

Defines the (simple) source injection for the extragalactic propagation

Defines source models for cosmic ray propagation

The standard interface requires to UHECRSolvers requires that each source defines a methods injection_rate(self, z)

class prince_cr.cr_sources.**AugerFitSource**(*prince_run*, *ncoids=None*, *params=None*, *norm=1.0*, *m='flat'*, **args*, ***kwargs*)

Simple source class with spectral index and rigidity dependent cutoff Defined to be parrallel for all species below the cutoff as in Auger Combined Fit paper

if E < Z * Rcut: inj(E) = norm * E**-gamma

else: inj(E) = norm * E**-gamma * exp(- E / Z * Rcut)

params defined as {pid: gamma, Rcut, norm}

injection_spectrum(*pid*, *energy*, *params*)

Prototype to be defined in each child class

class prince_cr.cr_sources.**RigdityCutoffSource**(*prince_run*, *ncoids=None*, *params=None*, *norm=1.0*, *m='flat'*, **args*, ***kwargs*)

Simple source class with spectral index and rigidity dependent cutoff

inj(E) = norm * E**-gamma * exp(- E / Z * Rcut)

params defined as {pid: gamma, Rcut, norm}

injection_spectrum(*pid*, *energy*, *params*)

Prototype to be defined in each child class

class prince_cr.cr_sources.**RigidityFlexSource**(*prince_run*, *ncoids=None*, *params=None*, *norm=1.0*, *m='flat'*, **args*, ***kwargs*)

Simple source class with spectral index and rigidity dependent cutoff Parameter alpha to scaled the rigidity dependence

inj(E) = norm * E**-gamma * exp(- E / Z**alpha * Rcut)

params defined as {pid: gamma, Rcut, alpha, norm}

injection_spectrum(*pid*, *energy*, *params*)

Prototype to be defined in each child class

class prince_cr.cr_sources.**SimpleSource**(*prince_run*, *ncoids=None*, *params=None*, *norm=1.0*, *m='flat'*, **args*, ***kwargs*)

Simple source class with spectral index and cutoff

inj(E) = norm * E**-gamma * exp(- E / Emax)

params defined as {pid: gamma, Emax, norm}

injection_spectrum(*pid*, *energy*, *params*)

Prototype to be defined in each child class

class prince_cr.cr_sources.**SpectrumSource**(*prince_run*, *ncoids=None*, *params=None*, *norm=1.0*, *m='flat'*, **args*, ***kwargs*)

Source class with the spectrum defined externally by an array The spectrum might be interpolated as needed

params defined as {pid: egrid, specgrid}

injection_spectrum(*pid*, *energy*, *params*)

Prototype to be defined in each child class

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

R

S

T

U

X

Y

Z