# AskaryanModule Tutorial

Jordan C. Hanson
Assistant Professor of Physics
Whittier College

June 19, 2018

## 1 Introduction

The AskaryanModule is a C++ package intended to calculate the complex electric field due to the Askaryan effect, given initial conditions. The notation, physical effects, and typical numbers are given by the paper "Complex Analysis of Askaryan Radiation: A fully analytic treatment including the LPM effect and Cascade Form Factor," by J.C. Hanson and A. Connolly. The reference is: Astroparticle Physics **91** (2017) p. 75-89. For the most complete technical understanding of the code, please read the paper. Figure 1 of the paper gives the coordinate system in which we cast the field, which is particularly important for understanding the outputs.

The Askaryan field spectrum depends several obvious parameters: $E_C$, the cascade energy, $\theta_C$ and $\theta$, the Cherenkov and viewing angles, and whether the LPM effect is significant. The particle cascade in the dense medium (ice) is treated theoretically, using the Gaisser-Hillas and Greisen cascade profiles. Once the cascade equation is generated, two important effects alter the corresponding Askaryan field spectrum. First, the spectrum is filtered by the effect of field coherence in the near-zone, Fresnel zone, and the far-field zones. It is important to know in which zone we are observing the field, and it is by no means obvious that we may always make the far-field approximation. Second, the spectrum is filtered by the cascade form factor, which is a description of the *instantaneous charge distribution*, or ICD. This is distinct from the cascade profile, which is the total excess negative charge ever produced by the cascade, versus distance. The ICD describes the *instantaneous* charge, which is like a thin disc propagating along the shower axis. The radius of this disc $1/\sqrt{2\pi}\rho_0$ (with $\rho_0$ in inverse meters) and viewing angle $\theta$ have a strong effect on the Askaryan field.

## 2 Required Code

The code is stored in an open-source fashion on github.com. To download the code into a directory called AskaryanModule under the home directory:

```
git clone https://github.com/918particle/AskaryanModule.git ~/AskaryanModule
git checkout simplified
```

The above two lines of code will clone the git repository to the local machine, and checkout a branch named *simplified.* This branch contains just the basic class and source file, without numerious other directories, pseudo-data and plotting scripts used to write the paper (Astroparticle Physics **91** (2017) p. 75-89). Typing *ls* shows three different directories: *RalstonBuniy*, *JordanEugene*, and *EugeneHong.* **The paper covers the RalstonBuniy version of the Askaryan class, and the other two versions are for comparison.** The EugeneHong version came from E. Hong (CCAAP post-doctoral fellow at Ohio State University), and is used in AraSim by the ARA collaboration (along with subsequent modifications). The EugeneHong module takes the ARVZ model (e.g. Physical Review D **84** 103003, 2011) and uses the Gaisser-Hillas and Greisen cascade profiles to calculate Askaryan fields without having to simulate particle cascades.

# 3 A Simple Example

Change to the RalstonBuniy directory, and notice the class definition and the source file, along with a reference to the original Ralston and Buniy framework paper upon which our calculations were built.

```
$ cd RalstonBuniy
$ ls
Askaryan.cxx Askaryan.h global.h REFERENCE
```

The complete Askaryan.h class should look like this:

```
#ifndef ASKARYAN_H_
#define ASKARYAN_H_
//Askaryan class
//Author: Jordan C. Hanson
//June 19th, 2018
//Adapted from Ralston and Buniy (2001)

//Variables defined for one interaction, (one angle and distance),
//but continuous frequency.

#include <vector>
#include <cmath>
#include <complex>
#include "global.h"

typedef std::complex<float> cf;

int factorial(int);
int dfactorial(int);

class Askaryan {
protected:
float _askaryanTheta; //radians
std::vector<float>* _askaryanFreq; //GHz
float _askaryanR; //meters
float _askaryanDepthA; //meters
float _Nmax; //excess electrons over positrons, per 1000, at shower max
float _E; //energy in GeV
int _isEM; //Electromagnetic parameterizations
int _isHAD; //Hadronic parameterizations
float _rho0; //Form factor parameter, with units 1/m
//Use the _rho0 parameter above, in a single exponential model from the complex analysis paper (2017)
bool _useFormFactor;
//Require that even under the LPM elongation, the low-frequency radiation is the same as without LPM
//Similar to a strict total track length requirement
bool _strictLowFreqLimit;
public:
Askaryan(): _isEM(0), //EM shower, use emShower()
_isHAD(0), //HAD shower, use hadShower()
_rho0(10.0),
_useFormFactor(true),
_askaryanDepthA(STANDARD_ASK_DEPTH),
_askaryanR(STANDARD_ASK_R),
_Nmax(STANDARD_ASK_NMAX),
```

```
_askaryanTheta(THETA_C*PI/180.0),
_strictLowFreqLimit(false) {};
void toggleFormFactor(); //What it sounds like: use or don't use form factor.
void toggleLowFreqLimit(); //What it sounds like: turn on strictLowFreqLimit.
void setAskTheta(float); //radians
void setAskFreq(std::vector<float>*); //GHz
void setAskR(float); //m
void setAskDepthA(float); //m
void setNmax(float); //per 1000
void setAskE(float); //GeV
float criticalF(); //GHz
float getAskE(); //GeV
float getAskR(); //meters
float getAskDepthA(); //m
float getAskNmax(); //pure number
float getAskEta(float); //pure number
void emShower(float); //Shower parameters from energy in GeV
void hadShower(float); //Shower parameters from energy in GeV
void setFormScale(float); //Set shape of shower (meters^{-1}).
std::vector<float>* k(); //1/meters
std::vector<float>* eta(); //unitless
std::vector<cf>* I_ff(); //m
std::vector<std::vector<cf> >* E_omega(); //V/m/MHz
std::vector<std::vector<float> >* E_t(); //V/m
std::vector<float>* time(); //ns
void lpmEffect();
};
#endif
```

# 4 Class Listing