PyREx Documentation

Release 1.0.0

Ben Hokanson-Fasig

Aug 25, 2017

CONTENTS:

CHAPTER

ONE

INTRODUCTION

PyREx (**Py**thon package for an IceCube **R**adio **Ex**tension) is, as its name suggests, a python package designed to simulate the measurement of Askaryan pulses via a radio antenna array around the IceCube South Pole Neutrino Observatory. The code is designed to be modular so that it can also be applied to other askaryan radio antennas, as in the ARA and ARIANA collaborations.

The most basic simulation can be produced as follows:

First, import the package:

```
import pryex
```

Then, create a particle generator object that will produce random particles in a cube of 1 km on each side with a fixed energy of 100 PeV:

```
particle_generator = pyrex.ShadowGenerator(dx=1000, dy=1000, dz=1000, egen=lambda:␣
↪1e8)
```

An array of antennas that represent the detector is also needed. The base Antenna class provides a basic antenna with a flat frequency response and no trigger condition. Here we make a single vertical "string" of four antennas with no additional noise:

```
antenna_array = []
for z in [-100, -150, -200, -250]:
    antenna_array.append(
        pyrex.Antenna(position=(0,0,depth), temperature=300, freq_range=(1e6,1e9),␣
↪noisy=False)
    )
```

Finally, we want to pass these into the EventKernel and produce an event:

```
kernel = pyrex.EventKernel(generator=particle_generator, ice_model=pyrex.IceModel,␣
↪antennas=antenna_array)
kernel.event()
```

Now the signals received by each antenna can be accessed by their waveforms parameter:

```
import matplotlib.pyplot as plt
for ant in kernel.antennas:
    for wave in ant.waveforms:
        plt.figure()
        plt.plot(wave.times, wave.values)
        plt.show()
```

CHAPTER

TWO

CODE EXAMPLES

For more code examples, see the PyREx Demo python notebook.

CHAPTER

THREE

PYREX API

3.1 Package contents

class pyrex.**Signal**(*times*, *values*)
Base class for signals. Takes arrays of times and values (values array forced to size of times array by zero padding or slicing). Supports adding between signals with the same time values, resampling the signal, and calculating the signal's envelope.

dt
Returns the spacing of the time array, or None if invalid.

envelope
Calculates envelope of the signal by Hilbert transform.

resample(*n*)
Resamples the signal into n points in the same time range.

spectrum
Returns the FFT spectrum of the signal.

frequencies
Returns the FFT frequencies of the signal.

filter_frequencies(*freq_response*)
Applies the given frequency response function to the signal.

class pyrex.**EmptySignal**(*times*)
Bases: *pyrex.signals.Signal*

Class for signal with no amplitude (all values = 0)

class pyrex.**FunctionSignal**(*times*, *function*)
Bases: *pyrex.signals.Signal*

Class for signals generated by a function

pyrex.**AskaryanSignal**
alias of FastAskaryanSignal

class pyrex.signals.**FastAskaryanSignal**(*times*, *energy*, *theta*, *n=1.78*, *t0=0*)
Bases: *pyrex.signals.Signal*

Askaryan pulse binned to times from neutrino with given energy (TeV) observed at angle theta (radians). Optional parameters are the index of refraction n, and pulse offset to start time t0 (s). Returned signal values are electric fields (V/m).

Note that the amplitude of the pulse goes as 1/R, where R is the distance from source to observer. R is assumed to be 1 meter so that dividing by a different value produces the proper result.

vector_potential
Recover the vector_potential from the electric field. Mostly just for testing purposes.

RAC(*time*)
Calculates R * vector potential (A) at the Cherenkov angle in Vs at the given time (s).

charge_profile(*z*, *density=0.92*, *crit_energy=7.86e-05*, *rad_length=36.08*)
Calculates the longitudinal charge profile in the EM shower at distance z (m) with parameters for the density (g/cm^3), critical energy (TeV), and electron radiation length (g/cm^2) in ice.

max_length(*density=0.92*, *crit_energy=7.86e-05*, *rad_length=36.08*)
Calculates the maximum length (m) of an EM shower with parameters for the density (g/cm^3), critical energy (TeV), and electron radiation length (g/cm^2) in ice.

class pyrex.**ThermalNoise**(*times*, *temperature*, *resistance*, *f_band*, *f_amplitude=1*, *n_freqs=0*)
Bases: *pyrex.signals.Signal*

Thermal Rayleigh noise at a given temperature (K) and resistance (ohms) in the frequency band f_band=[f_min,f_max] (Hz). Optional parameters are f_amplitude (default 1) which can be a number or a function designating the amplitudes at each frequency, and n_freqs which is the number of frequencies to use (in f_band) for the calculation (default is based on the FFT bin size of given times array). Returned signal values are voltages (V).

class pyrex.**Antenna**(*position*, *temperature*, *freq_range*, *resistance=None*, *noisy=True*)
Base class for an antenna with a given position (m), temperature (K), allowable frequency range (Hz), total resistance (ohm) used for Johnson noise, and whether or not to include noise in the antenna's waveforms. Defines default trigger, frequency response, and signal reception functions that can be overwritten in base classes to customize the antenna.

is_hit
Test for whether the antenna has received a signal.

isHit()
Deprecated. Replaced by is_hit property.

clear()
Reset the antenna to a state of having received no signals.

waveforms
Signal + noise (if noisy) at each triggered antenna hit.

all_waveforms
Signal + noise (if noisy) at all antenna hits, even those that didn't trigger.

trigger(*signal*)
Function to determine whether or not the antenna is triggered by the given Signal object.

response(*frequencies*)
Function to return the frequency response of the antenna at the given frequencies (Hz). This function should return the response as imaginary numbers, where the real part is the amplitude response and the imaginary part is the phase response.

receive(*signal, polarization=[0, 0, 1]*)
Process incoming signal according to the filter function and store it to the signals list. Subclasses may extend this fuction, but should end with super().receive(signal).

class pyrex.**DipoleAntenna**(*name, position, center_frequency, bandwidth, resistance, effective_height, polarization=[0, 0, 1], trigger_threshold=0, noisy=True*)
Bases: *pyrex.antenna.Antenna*

Antenna with a given name, position (m), center frequency (MHz), bandwidth (MHz), resistance (ohm), effective height (m), polarization direction, and trigger threshold (V).

trigger(*signal*)
Trigger on the signal if the maximum signal value is above the given threshold.

response(*frequencies*)
Butterworth filter response for the antenna's frequency range.

receive(*signal, polarization=[0, 0, 1]*)
Apply polarization effect to signal, then proceed with usual antenna reception.

pyrex.**IceModel**
alias of AntarcticIce

class pyrex.ice_model.**AntarcticIce**
Bases: object

Class containing characteristics of ice at the south pole.

k = 0.438

a = 0.0132

n0 = 1.32

thickness = 2850

classmethod gradient(*z*)
Returns the gradient of the index of refraction at depth z (m).

classmethod index(*z*)
Returns the medium's index of refraction, n, at depth z (m).

static temperature(*z*)
Returns the temperature (K) of the ice at depth z (m)

classmethod attenuation_length(*z*, *f*)
Returns the attenuation length at depth z (m) and frequency f (MHz).

pyrex.**prem_density**(*r*)
Returns the earth's density (g/cm^3) for a given radius r (m). Calculated by the Preliminary Earth Model (PREM).

pyrex.**slant_depth**(*angle*, *depth*, *step=5000*)
Returns the material thickness (g/cm^2) for a chord cutting through earth at Nadir angle and starting at depth (m).

class pyrex.**Particle**
Named tuple for containing particle attributes. Consists of a 3-D vertex (m), 3-D direction vector, and an energy (GeV).

direction

energy

vertex

class pyrex.**ShadowGenerator**(*dx*, *dy*, *dz*, *energy_generator*)
Class to generate UHE neutrino vertices in (relatively) shallow detectors. Takes into accout Earth shadowing (sort of). energy_generator should be a function that returns a particle energy in GeV.

create_particle()
Creates a particle with random vertex in cube and random direction.

class pyrex.**PathFinder**(*ice_model*, *from_point*, *to_point*)
Class for ray tracking.

exists
Boolean of whether path exists.

emitted_ray
Direction in which ray is emitted.

path_length
Length of the path (m).

tof
Time of flight (s) for a particle along the path. Calculated using default values of self.time_of_flight()

time_of_flight (*n_steps=10*)
Time of flight (s) for a particle along the path.

attenuation (*f*, *n_steps=10*)
Returns the attenuation factor for a signal of frequency f (Hz) traveling along the path.

propagate (*signal*)
Applies attenuation to the signal along the path.

class pyrex.**EventKernel** (*generator*, *ice_model*, *antennas*)
Kernel for generation of events with a given particle generator, ice model, and list of antennas.

event ()
Generate particle, propagate signal through ice to antennas, process signal at antennas, and return the original particle.

3.2 Submodules

3.2.1 pyrex.signals module

Module containing classes for digital signal processing

class pyrex.signals.**Signal** (*times*, *values*)
Bases: object

Base class for signals. Takes arrays of times and values (values array forced to size of times array by zero padding or slicing). Supports adding between signals with the same time values, resampling the signal, and calculating the signal's envelope.

dt
Returns the spacing of the time array, or None if invalid.

envelope
Calculates envelope of the signal by Hilbert transform.

resample (*n*)
Resamples the signal into n points in the same time range.

spectrum
Returns the FFT spectrum of the signal.

frequencies
Returns the FFT frequencies of the signal.

filter_frequencies (*freq_response*)
Applies the given frequency response function to the signal.

class pyrex.signals.**EmptySignal**(*times*)
Bases: *pyrex.signals.Signal*

Class for signal with no amplitude (all values = 0)

class pyrex.signals.**FunctionSignal**(*times*, *function*)
Bases: *pyrex.signals.Signal*

Class for signals generated by a function

class pyrex.signals.**SlowAskaryanSignal**(*times*, *energy*, *theta*, *n=1.78*, *t0=0*)
Bases: *pyrex.signals.Signal*

Askaryan pulse binned to times from neutrino with given energy (TeV) observed at angle theta (radians). Optional parameters are the index of refraction n, and pulse offset to start time t0 (s). Returned signal values are electric fields (V/m).

Note that the amplitude of the pulse goes as 1/R, where R is the distance from source to observer. R is assumed to be 1 meter so that dividing by a different value produces the proper result.

RAC(*time*)
Calculates R * vector potential at the Cherenkov angle in Vs at the given time (s).

charge_profile(*z*, *density=0.92*, *crit_energy=7.86e-05*, *rad_length=36.08*)
Calculates the longitudinal charge profile in the EM shower at distance z (m) with parameters for the density (g/cm^3), critical energy (TeV), and electron radiation length (g/cm^2) in ice.

max_length(*density=0.92*, *crit_energy=7.86e-05*, *rad_length=36.08*)
Calculates the maximum length (m) of an EM shower with parameters for the density (g/cm^3), critical energy (TeV), and electron radiation length (g/cm^2) in ice.

class pyrex.signals.**FastAskaryanSignal**(*times*, *energy*, *theta*, *n=1.78*, *t0=0*)
Bases: *pyrex.signals.Signal*

Askaryan pulse binned to times from neutrino with given energy (TeV) observed at angle theta (radians). Optional parameters are the index of refraction n, and pulse offset to start time t0 (s). Returned signal values are electric fields (V/m).

Note that the amplitude of the pulse goes as 1/R, where R is the distance from source to observer. R is assumed to be 1 meter so that dividing by a different value produces the proper result.

vector_potential
Recover the vector_potential from the electric field. Mostly just for testing purposes.

RAC(*time*)
Calculates R * vector potential (A) at the Cherenkov angle in Vs at the given time (s).

charge_profile(*z*, *density=0.92*, *crit_energy=7.86e-05*, *rad_length=36.08*)
Calculates the longitudinal charge profile in the EM shower at distance z (m) with parameters for the density (g/cm^3), critical energy (TeV), and electron radiation length (g/cm^2) in ice.

max_length(*density=0.92*, *crit_energy=7.86e-05*, *rad_length=36.08*)
Calculates the maximum length (m) of an EM shower with parameters for the density (g/cm^3), critical energy (TeV), and electron radiation length (g/cm^2) in ice.

pyrex.signals.**AskaryanSignal**
alias of *FastAskaryanSignal*

class pyrex.signals.**GaussianNoise**(*times*, *sigma*)
Bases: *pyrex.signals.FunctionSignal*

Gaussian noise signal with standard deviation sigma

class pyrex.signals.**ThermalNoise**(*times*, *temperature*, *resistance*, *f_band*, *f_amplitude=1*, *n_freqs=0*)

Bases: *pyrex.signals.Signal*

Thermal Rayleigh noise at a given temperature (K) and resistance (ohms) in the frequency band f_band=[f_min,f_max] (Hz). Optional parameters are f_amplitude (default 1) which can be a number or a function designating the amplitudes at each frequency, and n_freqs which is the number of frequencies to use (in f_band) for the calculation (default is based on the FFT bin size of given times array). Returned signal values are voltages (V).

3.2.2 pyrex.antenna module

Module containing antenna class capable of receiving signals

class pyrex.antenna.**Antenna**(*position*, *temperature*, *freq_range*, *resistance=None*, *noisy=True*)

Bases: object

Base class for an antenna with a given position (m), temperature (K), allowable frequency range (Hz), total resistance (ohm) used for Johnson noise, and whether or not to include noise in the antenna's waveforms. Defines default trigger, frequency response, and signal reception functions that can be overwritten in base classes to customize the antenna.

is_hit
Test for whether the antenna has received a signal.

isHit()
Deprecated. Replaced by is_hit property.

clear()
Reset the antenna to a state of having received no signals.

waveforms
Signal + noise (if noisy) at each triggered antenna hit.

all_waveforms
Signal + noise (if noisy) at all antenna hits, even those that didn't trigger.

trigger(*signal*)
Function to determine whether or not the antenna is triggered by the given Signal object.

response(*frequencies*)
Function to return the frequency response of the antenna at the given frequencies (Hz). This function should return the response as imaginary numbers, where the real part is the amplitude response and the imaginary part is the phase response.

receive(*signal, polarization=[0, 0, 1]*)
Process incoming signal according to the filter function and store it to the signals list. Subclasses may extend this fuction, but should end with super().receive(signal).

class pyrex.antenna.**DipoleAntenna**(*name, position, center_frequency, bandwidth, resistance, effective_height, polarization=[0, 0, 1], trigger_threshold=0, noisy=True*)

Bases: *pyrex.antenna.Antenna*

Antenna with a given name, position (m), center frequency (MHz), bandwidth (MHz), resistance (ohm), effective height (m), polarization direction, and trigger threshold (V).

trigger(*signal*)
Trigger on the signal if the maximum signal value is above the given threshold.

response (*frequencies*)
Butterworth filter response for the antenna's frequency range.

receive (*signal, polarization=[0, 0, 1]*)
Apply polarization effect to signal, then proceed with usual antenna reception.

3.2.3 pyrex.ice_model module

Module containing ice model. AntarcticIce class contains static and class methods for easy swapping of models. IceModel class is set to the preferred ice model.

class pyrex.ice_model.**AntarcticIce**
Bases: object

Class containing characteristics of ice at the south pole.

k = 0.438

a = 0.0132

n0 = 1.32

thickness = 2850

classmethod gradient (*z*)
Returns the gradient of the index of refraction at depth z (m).

classmethod index (*z*)
Returns the medium's index of refraction, n, at depth z (m).

static temperature (*z*)
Returns the temperature (K) of the ice at depth z (m)

classmethod attenuation_length (*z, f*)
Returns the attenuation length at depth z (m) and frequency f (MHz).

class pyrex.ice_model.**NewcombIce**
Bases: *pyrex.ice_model.AntarcticIce*

Class inheriting from AntarcticIce, with new attenuation_length function based on Matt Newcomb's fit (DOESN'T CURRENTLY WORK - USE ANTARCTICICE).

classmethod attenuation_length (*z, f*)
Returns the attenuation length at depth z (m) and frequency f (MHz) by Matt Newcomb's fit (DOESN'T CURRENTLY WORK - USE BOGORODSKY).

pyrex.ice_model.**IceModel**
alias of *AntarcticIce*

3.2.4 pyrex.earth_model module

Module containing earth model. Uses PREM for density as a function of radius and a simple integrator for calculation of the slant depth as a function of nadir angle.

pyrex.earth_model.**prem_density** (*r*)
Returns the earth's density (g/cm^3) for a given radius r (m). Calculated by the Preliminary Earth Model (PREM).

pyrex.earth_model.**slant_depth** (*angle*, *depth*, *step=5000*)
Returns the material thickness (g/cm^2) for a chord cutting through earth at Nadir angle and starting at depth (m).

3.2.5 pyrex.particle module

Module for particles (namely neutrinos) and neutrino interactions in the ice. Interactions include Earth shadowing (absorption) effect.

class pyrex.particle.**NeutrinoInteraction**(*c*, *p*)
Bases: object

Class for neutrino interaction attributes.

cross_section(*E*)
Return the cross section at a given energy E (GeV).

interaction_length(*E*)
Return the interaction length at a given energy E (GeV).

class pyrex.particle.**Particle**
Bases: tuple

Named tuple for containing particle attributes. Consists of a 3-D vertex (m), 3-D direction vector, and an energy (GeV).

direction

energy

vertex

pyrex.particle.**random_direction**()
Generate an arbitrary 3D unit vector.

class pyrex.particle.**ShadowGenerator**(*dx*, *dy*, *dz*, *energy_generator*)
Bases: object

Class to generate UHE neutrino vertices in (relatively) shallow detectors. Takes into accout Earth shadowing (sort of). energy_generator should be a function that returns a particle energy in GeV.

create_particle()
Creates a particle with random vertex in cube and random direction.

3.2.6 pyrex.ray_tracing module

Module containing class for ray tracking through the ice. Ray tracing not yet implemented.

class pyrex.ray_tracing.**PathFinder**(*ice_model*, *from_point*, *to_point*)
Bases: object

Class for ray tracking.

exists
Boolean of whether path exists.

emitted_ray
Direction in which ray is emitted.

path_length
Length of the path (m).

tof
Time of flight (s) for a particle along the path. Calculated using default values of self.time_of_flight()

time_of_flight(*n_steps=10*)
Time of flight (s) for a particle along the path.

attenuation(*f*, *n_steps=10*)
Returns the attenuation factor for a signal of frequency f (Hz) traveling along the path.

propagate(*signal*)
Applies attenuation to the signal along the path.

3.2.7 pyrex.kernel module

Module for the simulation kernel. Includes neutrino generation, ray tracking (no raytracing yet), and hit generation.

class pyrex.kernel.**EventKernel**(*generator*, *ice_model*, *antennas*)
Bases: object

Kernel for generation of events with a given particle generator, ice model, and list of antennas.

event()
Generate particle, propagate signal through ice to antennas, process signal at antennas, and return the original particle.

PYTHON MODULE INDEX

p

INDEX