AskaryanModule Tutorial

Jordan C. Hanson
Assistant Professor of Physics
Whittier College

June 20, 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, θ_C and θ, 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 ρ_0 in inverse meters) and viewing angle θ 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 The Class Definition

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 Askaryan.h class is below. It contains data and data containers, a custom constructor, get-methods, and set-methods.

```
#ifndef ASKARYAN_H_
#define ASKARYAN_H_

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

typedef std::complex<float> cf;

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
```

3.1 Data and Data Containers

The following is a list of the units, function, and typical values of the data and data containers in the class.

- `_askaryanTheta`

  This is the viewing angle θ in units of radians, with allowed values between $[0, \pi/2]$. Typical values are close to the Cherenkov angle of $\theta_C = \cos^{-1}(1/n_{ice})$, where $n_{ice} = 1.78$. Constants like θ_C and n_{ice} are defined in global.h

- `_askaryanFreq`

  This is a pointer to a vector of floats corresponding to the frequencies at which the code will yield the Askaryan electric field. **Note: if the electric field versus time is desired, the user must distribute the frequencies uniformly.**

- `_askaryanR`

  This is the distance R from the observer to the cascade maximum, as defined in the papers by J. Ralston and R. Buniy, and J.C. Hanson and A. Connolly. **Note: the user may not assume that the field is evaluated in the far-field approximation. Thus, scaling by R to obtain the field at a different distance is not always valid.**

- `_askaryanDepthA`

  This is the width of the particle cascade in meters, as defined by J. Ralston and R. Buniy. It is not the distance of the cascade maximum to the primary interaction vertex.

- `_Nmax`

  This is the number of excess negatively charged particles at cascade maximum, divided by 1000. This definition comes from J. Ralston and R. Buniy. This parameter is calculated from the Gaisser-Hillas or Greisen parameterizations once emShower or hadShower is executed. The total number of charged particles is converted to excess negative charges by multiplying by the simulated charge excess fraction. See the Appendix of the paper by J.C. Hanson and A. Connolly for details.

- `_E`

  This is the total energy *of the particle cascade.* It plays a role in determining the strength of the LPM effect, among other functions.

- `_isEM, _isHad`

  These are two boolean variables that are set when emShower or hadShower are called. If emShower is called, then $\_isEM = true$, and hadShower sets $\_isHad = true$. If the function lpmEffect is activated, the radiation is modified if $\_isEM = true$.

- `_rho0`

  The ICD excess negative charge is distributed radially away from the shower axis like $\approx \exp(-\sqrt{2\pi}\rho_0\rho)$. While this notation appears strange, it makes the normalization for the cascade form factor trivial and intuitive. Setting the variable $\_rho0$ is equivalent to setting ρ_0 in the equation. The code leaves the choice of value for $(\sqrt{2\pi}\rho_0)^{-1}$ up to the user, however, this value is usually $\approx 1/20$ m. See paper for detail.

- `_useFormFactor`

  This is a boolean parameter that determines whether or not the form factor $\widetilde{F}(\omega, \theta)$ is applied to the observed field. It is sometimes useful to examine the radiation without the form factor, and then apply it to see if the field is still observable.

- `_strictLowFreqLimit`

  Usually this boolean parameter is set to false. When the LPM effect is applied, there are different takes on how to ensure energy is conserved, given the delayed creation of many low energy particles. Toggling this parameter changes the radiation strength between 1-100 MHz.

-

4 Class Member Functions

- `Askaryan() : ...`

  The constructor sets default values for many of the variables. Notice that some of the values (at the moment) are defined in global.h. For example $STANDARD_ASK_R$ is defined in global.h as 1000 meters, but the user might want to set that default value to some other value so that every cascade is observed at some other standard distance.

- `void toggleFormFactor()`

  This function sets $\_useFormFactor$ to the value it currently does not hold.

- `void toggleLowFreqLimit`

  This function is the same as the prior one, except for $\_strictLowFreqLimit$.

- Set functions. The set functions just set the parameter to the given data value or data container. The most commonly used set function is

  ```
  void setAskFreq(std::vector<float>*); \\GHz
  ```

  This function specifies the frequencies at which the code will evaluate the Askaryan field. The units are GHz. The other commonly used set function is

  ```
  void setFormScale(float)
  ```

  This function sets the variable ρ_0.

- Get functions. The get functions just retrieve the relevant data or data containers. For example,

  ```
  float criticalF(); //GHz
  ```

This function retrieves the critical frequency, or the highest frequency specified by the user in setAskFreq. Remember, we specifiy the observable frequencies in this code primarily, so the critical frequency is not half of the maximum frequency provided by the user, nor is it $1/2\Delta t$. The function

```
std::vector<float>* time(); //ns
```

retrieves the time values that correspond to the critical frequency given by the user, starting with $t = 0$. The degrees of freedom are conserved.

- For the remaining get and set functions, simply read the source code.
- Electromagnetic and hadronic shower functions

  ```
  void emShower(float); //Shower parameters from energy in GeV
  void hadShower(float); //Shower parameters from energy in GeV
  ```

  These two functions determine the values of a and N_{max}, which in turn give the overall amplitude of the Askaryan radiation. The overall amplitude is proportional to $N_{max}a$. Naively, we may think of $N_{max}a$ as the area under the cascade profile curve. The *cascade energy*, not the total *neutrino* energy, is to be the argument of these two functions. The units are to be GeV.
- Field retrieval functions

  ```
  std::vector<std::vector<cf> >* E_omega(); //V/m/MHz
  std::vector<std::vector<float> >* E_t(); //V/m
  ```

  These two functions are to be called after emShower or hadShower. They return data containers that contain the field versus either the frequency $\nu = \omega/2\pi$ (user-specified in *_askaryanFreq* in GHz), or the time t in ns. The paper by J.C. Hanson and A. Connolly discusses the use of retarded time, and how to interpret field causality given the electric field versus time. Be careful retrieving the complex field versus time. The field will exhibit pathologies if the frequency bins do not reflect the actual amplitude versus frequency of the field being specified by the input parameters. For example, suppose the field is being viewed at the Cherenkov angle, and the form factor is minimal. If the frequency bins are too wide, or if the critical frequency is too low, the field will exhibit aliasing effects. One of the key features of this code is to keep track of the complex electric field, both the amplitude and phase of each component.
- The LPM scaling

  ```
  void lpmEffect();
  ```

  This function will rescale the cascade width "a" (*_askaryanDepthA*) according to the paper by J.C. Hanson and A. Connolly, if the cascade is electromagnetic ($\_isEM = true$). Note that this also changes the phase of the electric field, not just the amplitude.

5 Examples

What follows are a few examples of how to use the code effectively. Do not read this section until you have read the previous sections. For example, suppose the user wants to calculate the Askaryan field versus frequency, for a given set of parameters. Consider the code below:

```
#include "Askaryan.h"
#include <cstdlib>
#include <fstream>
#include <ctime>
#include <iostream>
```

```
#include <complex>
#include <cmath>
using namespace std;

int main(int argc, char **argv)
{
char title[100];
vector<float> *freqs = new vector<float>;
float df = 0.75;
for(float f1 = 1.0; f1<10.0; f1=f1+df) freqs->push_back(f1*1e-3);
for(float f1 = 1.0; f1<10.0; f1=f1+df) freqs->push_back(f1*1e-2);
for(float f1 = 1.0; f1<10.0; f1=f1+df) freqs->push_back(f1*1e-1);
for(float f1 = 1.0; f1<10.0; f1=f1+df) freqs->push_back(f1*1e+0);
for(float f1 = 1.0; f1<10.0; f1=f1+df) freqs->push_back(f1*1e+1);

Askaryan *h = new Askaryan();
h->setFormScale(1/(sqrt(2.0*3.14159)*0.03));
h->setAskFreq(freqs);
h->emShower(atof(argv[1]));
h->setAskDepthA(1.5);
h->setAskR(1000.0);
float theta = atof(argv[2]);
sprintf(title,"shower_%s.dat",argv[3]);
ofstream out(title);
h->setAskTheta(theta*PI/180.0);
vector<vector<cf> > *Eshow = new vector<vector<cf> >;
Eshow = h->E_omega();
vector<cf> eTheta = Eshow->at(1);
delete Eshow;
for(int j=0;j<eTheta.size();++j) out<<freqs->at(j)<<" "<<abs(eTheta[j])<<endl;
out.close();
delete h;
delete freqs;
return 0;
}
```

The code begins with initializing a vector of frequencies that is not evenly spaced. This facilitates plotting on a logarithmic frequency scale. The Askaryan object is declared, and the user selects a 3 cm cascade lateral width (the 0.03 number in the setFormScale call). The frequencies are then set, followed by a call to the emShower function given the user's first argument. Although the user has now set N_{max} and a, this particular user wants to see what happens if this particular cascade does not follow the Greisen profile exactly, and sets a new value for a. The observation distance R is then set, followed by a user-defined viewing angle. The electric field has three components: E_r, E_θ, and E_ϕ. By symmetry, the third component should always be zero. The largest component is almost always E_θ. Finally, the second component of $Eshow$ (which is E_θ) is retrieved and printed to a file defined by the user.

The next example plots the Askaryan field versus time.

```
#include "Askaryan.h"
#include <cstdlib>
#include <fstream>
#include <ctime>
#include <iostream>
#include <complex>
#include <algorithm>
using namespace std;
```

```
int main(int argc, char **argv)
{
float df_ghz = 0.02; //GHz
float f_max = 4.0; //GHz
float f_min = 0.0; //GHz
float Energy = atof(argv[1]); //GeV

Askaryan *h = new Askaryan();
vector<float> *freqs = new vector<float>;
for(float i=f_min;i<f_max;i+=df_ghz) freqs->push_back(i);
h->setAskFreq(freqs);
h->setAskTheta((THETA_C)*PI/180.0);
h->setFormScale(7.8);
h->emShower(1.0e9);
h->lpmEffect();
vector<vector<cf> > *Eshow = new vector<vector<cf> >;
vector<float> *t = new vector<float>;
vector<cf> e;
Eshow = h->E_t();
e = Eshow->at(1);
t = h->time();
ofstream out(argv[2]);
for(int j=0;j<e.size();++j) out<<t->at(j)<<" "<<real(e[j])<<endl;
out.close();

delete h;
delete Eshow;
delete freqs;
return 0;
}
```

The code first defines a set of evenly-spaced frequency bins, declares an Askaryan object, and sets the frequencies, viewing angle, and form factor scale. Next, the LPM effect is activated. Because the shower is electromagnetic in type, the lpmEffect() function will change the a value of the cascade. Once the $\vec{E}(t)$ is produced by $E\_t()$, the real part versus time is read out.

6 Contact Information

Jordan C. Hanson
Assistant Professor of Physics
Whittier College
jhanson2@whittier.edu
918particle@gmail.com

If you would like to be a developer of this code, or are interested in writing additional papers on the topic with me, please contact me, and clone the code on github.com.