Ioffe Institute of Russian Academy of Science

FAINA

Numerical code for modeling electromagnetic radiationfrom astrophysical sources

User's guide

Saint-Petersburg — 2023

Contents

Introduction

FAINA - is a numerical code for modeling different types of electromagnetic radiation of astrophysical source. It is written in C++ and supports parallel computations using openmp method. FAINA allows to model observable fluxes from sources with different parameters and geometries vai different emission mechanisms, and also to optimize source parameters to fit observational data.

Installation

Current version of the code is available on github https://github.com/VadimRomansky/Faina. FAINA is distributed freely under the MIT public license. Download the archive with code and extract it into preferred root directory.

Windows

With Windows OS it is recomended to use Microsoft Visual Studio and open solution Faina.sin with it. Operability was examined for Windows 10 and Visual Studio 2022 version.

Unix

There are two possible ways to run FAINA on Unix. We recommend to use IDE QtCreator and open with it file Faina.pro located in the rrot directory.

Other way is to use FAINA from terminal. To compile and run it you can use following commands

```
$ make
$ ./Faina
```

Operability was examined for Ubuntu 22.04.

Runnng simple problem

Let see a simple example of solving radiation problem with faina. You can find in the function evaluateSimpleSynchrotron in the file /Src/examples.cpp. Synchrotron radiation from homogenous source with the shape of cylinder with axis along line of sight and with powerlaw electron distribution is evaluated in this example. But it demonstrates a general approach to evaluation of radiation with FAINA code.

Let define values of magnetic field and electron number density in the source (code uses CGS units).

```
double B = 1.0;
double electronConcentration = 1.0;
```

Then you need to create distribution of emitting electrons. There are a different type of particle distribution implemented in the code, let use isotropic powerlaw distribution for this example. You should call the constructor of MassiveParticlePowerLawDistribution with following parameters - mass of emitting particles (electrons in this case), powerlaw index of distribution, which is defined as positive number p in $F(E)$ $1/E^p$, starting energy of powerlaw distribution, and electrons number density.

```
MassiveParticleDistribution* distribution =
new MassiveParticlePowerLawDistribution(
massElectron, 3.0, me_c2, electronConcentration);
```

After that you should create a radiation source, for example it would be homogenous flat disk with axis along line of sight. You should call the constructor of SimpleFlatSource with following parameters: electrons distribution, magnetic field, sinus of it's inclination angle to the line of sight, radous of cylinder, it's hight and distance to the observer.

```
RadiationSource* source = new SimpleFlatSource(
distribution, B, 1.0, parsec, parsec, 1000 * parsec);
```

And the last thing you need is an radiation evaluator. They are different for every specific type of radiation. Here we create a SybchrotronEvaluator with following parameters: number of grid points for integration electron distribution function over energy, lower and upper limits of electron energy that will be taken into account and boolean parameter determining include synchrotron self absorption or not.

```
RadiationEvaluator* evaluator = new
SynchrotronEvaluator(1000, me_c2, 1000 * me_c2, true);
```

Synchrotron approximation is valid only for frequencies of radiation much greater than cyclotron frequency, so let evaluate it

```
double cyclOmega =
electron_charge * B / (massElectron * speed_of_light);
```

Now you can evaluate spectrum of synchrotron radiation. Radiation evaluator has a method writeFluxFromSourceToFile which allows to calculate flux energy density and write it into the file in units energy vs power per energy per area, or erg vs sm^-2s^-1. This method takes following input parametes: output file name which will be created or rewritten, lower and upper limits of energy range and number of grid points, which will be distributed logarithmically in the range.

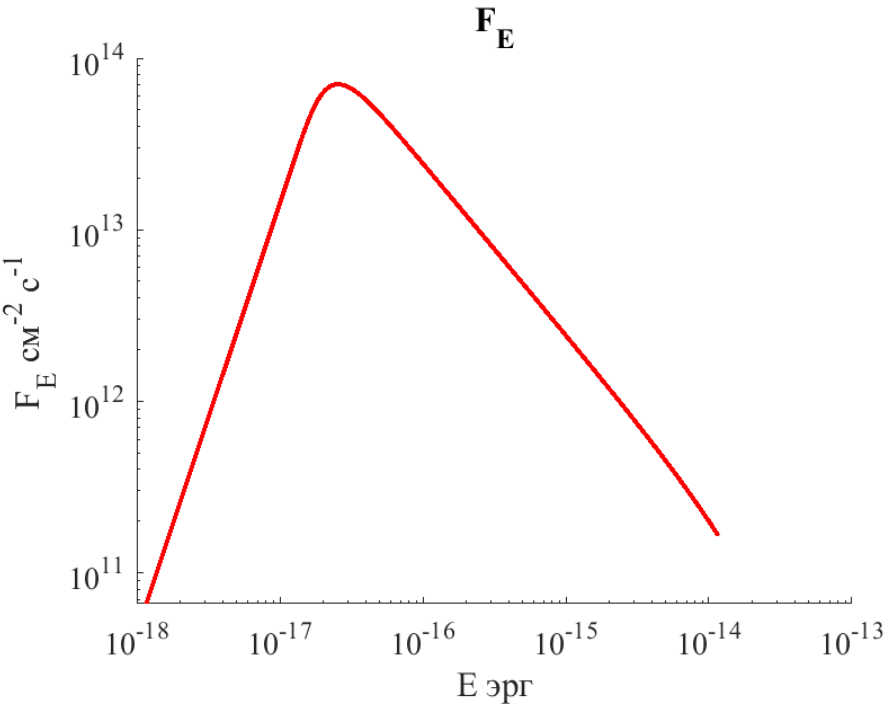

Figure 1: Synchrotron radiation flux energy density from test source

If you need other units you should use method evaluateFluxFromSource which provides a flux energy density at given energy and rewrite output.

```
evaluator->writeFluxFromSourceToFile("out.dat",source,
10*hplank*cyclOmega, 1E5*hplank*cyclOmega, 1000);
```

Evaluated spectrum of flux energy density from this source is shown in 1. Examples of plotting scripts you can find in Figure directory pyFAINA.

Chapter 1

Evaluation the radiation of the sources

FAINA allows to evaluate electromagnetic radiation from sources with various type of particle distributions and different parameters such as magnetic fiels, number density and other. In this chapter we describe creating various types of radiation sources.

1.1 Particle distributions

Crucial parameter for evaluation of any type of radiation is a distribution function of emitting particles. In the FAINA code abstract class ParticleDistribution and derived classes are used for representation of distributions. Public methods of class ParticleDistribution are listed in Table 1.1:

Table 1.1: Public methods of ParticleDistribution class

ParticleDistribution	abstract class for particle distributions
double distribution(const double& energy, const double& mu, const double& phi)	returns probability density function in polar coordinates with given energy, cosinus of polar angle and azimutal angle, normalized to the particles number density
virtual double distributionNormalized(const double& energy, const double& mu, const double& phi)	virtual method, returns probability density function in polar coordinates with given energy, cosinus of polar angle and azimutal angle, normalized to unity
virtual double getMeanEnergy()	virtual method, returns mean energy of particles in distribution
double getConcentration()	returns particles number density
void resetConcentration(const double& concentration)	changes number density to the given value

For creating a distribution object you need some inherited class. Inheritance tree of ParticleDistribution splits into two big branches - PhotonDistribution for distribution of photons, and MassiveParticleDistribution - for massive particles. Scheme of class hierarchy is shown in Figure 1.1.

It is important to note, that photons distributions are not used to represent results of evaluation of electromagnetic radiation. They are necessary only as input parameter for evaluation of inverse Compton scattering. Class PhotonDistribution is only an interface and

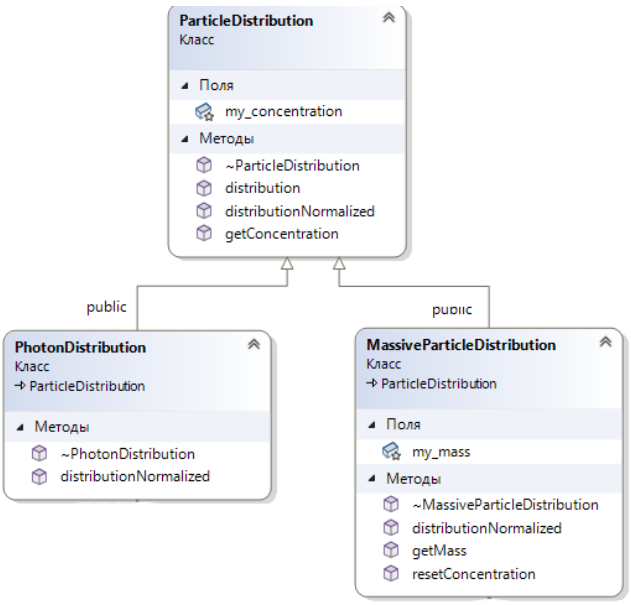

Figure 1.1: Two branches of inheritance tree of ParticleDIstribution

has not its own specific methods. Class MassiveParticleDistribution is also abstract, but his methods are listed in Table 1.2

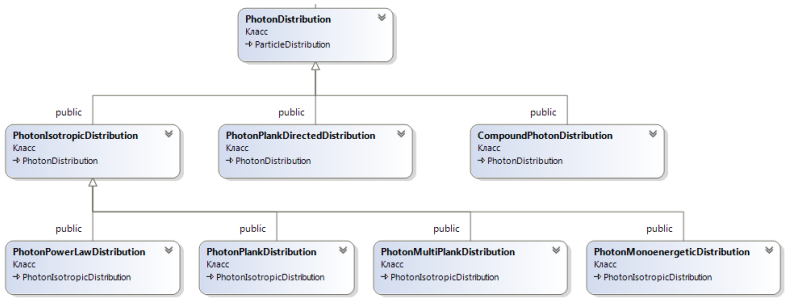

Figure 1.2: Class hierarchy of photon distributions

Table 1.2: Public methos of MassiveParticleDistribution class

MassiveParticleDistribution	abstract class for massive particles distribution
virtual double minEnergy()	virtual method, returns the lowest possible energy of particle in this distribution
virtual double maxEnergy()	virtual method, returns the upper limit of energy of particle in this distribution. NOTE that if upper limit of energy is infinite, this method returns negative number
double getMass()	returns mass of single particle

1.1.1 Photon distributions

Abstract class PhotonDistribution has following derived class: abstract PhotonIsotropicDistribution, which represented isotopic distributions and some non-abstract classes: PhotonPlankDirectedDistribution, which represent photons with Plank distribution with respect to energy, but collimated in some solid angle, and CompoundPhotonDistribution, which is usefull for sum of several arbitrary photon distributions.

Class PhotonIsotropicDistribution again has its own inherited classes. It is a PhotonPowerLawDistribution for powerlaw distribution, PhotonPlankDistribution for Plank distributions, PhotonMultiPlankDistribution for sum of several Plank distributions and PhotonMonoenergeticDistribution for isotropic photons with same energy. Class hoerarchy of photon distributions is presented in Figure 1.2.

Methods of PhotonDistribution and it's inherited classes are listed in Table 1.3. NOTE, that metods distributionNormalized(const double& energy) and distribution(const double& energy) are not distribution with respect to energy, but just full distribution with dropped angular arguments. So to obtain distribution with respect to energy one should multiply result of this functions by 4π.

Table 1.3: Public methods of PhotonDistribution class and derived classes

PhotonDistribution	abstract interface for photon distributions

PhotonIsotropicDistribution	abstract class for isotropic distributions of photons
double distribution(const double& energy)	returns probability density function in polar coordinates with dropped angular arguments (normalized to the number density divided by 4π)
virtual double distributionNormalized(const double& energy)	virtual method, returns probability density function in polar coordinates with dropped angular arguments (mormalized to the $1/4\pi$)
void writeDistribution(const char* fileName, int Ne, const double& Emin, const double& Emax)	writes distribution into given file as to columns - energy and distribution from Emin to Emax with Ne logarithmically distributed points
PhotonPowerLawDistribution	class representing powerlaw distribution of photons
PhotonPowerLawDistribution(const double& index, const double& E0, const double& concentration)	constructor, creates distribution with given power-law index p such as $F(E)$ $1/E^p$, starting energy and number density
double getIndex()	returns power-law index
double getE0()	returns starting energy of distribution
PhotonPlankDistribution	class representing Plank distribution
PhotonPlankDistribution(const double& temperature, const double& amplitude)	constructor, creates distribution with given temperature and amplitude - relation of number density to the number density of photons in equilibrium black-body radiation
static PhotonPlankDistribution* getCMBRadiation()	static method, returns object representing Cosmic Microwave Background Radiation (temperature 2.725 K, amplitude 1)
double getTemperature()	returns temperature of distribution
PhotonMultiPlankDistribution	class representing sum of several Plank distributions
PhotonMultiPlankDistribution(int N, const double* const temperatures, const double* const amplitudes)	constructor, creates distribution constisting of N plank distributions with given temperatures and amplitudes
static PhotonMultiPlankDistribution* getGalacticField()	static method, returns object representing mean Galactic photon field described in [1]. This distribution consists of five plank components with temperatures $2.725K, 20K, 3000K, 4000K, 7000K$ and amplitudes $1.0, 4 \cdot 10^{-4}, 4 \cdot 10^{-13}, 1.65 \cdot 10^{-13}, 1.0 \cdot 10^{-14}$ respectively

PhotonMonoenergeticDistribution	class representing population of isotropic photons with close energy
PhotonMonoenergeticDistribution(const double& Energy, const double& halfWidth, const double& concentration)	constructor, creates object with given mean energy, half-width of uniform distribution around mean energy and number density
CompoundPhotonDistribution	class representing sum of several arbitrary distributions
CompoundPhotonDistribution(int N, PhotonDistribution** distributions)	constructor, creates distribution consisting of N arbitrary distributions
CompoundPhotonDistribution(PhotonDistribution* dist1, PhotonDistribution* dist2)	constructor, creates distribution which is sum of two given distributions
CompoundPhotonDistribution(PhotonDistribution* dist1, PhotonDistribution* dist2, PhotonDistribution* dist3)	constructor, creats distribution which is sum of three given distributions
PhotonPlankDirectedDistribution	class representing distribution which is Plank-like with respect to energy, but collimated into given direction
PhotonPlankDirectedDistribution(const double& temperature, const double& amplitude, const double& theta0, const double& phi0, const double& deltaTheta)	constructor, creates distribution with given temperature, amplitude, angles determining mean direction of photons and half-width angle of cone in which photons propagate
double getTemperature()	return temperature of distribution

User can define other photons distribution, creating class inherited from PhotonDistribution or PhotonIsotropicDistribution and overriding virtual method distributionNormalized.

1.1.2 Distributions of massive particles

Distributions of massive particles are represented by class MassiveParticleDistribution and inherited classes. Similarly to the photon distributions, isotropic distrinutions are important type, represented by class MassiveParticleIsotropicDistribution. This class also has methods distributionNormalized(const double& energy) and distribution(const double& energy), which are not distribution with respect to energy, but just full distribution with dropped angular arguments. So to obtain distribution with respect to energy one should multiply result of this functions by 4π.

Abstract class of isotropic distributions has seven inherited classes for specific distributions: MassiveParticlePowerLawDistribution - for power-law distributions, MassiveParticleBrokenPowerLawDistribution - for double power-law distibutions with knee, MassiveParticlePowerLawCutoffDistribution - for power-law distributions with exponential cutoff, MassiveParticleMaxwellDistribution - for non-relativistic maxwellian distribution (but it use full energy, including rest energy), MassiveParticleMaxwellJuttnerDistribution - for

Maxwell -Juttner distribution, MassiveParticleTabulatedIsotropicDistribution - for arbitrary distributions, described with array of values and MassiveParticleMonoenergeticDistribution - for beam of particles with close energies. Also there are six anisotropic distributions, implemented in the code. MassiveParticleTabulatedPolarDistribution - for tabulated distribution with dependence on energy and polar angle, MassiveParticleAnisotropicDistribution - for arbitrary tabulated anisotropic distributions, MassiveParticleMonoenergeticDirectedDistribution - for distributions represented narrow beam of particles with close energies, MassiveParticleMovingDistribution - for transformation the distributions from one frame to another, CompoundMassiveParticleDistribution - for sum of arbitraty distributions and CompoundWeightedMassiveParticleDistribution - for weighted sum of arbitrary distributions. In some cases operating with relative weghts of distributions is more useful than with absolute concentrations. Class hierarchy of distributions of massive particles is shown in Figure 1.3.

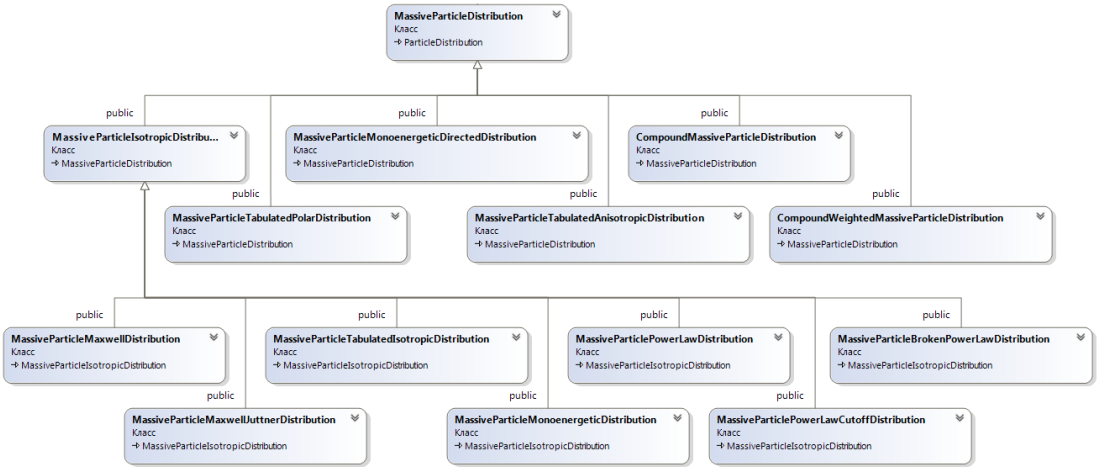

Figure 1.3: Class hierarchy of massive particles distributions

public methods of classes for massive particle distributions are listed in Table 1.4. User can define his own specific distributions, creating class inherited from MassiveParticleDistribution or MassiveParticleIsotropicDistribution.

Table 1.4: Public methods of classes derived from MassiveParticleDistribution

MassiveParticleIsotropicDistribution	Abstract class for isotropic distributions of massive particles
double distribution(const double& energy)	returns probability density function in polar coordinates with dropped angular arguments (normalized to the number density divided by 4π)

virtual double distributionNormalized(const double& energy)	virtual method, returns probability density function in polar coordinates with dropped angular arguments (mormalized to the $1/4\pi$)
void writeDistribution(const char* fileName, int Ne, const double& Emin, const double& Emax)	writes distribution into given file as to columns - energy and distribution from Emin to Emax with Ne logarithmically distributed points
double distributionNormalizedWithLosses(const double& energy, const double& lossRate, const double& time)	returns a distribution which evaluted till time t via synchrotron losses with equation $f_t(E) = f\left(\frac{E}{1-Elt}\right) \cdot \frac{1}{(1-Elt)^2}$ and loss rate $l = 4e^4B^2/9m^4c^7$
MassiveParticlePowerLawDistribution	class representibg power-law distribution
MassiveParticlePowerLawDistribution(const double& mass, const double& index, const double& E0, const double& concentration)	cobstructor, creates distribution with given particle mass, power-law index, starting energy and concentration
double getIndex()	returns power-law index
double getE0()	returns starting energy of distribution
MassiveParticleBrokenPowerLawDistribution	class representing double power-law distribution with knee
MassiveParticleBrokenPowerLawDistribution(const double& mass, const double& index1, const double& index2, const double& E0, const double& Etran, const double& concentration)	constructor, creates distribution with given particle mass, power-law indexes at low and high energies, starting energy, energy of transition from one index to another and concentration
double getIndex1()	returns power-law index at low energies
double getIndex2()	returns power-law index at high energies
double getE0()	returns starting energy of distribution
double getEtran()	returns energy of transition fron one index to another
MassiveParticlePowerLawCutoffDistribution	class representing power-law distribution with exponential cutoff
MassiveParticlePowerLawCutoffDistribution(const double& mass, const double& index, const double& E0, const double& beta, const double& Ecut, const double& concentration)	constructor, creates distribution with given particle mass, power-law index, starting energy, power of cutoff and cutoff energy and concentration $F(E) \propto (E/E_0)^{-index} \cdot \exp(-(E/E_{cut})^{\beta})$

double getIndex()	returns power-law index
double getBeta()	returns cutoff power parameter
double getE0()	returns starting energy of distribution
double getEcutoff()	returns cutoff energy
MassiveParticleMaxwellDistribution	class representing non-relativistic maxwellian distribution
MassiveParticleMaxwellDistribution(const double& mass, const double& temperature, const double& concentration)	creates distribution with given particles mass, temperature and concentration
double getTemperature()	returns temperature
MassiveParticleMaxwellJuttnerDistribution	class representing Maxwell-Juttner distribution
MassiveParticleMaxwellJuttnerDistribution(const double& mass, const double& temperature, const double& concentration)	creates distribution with given particle mass, temperature and concentration
double getTemperature()	returns temperature
MassiveParticleTabulatedIsotropicDistribution	class for tabulated isotropic distribution
MassiveParticleTabulatedIsotropicDistribution(const double& mass, const char* fileName, const int N, const double& concentration, DistributionInputType inputType)	constructor, creates distribution with given mass and concentraion, reading table with N lines from given file. inpuType - enum variable determining in which coordinates distribution is defined in file
MassiveParticleTabulatedIsotropicDistribution(const double& mass, const char* energyFileName, const char* distributionFileName, const int N, const double& concentration, DistributionInputType inputType)	constructor, creates distribution with given mass and concentraion, reading Nx2 table with from two given files. inpuType - enum variable determining in which coordinates distribution is defined in files
MassiveParticleTabulatedIsotropicDistribution(const double& mass, const double* energy, const double* distribution, const int N, const double& concentration, DistributionInputType inputType)	constructor, creates distribution with given mass and concentraion, reading two data columns from given arrays. inpuType - enum variable determining in which coordinates distribution is defined in arrays
int getN()	returns number of grid points in distribution array
double rescaleDistribution(const double& k)	rescales distribution through the energy axis using fourmula $E' = mc^2 + k \cdot (E - mc^2)$, $F(E') = F(E)/k$. It may be useful when e.g. distribution of electrons is obtained by numerical code with increased electron mass

void addPowerLaw(const double& Epower, const double& index)	replaces the tail of distribution with power-law distribution with given spectral index starting from Epower. Also renorms distribution
MassiveParticleMonoenergeticDistribution	class representing population of isotropic particles with close energy
MassiveParticleMonoenergeticDistribution(const double& mass, const double& Energy, const double& halfWidth, const double& concentration)	constructor, creates distribution with given particle mass, mean energy, half-width of uniform distribution around mean energy and number density
MassiveParticleTabulatedPolarDistribution	class for tabulated distribution with dependence on energy and polar angle
MassiveParticleTabulatedPolarDistribution(const double& mass, const char* energyFileName, const char* muFileName, const char* distributionFileName, const int Ne, const int Nmu, const double& concentration, DistributionInputType inputType)	constuctor, creates distribution with given particle mass and concentration, reading it from files with energy grid points, angular grid points and distribution. inpuType - enum variable determining in which coordinates distribution is defined in files
MassiveParticleTabulatedPolarDistribution(const double& mass, const double* energy, const double* mu, const double** distribution, const int Ne, const int Nmu, const double& concentration, DistributionInputType inputType)	constuctor, creates distribution with given particle mass and concentration, using arrays with energy grid points, angular grid points and distribution. inpuType - enum variable determining in which coordinates distribution is defined in arrays
int getNe()	returns number of energy grid points in distribution array
int getNmu()	returns number of polar angle grid points in distribution array
void double rescaleDistribution(const double& k)	rescales distribution through the energy axis using fourmula $E' = mc^2 + k \cdot (E - mc^2)$, $F(E', \mu) = F(E, \mu)/k$. It may be useful when e.g. distribution of electrons is obtained by numerical code with increased electron mass

MassiveParticleTabulatedAnisotropicDistribution	class for arbitrary tabulated distribution
MassiveParticleTabulatedAnisotropicDistribution(const double& mass, const char* energyFileName, const char* muFileName, const char* distributionFileName, const int Ne, const int Nmu, const int Nphi, const double& concentration, DistributionInputType inputType)	constuctor, creates distribution with given particle mass and concentration, reading it from files with energy grid points, angular grid points and distribution. Grid with respect to azimuthal angle considered uniform and depends only on number of drid points Nphi. inpuType - enum variable determining in which coordinates distribution is defined in files
MassiveParticleTabulatedAnisotropicDistribution(const double& mass, const double* energy, const double* mu, const double*** distribution, const int Ne, const int Nmu, const int Nphi, const double& concentration, DistributionInputType inputType)	constuctor, creates distribution with given particle mass and concentration, using arrays with energy grid points, angular grid points and distribution. Grid with respect to azimuthal angle considered uniform and depends only on number of drid points Nphi. inpuType - enum variable determining in which coordinates distribution is defined in arrays
int getNe()	returns number of energy grid points in distribution array
int getNmu()	returns number of polar angle grid points in distribution array
int getNphi()	returns number of azimuthal angle grid points in distribution array
void rescaleDistribution(const double& k)	rescales distribution through the energy axis using fourmula $E' = mc^2 + k \cdot (E - mc^2)$, $F(E', \mu, \phi) = F(E, \mu, \phi)/k$. It may be useful when e.g. distribution of electrons is obtained by numerical code with increased electron mass
MassiveParticleMonoenergeticDirectedDistribution	class representing narrow beam of particles with close energies
MassiveParticleMonoenergeticDirectedDistribution(const double& mass, const double& Energy, const double& halfWidth, const double& concentration, const double& theta0, const double& phi0, const double& deltaTheta)	constructor, creates distribution with given particle mass, mean energy, half-width of uniform distribution around the mean energy, polar and azimuthal angles determining direction of mean velocity and half-width angle of velocity cone

MassiveParticleMovingDistribution	class transforming distribution from one frame to another
MassiveParticleMovingDistribution(MassiveParticleDistribution* distribution, const double& velocity)	constructor, transforms the given distribution from the frame with givem velocity along z-axis to the lab frame
CompoundMassiveParticleDistribution	class representing distribution as sum of other distributions
CompoundMassiveParticleDistribution(int N, MassiveParticleDistribution** distributions)	constructor, creates distribution which is sum of given distributions
CompoundMassiveParticleDistribution(MassiveParticleDistribution* dist1, MassiveParticleDistribution* dist2)	constructor, creates distribution which is sum of two given distributions
CompoundMassiveParticleDistribution(MassiveParticleDistribution* dist1, MassiveParticleDistribution* dist2, MassiveParticleDistribution* dist3)	constructor, creates distribution which is sum of three given distributions
CompoundWeightedMassiveParticleDistribution	class representing distribution as weighted sum of other distributions
CompoundWeightedMassiveParticleDistribution(int N, const double* weights, MassiveParticleDistribution** distributions)	constructor, creates distribution which is sum of given distributions with given weights
CompoundWeightedMassiveParticleDistribution(MassiveParticleDistribution* dist1, const double& w1, MassiveParticleDistribution* dist2, const double& w2)	constructor, creates distribution which is sum of two given distributions with given weights
CompoundWeightedMassiveParticleDistribution(MassiveParticleDistribution* dist1, const double& w1, MassiveParticleDistribution* dist2, const double& w2, MassiveParticleDistribution* dist3, const double& w3)	constructor, creates distribution which is sum of three given distributions with given weights

1.1.3 Reading distributions from file

Classes for tabulated distribution, such as MassiveParticleTabulatedIsotropicDistribution, have constructors allowing to read distributions from files. It should be text files with tables of data, and format of data can be different. For determining data format there is enumerable type DistributionInputType with five possible values:

- ENERGY_FE - input file contains full energy in CGS units and distribution function $F(E)$

- ENERGY_KIN_FE - input file contains kinetic energy in CGS units and distribution function $F(E_{kin})$
- GAMMA_FGAMMA - input file contains lorentz-factor and distribution function with respect to it $F(\gamma)$
- GAMMA_KIN_FGAMMA - input file contains reduced lorentz-factor $(\gamma - 1)$ and distribution function with respect to it $F(\gamma - 1)$
- MOMENTUM_FP - input file contains momentum in CGS units and distribution function with respect to it $F(p)$

Regardless of input file format, distribution function would be transformed to the units energy vs distribution $F(E)$. Example of reading distribution from file is given below

```
double electronConcentration = 1.0;
int N = 100;
MassiveParticleIsotropicDistribution* distribution = new
MassiveParticleTabulatedIsotropicDistribution(massElectron,
"energy.dat", "distribution.dat", N, electronConcentration,
DistributionInputType::ENERGY_FE);
```

Class MassiveParticleDistributionFactory is implemented for simplicity of reading distributions from files in complicated cases. It has several similar static methods allowing to read array of distribution from set of numerated files. It can be useful in cases when distribution function depends on some external parameter which varies inside the radiation source. Example of reading array of ten distributions of electrons from files, named "Fe0.dat" , "Fe1.dat" etc., consisting of two columns - loerntz-factor and distribution function, and adding power-law tail with index 3, starting from energy $10m_ec^2$, calling one function is given below

```
double electronConcentration = 1.0;
int Nenergy = 100;
int Ndistribution = 100;
double powerLawEnergy = 100*me_c2;
double index = 3.0;
MassiveParticleIsotropicDistribution** distributions =
MassiveParticleDistributionFactory::
readTabulatedIsotropicDistributionsAddPowerLawTail(
massElectron, "./input/Fe", ".dat", Ndistribution,
DistributionInputType::GAMMA_FGAMMA, electronConcentration, Nenergy,
powerLawEnergy, index);
```

Also it is possible to create tabulated distributions not by reading them from files, but from arrays, which can be generated by user with any suitable method.

1.2 Radiation sources

Evaluation of radiation directly with distribution function, using some additional parameters like source volume and distance to it is implemented in code FAINA. But more useful, flexible and recommended way to do it is using the source model. It allows to take into account geomtry of the source, it's inhomogenuity and other features.

There are two types of radiation sources - sources without time dependency, represented with class RadiationSource, and cources depending on time, represented with class RadiationTimeDependentSource. This two classes are not related with each other through inheritance, but the object of class RadiationTimeDependentSource contains the object of class RadiationSource inside. Diagram of class hierarchy of radiation sources is shown in Figure 1.4.

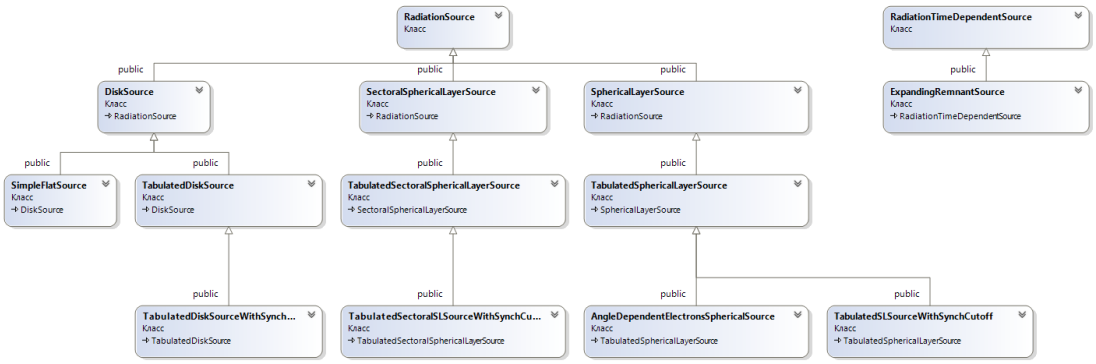

Figure 1.4: class hierarchy of radiation sources

1.2.1 Radiation sources without time dependency

Radiation sources without time dependency are represented with abstract class RadiationSourse and it's derived classes. All sources models uses cylinrical grid with z axis along line of sight. This allows easily integrate through z-axis taking into accound absorption processes. Difference of the real shape of the source from discrete shape of a grid is compensated with filling factor of every grid cell. It represent fraction of cell volume which is located inside the source. Model of the source with geometry of spherical layer is shown in Figure 1.5. Colour shows filling factors of each cell.

Radiation sources have following parameters, that can vary in different grid cells: concentration of emmitting particles, their distribution functions, magnetic field and it's orientation angles. Also distance to the observer is important parameter of the source.

Class RadiationSource has three abstract derived classes: DiskSource - for the sources with shape of a disk with axis along line of sight, SphericalLayerSource - for sources with shape of a spherical layer and SectoralSphericalLayerSource - for sources with shape of spherical layer restricted with some azimutal angle range, and also with minimum cylindrical radius limit. The

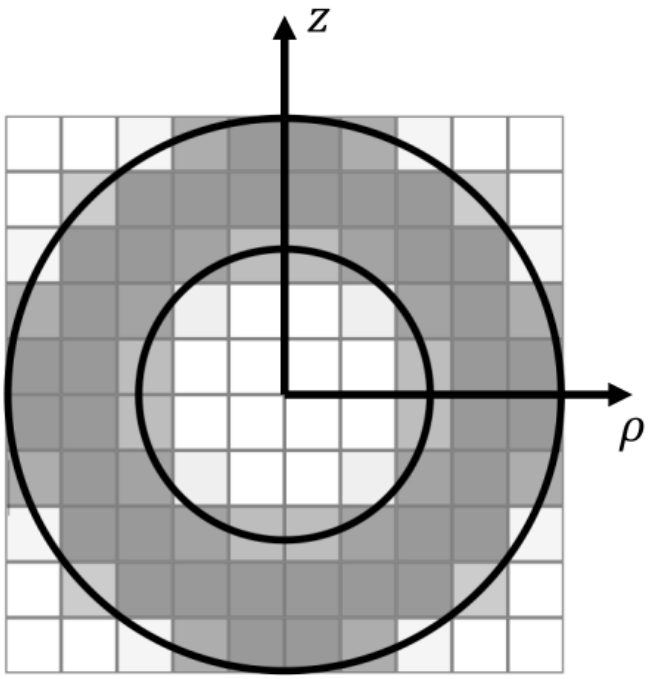

Figure 1.5: Model of the source with spherical layer geometry in the cylindrical grid. Colour shows fraction of cell volume which is located inside the source.

last one is useful when real source is a prolonged object observed with high resolution, and some features of radiation from different regions of the source are studied.

Sources with disk shape have three specific implementations: SimpleFlatSource - homogenous disk consisting of only one grid cell with given parameters, TabulatedDiskSource - inhomogenous disk in wich all parameters are tabulated on the spatial grid, and TabulatedDiskSourceWithSynchCutoff, which is inherited from previous one and allows to take into account synchrotron losses of emitting particles during their propagation inside the sourse. In this source model particles distribution is suppose to be generated on the upper face of disk, representing shock wave, and then particles are moving via convection inside the source. Evolution of the distribution function is described with equation:

$$f_l(E) = f\left(\frac{E}{1 - 4e^4B^2E\ l/9m^4c^7v}\right) \cdot \frac{1}{(1 - 4e^4B^2E\ l/9m^4c^7v)^2} \tag{1.1}$$

where $f(E)$ generated distribution function, E - energy of the particle, B - magnetic field, l - distance from given point to the upper face, v - velocity of convection movement, e - absolute value of particle charge, m - particle mass, c - speed of light.

Sources with shape of spherical layers have following implementations: TabulatedSphericalSource, in wich all parameters are tabulated on the spatial grid and two classes derived from it TabulatedSLSourceWithSynchCutoff and AngleDependentElectronsSphericalSource. First of them allows to take into account

synchrotron losse, like it was done in TabulatedDiskSourceWithSynchCutoff, but in this case distribution is generated on the outer spherical surface, and the second is useful for important case in astrophysics when distribution function of emitting particles is dependent on inclination angle of magnetic field to the direction of shock propagation [2, 3, 4, 5]. In AngleDependentElectronsSphericalSource such parameters as number density, magnetic field and it's orintation angles are tabulated on the spatial grid, while distribution function is tabulated with respect of inclination angle of magnetic field to the shock, which is considered spherically symmetrical and propagates along radius.

Sources with shape of sector of spherical layer has following implementations: TabulatedSectoralSphericalLayerSource, in wich all parameters are tabulated on the spatial grid and derived from it TabulatedSectoralSLSourceWithSynchCutoff, taking into account synchrotron energy losses like it was done in TabulatedSLSourceWithSynchCutoff.

Public methods of classes for radiation sources are listed in the Table 1.5.

Table 1.5: Public methods of classes for time independent radiation sources

RadiationSource	abstract class for radiation sources
virtual double getMaxRho()	virtual method, returns upper boundary of the source at the cylindrical radius
virtual double getMinZ()	virtual method, returns lower boundary of the source at the z axis
virtual double getMaxZ()	virtual method, returns upper boundary of the source at the z axis
virtual double getMaxB()	virtual method, returns maximal magnetic field in the source
virtual double getAverageSigma()	virtual method, returns average magnetization $\sigma = \frac{B^2}{4\pi n m_p c^2}$
virtual double getAverageConcentration()	virtual method, returns average number density
virtual double getRho(int irho)	virtual method, returns cylindrical radius of the cell with number irho through the radial axis
virtual double getZ(int iz)	virtual method, returns z coordinate of the cell with number iz through the z axis
virtual double getPhi(int iphi)	virtual method, returns azimutal angle of the cell with iphi number through the azimutal coordinate
virtual int getRhoIndex(const double& rho)	virtual method, returns radial number of cell containing given radial coordinate
virtual bool isSource(int irho, int iphi)	virtual method, return boolean value defining are cells with given radial and azimutal numbers part of the source or not. It is useful for modeling of sources with complex geometry

int getNrho()	returns number of grid points through the radial axis
int getNz()	returns number of grid points through the z axis
int getNphi()	returns number of grid points twith respect to the azimutal angle
double getDistance()	returns distance to the source
virtual getArea(int irho)	virtual method, returns average area of cross-section of part of the cell, filled with the source matter
virtual getLength(int irho, int iz, int iphi)	virtual method, returns average thickness of the part of the source, filled with the source matter
getVolume(int irho, int iz, int iphi)	returns volume of the part of the cell, filled with source matter. This method is consistent with getArea and getLength, and returns their product
virtual getB(int irho, int iz, int iphi)	virtual method, returns magnetic filled in given cell
virtual getConcentration(int irho, int iz, int iphi)	virtual method, returns number density in given cell
virtual getSinTheta(int irho, int iz, int iphi)	virtual method, returns sinus of angle between magnetic field and line of sight in given cell
virtual void getVelocity(int irho, int iz, int iphi, double& velocity, double& theta, double& phi)	virtual method, returns velocity in given cell, and polar and azimutal angles for it's direction
virtual getTotalVolume()	virtual method, returns total volume of the source
virtual resetParameters(const double* parameters, const double* normalizationUnits)	virtual method, reseting parameters of the source. Lists of parameters are different for different types of sources. Method takes for input array of parameters in normalized units, and array of normalization conctants. This method for example is used for fitting modelled radiation to the observational data and optimization such parameters as magnetic field, number density and others.

virtual getParticleDistribution(int irho, int iz, int iphi)	virtual method, returns emitting particles distribution in given cell
DiskSource	Абстрактный класс для источников в форме диска
SimpleFlatSource	Класс для источников в форме однородного диска
SimpleFlatSource(MassiveParticleDistribution* electronDistribution, const double& B, const double& sinTheta, const double& rho, const double& z, const double& distance, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными распределением частиц, магнитным полем, синусом угла его наклона, радиусом диска, толщиной диска, расстоянием до источника и скоростью движения вещества
TabulatedDiskSource	Класс для источников в форме диска с таблично заданными значениями параметров
TabulatedDiskSource(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, double*** B, double*** sinTheta, double*** concentration, const double& rho, const double& z, const double& distance, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными с помощью массивов распределением частиц, магнитным полем, синусом угла его наклона, а так же заданными радиусом диска, толщиной диска, расстоянием до источника и скоростью движения вещества
TabulatedDiskSource(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, const double& B, const double& sinTheta, const double& concentration , const double& rho, const double& z, const double& distance, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными однородными распределением частиц, магнитным полем, синусом угла его наклона, а так же заданными радиусом диска, толщиной диска, расстоянием до источника и скоростью движения вещества
TabulatedDiskSourceWithSynchCutoff	Класс для источников в форме диска с таблично заданными значениями параметров и учетом синхротронных потерь энергии частиц
TabulatedDiskSourceWithSynchCutoff(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, double*** B, double*** theta, double*** concentration, const double& rho, const double& z, const double& distance, const double& downstreamVelocity, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными с помощью массивов распределением частиц, магнитным полем, синусом угла его наклона, а так же заданными радиусом диска, толщиной диска, расстоянием до источника, скоростью конвекции частиц и скоростью движения вещества

TabulatedDiskSourceWithSynchCutoff(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, const double& B, const double& concentration, const double& theta, const double& rho, const double& z, const double& distance, const double& downstreamVelocity, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными однородными распределением частиц, магнитным полем, синусом угла его наклона, а так же заданными радиусом диска, толщиной диска, расстоянием до источника, скоростью конвекции частиц и скоростью движения вещества
SphericalLayerSource	Абстрактный класс для источников в форме шарового слоя
double getInnerRho()	возвращает внутренний радиус шарового слоя
TabulatedSphericalLayerSource	Класс для источников в форме шарового слоя с таблично заданными значениями параметров
TabulatedSphericalLayerSource(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, double*** B, double*** sinTheta, double*** concentration, const double& rho, const double& rhoin, const double& distance, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными с помощью массивов распределением частиц, магнитным полем, синусом угла его наклона к лучу зрения, а так же заданными внешним и внутренним радиусом шарового слоя, расстоянием до источника и скоростью движения вещества
TabulatedSphericalLayerSource(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, const double& B, const double& concentration, const double& sinTheta, const double& rho, const double& rhoin, const double& distance, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными однородными распределением частиц, магнитным полем, синусом угла его наклона, а так же заданными внутренним и внешним радиусом шарового слоя, расстоянием до источника и скоростью движения вещества
AngleDependentElectronsSphericalSource	Класс для источников в форме шарового слоя с таблично заданными значениями концентрации и магнитного поля и функцией распределения излучающих частиц, зависящей от угла наклона магнитного поля к направлению распространения ударной волны
AngleDependentElectronsSphericalSource(int Nrho, int Nz, int Nphi, int Ntheta, MassiveParticleDistribution** electronDistributions, double*** B, double*** sinTheta, double*** phi, double*** concentration, const double& rho, const double& rhoin, const double& distance, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными с помощью массивов магнитным полем, синусом угла его наклона к лучу зрения, а так же заданными внешним и внутренним радиусом шарового слоя, расстоянием до источника и скоростью движения вещества. Распределение частиц задается в виде массива табличных значений в зависимости от угла наклона магнитного поля к направлению распространения ударной волны

AngleDependentElectronsSphericalSource(int Nrho, int Nz, int Nphi, int Ntheta, MassiveParticleDistribution** electronDistributions, const double& B, const double& sinTheta, const double& phi, const double& concentration, const double& rho, const double& rhoin, const double& distance, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными однородными магнитным полем, синусом угла его наклона, а так же заданными внутренним и внешним радиусом шарового слоя, расстоянием до источника и скоростью движения вещества. Распределение частиц задается в виде массива табличных значений в зависимости от угла наклона магнитного поля к направлению распространения ударной волны
TabulatedSLSourceWithSynchCutoff	Класс для источников в форме шарового слоя с таблично заданными значениями параметров и учетом синхротронных потерь энергии частиц
TabulatedSLSourceWithSynchCutoff(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, double*** B, double*** theta, double*** concentration, const double& rho, const double& rhoin, const double& distance, const double& downstreamVelocity, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными с помощью массивов распределением частиц, магнитным полем, синусом угла его наклона к лучу зрения, а так же заданными внешним и внутренним радиусом шарового слоя, расстоянием до источника, скоростью конвекции частиц и скоростью движения вещества
TabulatedSLSourceWithSynchCutoff(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, const double& B, const double& concentration, const double& theta, const double& rho, const double& rhoin, const double& distance, const double& downstreamVelocity, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными однородными распределением частиц, магнитным полем, синусом угла его наклона, а так же заданными внутренним и внешним радиусом шарового слоя, расстоянием до источника, скоростью конвекции частиц и скоростью движения вещества
SectoralSphericalLayerSource	абстрактный класс для источников в форме сектора шарового слоя (дольки апельсина)
double getRhoin()	возвращает внутренний радиус шарового слоя
TabulatedSectoralSphericalLayerSource	Класс для источников в форме сектора шарового слоя с таблично заданными значениями параметров
TabulatedSectoralSphericalLayerSource(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, double*** B, double*** theta, double*** concentration, const double& rho, const double& rhoin, const double& minrho, const double& phi, const double& distance, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными с помощью массивов распределением частиц, магнитным полем, синусом угла его наклона к лучу зрения, а так же заданными внешним и внутренним радиусом шарового слоя, углом раствора сектора, расстоянием до источника и скоростью движения вещества

TabulatedSectoralSphericalLayerSource(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, const double& B, const double& concentration, const double& theta, const double& rho, const double& rhoin, const double& minrho, const double& phi, const double& distance, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными однородными распределением частиц, магнитным полем, синусом угла его наклона, а так же заданными внутренним и внешним радиусом шарового слоя, углом раствора сектора, расстоянием до источника и скоростью движения вещества
TabulatedSectoralSLSourceWithSynchCutoff	Класс для источников в форме сектора шарового слоя с таблично заданными значениями параметров и учетом синхротронных потерь энергии частиц
TabulatedSectoralSLSourceWithSynchCutoff(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, double*** B, double*** theta, double*** concentration, const double& rho, const double& rhoin, const double& minrho, const double& phi, const double& distance, const double& downstreamVelocity, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными с помощью массивов распределением частиц, магнитным полем, синусом угла его наклона к лучу зрения, а так же заданными внешним и внутренним радиусом шарового слоя, углом раствора сектора, расстоянием до источника, скоростью конвекции частиц и скоростью движения вещества
TabulatedSectoralSLSourceWithSynchCutoff(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, const double& B, const double& concentration, const double& theta, const double& rho, const double& rhoin, const double& minrho, const double& phi, const double& distance, const double& downstreamVelocity, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными однородными распределением частиц, магнитным полем, синусом угла его наклона, а так же заданными внутренним и внешним радиусом шарового слоя, углом раствора сектора, расстоянием до источника, скоростью конвекции частиц и скоростью движения вещества

1.2.2 Источники излучения, меняющиеся со временем

Источники излучения, учитывающие зависимость от времени, представлены абастрактным классом классом RadiationTimeDependentSource. Этот класс не является наследником класса RadiationSource, но содержит экземпляр такого класса внутри себя, чтобы использовать его для расчета излучения в конкретный момент времени. Для этого пользователь должен самостоятельно создать имплементацию виртуальной функции getRadiationSource, в которой будут вычислены параметры источника в зависимости от времени. В текущей версии кода реализован только один наследник RadiationTimeDependentSource - ExpandingRemnantSource, представляющий собой модель расширяющегося остатка сверхновой. В данной модели предполагается, что размер источ-

ника увеличивается во времени с постоянной скоростью, магнитное поле падает обратно пропорционально размеру источника, концентрация обратно пропорционально квадрату размера а толщина шарового слоя остается постоянной. Пользователь может создавать свои классы источников с другими зависимостями параметров от времени. Публичные методы классов RadiationTimeDependentSource и ExpandingRemnantSource перечислены в Таблице 1.6.

Table 1.6: Публичные методы классов источников излучения учитывающих зависимость от времени

RadiationTimeDependentSource	Абстрактный класс для учета изменений источников излучения со временем
virtual resetParameters(const double* parameters, const double* normalizationUnits)	чисто виртуальный метод, меняющий параметры источника. Список параметров, их количество, их влияние на источник определяются пользователем в конкретных реализациях класса. Принимает массив параметров и массив единиц в которых они измерены. Данный метод применяется в процедурах оптимизации
virtual getRadiationSource(double& time, const double* normalizationUnits)	возвращает источник излучения с параметрами соответствующими заданному моменту времени. Так же принимает на вход массив единиц, в которых измеряются параметры этого источника.
ExpandingRemnantSource	класс, представляющий модель расширяющегося с постоянной скоростью остатка сверхновой, имеющего форму шарового слоя постоянной толщины с однородными концентрацией и магнитным полем
ExpandingRemnantSource(const double& R0, const double& B0, const double& concentration0, const double& v, const double& widthFraction, RadiationSource* source, const double& t0)	конструктор, создает экземпляр класса расширяющейся сферической оболочки с заданными в момент t0 радиусом, магнитным полем, концентрацией, скоростью расширения, отношением толщины оболочки к радиусу и моделью источника. Для коректного учета изменения источника во времени важно, чтобы конретная реализация метода source->resetParameters соответствовала той,что используется в методе getRadiationSource. В данном случае подходят все перечисленные выше реализации источников не зависящих от времени

Chapter 2

Evaluation of radiation

In current version of the code following types of radiation are implemented: synchrotron radiation, inverse Compton scattering, gamma-ray emission due to pion decay in free-free proton interaction and also bremsstrahlung.

Для расчета излучения источников используется абстрактный класс RadiationEvaluator и его наследники, предназначенные для конкретных видов излучения. Так же есть класс RadiationSumEvaluator, предназначенный для суммирования нескольких различных видов излучения. Список публичных методов этих двух классов приведен в Таблице 2.1. Общая схема расчета излучения такова: создать источник излучения, используя один из классов описанных в предыдущем разделе или написанный самостоятельно, затем создать вычислитель излучения нужного типа, и вызвать у него метод evaluateFluxFromSource(const double& photonFinalEnergy, RadiationSource* source), вычисляющий энергетическую плотность потока излучения источника на данной энергии принимаемого фотона в единицах $см^{-2}с^{-1}$. Далее в данном разделе описаны реализации класса RadiationEvaluator для конкретных видов излучения. Схема наследования классов вычислителей излучения представлена на рисунке 2.1. Физическая сторона вопроса, формулы по которым расчитывается излучение подробно описаны в Главе 4.

Table 2.1: Публичные методы класса RadiationEvaluator

RadiationEvaluator	абстрактный класс для вычисления излучения
virtual evaluateFluxFromSource(const double& photonFinalEnergy, RadiationSource* source)	чисто виртуальный метод, возвращает энергетическую плотность потока излучаемого данным источником в единицах $см^{-2}с^{-1}$
virtual double evaluateFluxFromSourceAtPoint(const double& photonFinalEnergy, RadiationSource* source, int rhoi, int phi) double evaluateTotalFluxInEnergyRange(const double& Ephmin, const double& Ephmax, int Nph, RadiationSource* source)	чисто виртуальный метод, возвращает энергетическую плотность потока, излучаемого данной областью источника на картинной плоскости возвращает интегральны поток излучаемый источником в заданном диапазоне энергий (проинтегрированный по Nph точкам) в единицах $эргсм^{-2}с^{-1}$

virtual resetParameters(const double* parameters, const double* normalizationUnits)	чисто виртуальный метод, позволяет изменить внутренние параметры вычислителя излучения. Список параметров, их количество, их влияние на источник определяются в конкретных реализациях класса, данный метод используется при оптимизации
writeFluxFromSourceToFile(const char* fileName, RadiationSource* source, const double& Ephmin, const double& Ephmax, const int Nph)	записывает в файл с данным именем излучение источника в единицах $см^{-2}с^{-1}$ в диапазоне от минимальной до максимальной энергии, с заданным количеством точек, распределенных логарифмически
void writeImageFromSourceToFile(const char* fileName, RadiationSource* source, const double& Ephmin, const double& Ephmax, const int Nph)	записывает в файл с данным именем изображение - двумерный массив с интегральным потоком излучаемым разными областями источника в единицах $эргсм^{-2}с^{-1}$ в диапазоне от минимальной до максимальной энергии, проинтегрированым по заданному количеству точек, распределенных логарифмически
void writeImageFromSourceAtEToFile(const double& photonFinalEnergy, const char* fileName, RadiationSource* source)	записывает в файл с данным именем изображение - двумерный массив с энергетической плотностью потока излучаемого разными областями источника на данных энергиях в единицах $см^{-2}с^{-1}$
RadiationSumEvaluator	класс предназначенный для суммирования нескольких видов излучения
RadiationSumEvaluator(int Ne, const double& Emin, const double& Emax, RadiationEvaluator* evaluator1, RadiationEvaluator* evaluator2)	конструктор, создающий экземпляр с указанным диапазоном рассматриваемых энергий излучающих частиц, вычисляющий и складывающий результаты двух указанных вычислителей
RadiationSumEvaluator(int Ne, const double& Emin, const double& Emax, int Nev, RadiationEvaluator** evaluators)	конструктор, создающий экземпляр с указанным диапазоном рассматриваемых энергий излучающих частиц, вычисляющий и складывающий результаты вычислителей излучения в указанном массиве

2.1 Синхротронное излучение

Для расчета синхротронного излучения используется класс SynchrotronEvaluator. В нем используется приближение непрерывного спектра, то есть рассматриваемые частоты фотонов предполагаются намного большими, чем частота вращения излучающих частиц

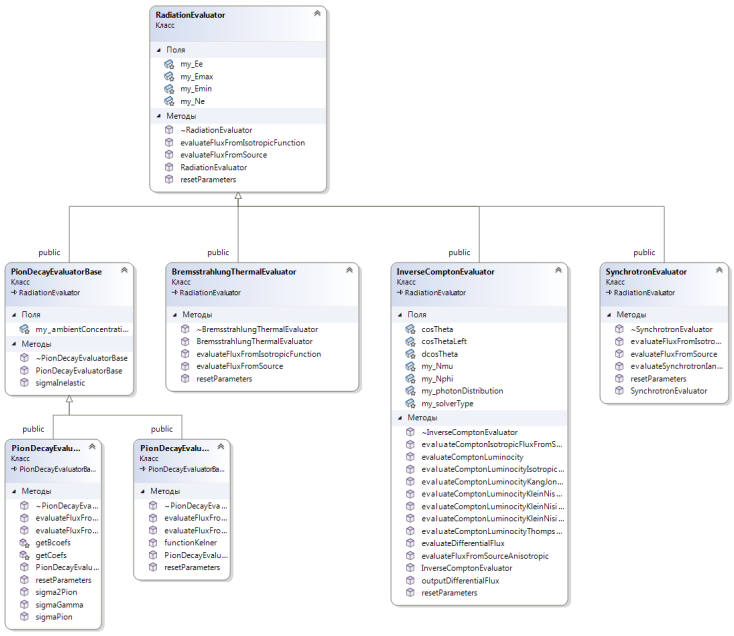

Figure 2.1: Схема наследования классов вычислителей излучения.

в магнитном поле. Реализован случай только изотропной функции распределения излучающих частиц. Так же возможен учет синхротронного самопоглощения. Используемая геометрия источников, показанная на рисунке 1.5, позволяет легко интегрировать излучение по лучу зрения, и учитывать при этом поглощение внутри источника. При создании объекта класса необходимо указать рассматриваемый диапазон энергий частиц и количество точек в нем, параметр отвечающий за учет самопоглощения (значение по умолчанию true), а так же значения магнитного поля, синуса угла наклона к лучу зрения и толщины излучаемой области, которые будут использоваться в случае расчета излучения без указания источника, а только с использованием распределения частиц. Публичные методы класса SynchrotronEvaluator перечислены в Таблице 2.2. Пример вычисления синхотронного излучения приведен в разделе **??**.

2.2 Обратное комптоновское рассеяние

Для расчета излучения, получающегося в результате процесса обратного комптоновского рассеяния, используются классы InverseComptonEvaluator и его наследник InverseComptonEvaluatorWithSource. Отличие между ними в том, что в первом функция распределения рассеиваемых фотонов одинакова во всем излучающем объеме, а во втором изменяется обратно пропорционально квадрату расстояния до источника фотонов. Внутри класса InverseComptonEvaluator реализованы четыре различных метода расчета излучения, для обозначения которых используется перечислимый тип ComptonSolverType, имеющий следующие значения:

Table 2.2: Публичные методы класса SynchrotronEvaluator

SynchrotronEvaluator	класс предназначенный для вычисления синхротронного излучения
SynchrotronEvaluator(int Ne, double Emin, double Emax, bool selfAbsorption = true, bool doppler = false)	конструктор, создает экземпляр с указанным диапазоном рассматриваемых энергий излучающих частиц,и параметрами учета самопоглощения и допплеровского эффекта
evaluateSynchrotronIandA(const double& photonFinalFrequency, const double& photonFinalTheta, const double& photonFinalPhi, const double& B, const double& sinhi, const double& concentration, MassiveParticleIsotropicDistribution* electronDistribution, double& I, double& A)	вычисляет значения плотности излучательной способности и коэффициента поглощения для фотона с данной энергией и направлением, в области с данными концентрацией и распределением излучающих частиц в данном магнитном поле

- ISOTROPIC_THOMSON - модель рассеяния в томсоновсков режиме. Реализовано только для степенного распределения электронов и теплового фотонов [6] глава 17, с. 466
- ANISOTROPIC_KLEIN_NISHINA - модель расчитывающее излучение напрямую из сечения Клейна-Нишины, возможен учет анизотропных функций распределения [7, 8]
- ISOTROPIC_KLEIN_NISHINA - модель расчитывающее излучение напрямую из сечения Клейна-Нишины, но для изотропных функций распределения, что позволяет уменьшить количество интегрирований
- ISOTROPIC_JONES - модель, использующая аналитически проинтегрированное по углам сечение Клейна-Нишины [9, 10]

При создании объекта класса InverseComptonEvaluator необходимо указать рассматриваемый диапазон энергий частиц и количество точек в нем, количество ячеек в сетке по полярному и азимутальному углу, изотропную функцию распределения фотонов, которая будет использоваться по умолчанию и метод расчета излучения. Публичные методы классов InverseComptonEvaluator и InverseComptonEvaluatorWithSource перечислены в Таблице 2.3.

Table 2.3: Публичные методы класса InverseComptonEvaluator

InverseComptonEvaluator	класс предназначенный для вычисления излучения рождащегося в результате обратного комптоновского рассеяния

InverseComptonEvaluator(int Ne, int Nmu, int Nphi, double Emin, double Emax, PhotonDistribution* photonDistribution, ComptonSolverType solverType)	конструктор, создает экземпляр с заданным рассматриваемым диапазоном энергии, количеством ячеек в сетке по полярному и азимутальному углу, функцией распределения фотонов, которая будет использоваться по умолчанию и методом расчета излучения
evaluateComptonFluxKleinNishinaAnisotropic const double& photonFinalEnergy, const double& photonFinalTheta, const double& photonFinalPhi, PhotonDistribution* photonDistribution, MassiveParticleDistribution* electronDistribution, const double& volume, const double& distance)	возвращает энергетическую плотность потока энергии в заданном направлении, излучением созданным заданными функциями распределения фотонов и рассеивющих частиц (которые могут быть анизотропными) в заданном объеме на данном расстоянии
evaluateFluxFromSourceAnisotropic(const double& photonFinalEnergy, const double& photonFinalTheta, const double& photonFinalPhi, PhotonDistribution* photonDistribution, RadiationSource* source)	возвращает энергетическую плотность потока энергии в заданном направлении, излучением созданным заданными распределения фотонов и источником, содержащим распределения рассеивающих частиц
InverseComptonEvaluatorWithSource	класс предназначенный для вычисления излучения рождащегося в результате обратного комптоновского рассеяния с учетом зависимости функции распределения фотонов от расстояния до их источника
InverseComptonEvaluatorWithSource(int Ne, int Nmu, int Nphi, double Emin, double Emax, double Ephmin, double Ephmax, PhotonDistribution* photonDistribution, ComptonSolverType solverType, const double& sourceR, const double& sourceZ, const double& sourcePhi)	конструктор, создает экземпляр с заданным рассматриваемым диапазоном энергии, количеством ячеек в сетке по полярному и азимутальному углу, функцией распределения фотонов, методом расчета излучения и координатами источника фотонов

Пример вычисления излучения от обратного комптоновского рассеяние содержится в процедуре evaluateComtonWithPowerLawDistribution() в файле examples.cpp. В ней расчитывается рентгеновское излучение, исходящее от объекта CSS161010 при рассеивании степенного распределения электронов, определенного в работе [11], на среднегалактическом распределении фотонов. Сначала определим переменные, задающие основные параметры источника - концентрацию частиц, его размер и магнитное поле. Для вычисления обратного комптоновского рассеяния магнитное поле не используется, но в источнике нужно его задать, поэтому положим его равным нулю. Так же зададим параметры сетки по энергиям и углам, которая будет использоваться вычислителем

```
double electronConcentration = 150;
double sinTheta = 1.0;
double rmax = 1.3E17;
```

```
double B = 0.0;
double distance = 150*1E6*parsec;

double Emin = me_c2;
double Emax = 1000 * me_c2;
int Ne = 200;
int Nmu = 20;
int Nphi = 4;
```

Далее создадим распределение фотонов, воспользовавшись статическим методом класса MultiPlankDistribution getGalacticField, который возвращает среднегалактическое фотонное распределение, и распределение электронов - возьмем степенное рспределение с показателем 3.5.

```
PhotonIsotropicDistribution* photonDistribution =
    PhotonMultiPlankDistribution::getGalacticField();
MassiveParticlePowerLawDistribution* electrons = new
    MassiveParticlePowerLawDistribution(massElectron, 3.5,
    Emin, electronConcentration);
```

С помощью введенных ранее переменных создадим источник излучения и вычислитель излучения. В качестве метода расчета выберем самый универсальный - ANISOTROPIC_KLEIN_NISHINA

```
RadiationSource* source = new SimpleFlatSource(
  electrons, B, sinTheta, rmax, rmax, distance);

InverseComptonEvaluator* comptonEvaluator = new
    InverseComptonEvaluator(Ne, Nmu, Nphi, Emin, Emax,
    photonDistribution, ComptonSolverType::ANISOTROPIC_KLEIN_NISHINA)
```

Предположим, что мы не хотим пользоваться встроенным методом вывода излучения в файл, так как хотим получить конечный результат в других единицах, например энергию фотона измерять в электронвольтах, а поток вывести в формате $EF(E)$ - эргсм$^{-2}$с$^{-1}$. Создадим тогда сетку значений энергии фотонов

```
int Nnu = 200;
double* E = new double[Nnu];
double* F = new double[Nnu];
double Ephmin = 0.01 * kBoltzman * 2.725;
double Ephmax = 2 * Emax;
double factor = pow(Ephmax / Ephmin, 1.0 / (Nnu - 1));
E[0] = Ephmin;
```

```
F[0] = 0;
for (int i = 1; i < Nnu; ++i) {
        E[i] = E[i - 1] * factor;
        F[i] = 0;
}
```

после этого вычислим в цикле желаемые потоки излучения

```
for (int i = 0; i < Nnu; ++i) {
        F[i] = comptonEvaluator->evaluateFluxFromSource(
            E[i], source);
}
```

и запишем их в файл, переведя в желаемые единицы

```
FILE* output_ev_EFE = fopen("output.dat", "w");

for (int i = 0; i < Nnu; ++i) {
        double nu = E[i] / hplank;
        fprintf(output_ev_EFE, "%g %g\n",
            E[i] / (1.6E-12), E[i] * F[i]);
}

fclose(output_ev_EFE);
```

Спектр излучения, полученный в результате работы данной программы приведен на рисунке 2.2

2.3 Распад пионов

Для расчета излучения, получающегося в результате распада пионов, родившихся в результате свободно-свободного взаимодействия протонов используются абастрактный класс PionDecayEvaluatorBase и двае его наследника: PionDecayEvaluatorKelner, в котором сечение излучения гамма-фотона считается долей от полного сечения неупругого взаимодействия протонов, как описано в статье [12], и PionDecayEvaluator, в котором используется более точное описание сечения рождения пионов на низких энергиях по методу, описанному в [13]. В текущей версии предполагается, что характерное время потерь энергии протонов при неупругом взаимодействии намного больше времени их удержания в источнике, система является прозрачной для протонов, и каждый из них взаимодействует не более одного раза. В противном случае используемая модель излучения не применима.

При создании объекта класса PionDecayEvaluator необходимо указать рассматриваемый диапазон энергий частиц и количество точек в нем, а так же концентрацию фоно-

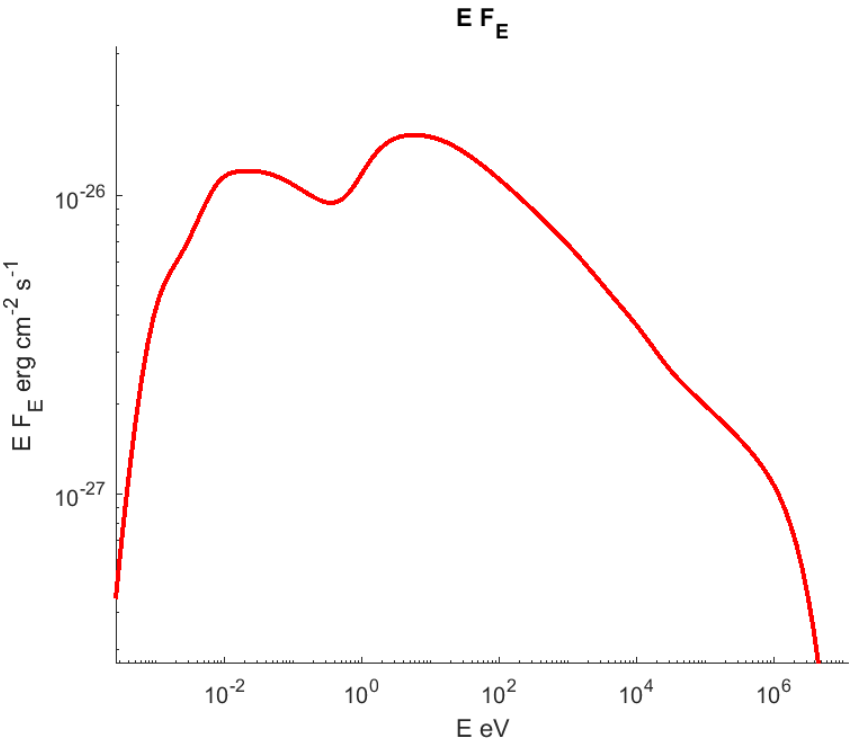

Figure 2.2: Энергетическая плотность потока синхротронного излучения от тестового источника

вых протонов, так как предполагается рассеяние высокоэнергичных фотонов на покоящихся, а не взаимодействие высокоэнергичных между собой. Публичные методы класса PionDecayEvaluatorBase и его наследников приведены в Таблице 2.4

Table 2.4: Публичные методы класса PionDecayEvaluatorBase и его наследников

PionDecayEvaluatorBase	абстрактный класс для вычисления гамма излучения от распада пионов
sigmaInelastic(const double& energy)	возвращает полное сечение неупругого взаимодействия протонов в лабораторной системе, принимает кинетическую энергию движущегося протона
PionDecayEvaluatorKelner	класс для вычисления гамма излучения от распада пионов по методу из статьи [12]
PionDecayEvaluatorKelner(int Ne, double Emin, double Emax, const double& ambientConcentration)	конструктор, создает экземпляр с заданным рассматриваемым диапазоном энергии и концентрацией фоновых протонов
PionDecayEvaluator	класс для вычисления гамма излучения от распада пионов по методу из статьи [13]

PionDecayEvaluator(int Ne, double Emin, double Emax, const double& ambientConcentration)	конструктор, создает экземпляр с заданным рассматриваемым диапазоном энергии и концентрацией фоновых протонов
sigmaGamma(const double& photonEnergy, const double& protonEnergy)	возвращает дифференциальное сечение рождения фотона с данной энергией при данной кинетической энергии протона, усредненное по углам

Пример вычисления излучения от гамма излучения от распада пионов показан в функции evaluatePionDecay() в файлк examples.cpp. В нем рассмотрено моделирование излучение объекта Кокон Лебедя в модели ускорения частиц на вторичных ударных волнах, следуя статье [14]. В данной работе вычислено, что спектр ускоренных протонов имеет вид степенной функции с изломом со следующими параметрами - показатели спектра 2.1 и 2.64 на низких и высоких энергиях соответственно, энергия излома - 2.2 ТэВ. Размер излучающей области брался равным размеру сверхкаверны Лебедя - 55 пк. Как и ранее, сначала определим переменные, задающие основные параметры источника - концентрацию частиц, его размер и магнитное поле, которое опять положим равным нулю. Диапазон энергий протонов рассмотрим от 0.01 ГэВ до 10 ТэВ. Так же укажем энергию излома.

```
double protonConcentration = 150;
double rmax = 55 * parsec;
double B = 0;
double sinTheta = 1.0;

double distance = 1400 * parsec;
double Emin = massProton*speed_of_light2 + 0.01E9 * 1.6E-12;
double Emax = 1E13 * 1.6E-12;
double Etrans = 2.2E12 * 1.6E-12;
```

После этого создадим распределение протонов и источник излучения

```
MassiveParticleBrokenPowerLawDistribution* protons = new
        MassiveParticleBrokenPowerLawDistribution(
        massProton, 2.1, 2.64, Emin, Etrans, protonConcentration);
RadiationSource* source = new SimpleFlatSource(
        protons, B, sinTheta, rmax, rmax, distance);
```

Далее потребуется вычислитель излучения. В случае пионного распада необходимо указать концентрацию фоновых протонов.

```
double protonAmbientConcentration = 20;
PionDecayEvaluator* pionDecayEvaluator = new PionDecayEvaluator(
        200, Emin, Emax, protonAmbientConcentration);
```

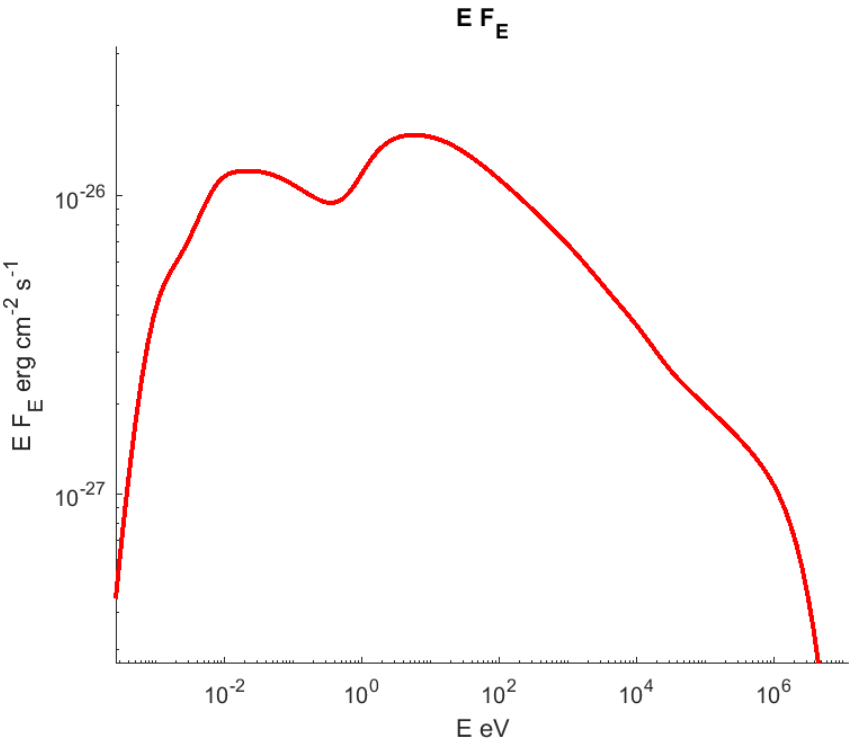

Figure 2.3: Расчетная энергетическая плотность потока гамма излучения Кокона Лебедя и данные наблюдений

Как и в предыдущих случаях далее необходимо внутри цикла вычислить излучение в интересующем диапазоне энергий, используя функцию evaluateFluxFromSource, и вывести результат в файл в удобных единицах. Спектр излучения, полученный в результате работы данной программы и результаты наблюдений Кокона Лебедя на Fermi LAT, ARGO и HAWC [15, 16, 17] приведены на рисунке 2.3

2.4 Тормозное излучение

В текущей версии кода реализовано вычисление тормозного излучения электронов в плазме только для случая теплового распределения. Для этого предназначен класс BremsstrahlungThermalEvaluator. В процессе расчета предполагается, что плазма электрон-протонная, с одинаковыми температурами электронов и протонов, в вычислении используются Гаунт-факторы, приведенные в [18]. Пример вычисления тормохного излучения приведен в функции evaluateBremsstrahlung в файле examples.cpp.

Chapter 3

Оптимизация параметров

Код FAINA позволяет не только расчитывать излучение заданных источников, но и фитировать наблюдательные данные модельными, подбирая необходимые параметры. Реализованы методы оптимизации, пригодные для произвольного числа параметров и широкого класса моделей источников.

3.1 Вычислители целевой функции

Для оптимизации в первую очередь нужно указать целевую функцию, которую требуется минимизировать. Для этой цели используется абстрактный класс LossEvaluator и его наследники. В реализованных наследниках используются квадратичные функции оптимизации, взвешенные с учетом наблюдательных ошибок $f = \sum \frac{(F(x_i)-F_{obs,i})^2}{\sigma_i^2}$, где $F(x_i)$ - некая расчетная функция излучения (спектральная плотность потока, например) вычисленная при неком фиксированном значении параметра x_i (например при данной частоте), $F_{obs,i}$ - наблюдаемая величина этой функции, σ_i - её погрешность. В коде реализованы следующие классы целевых функций SpectrumLossEvaluator - для фитирования спектральной плотности потока излучения в данный момент времени, TimeDependentSpectrumLossEvaluator - для фитирования спектральной плотности потока излучения переменного источника, измеренного в несколько моментов времени и RadialProfileLossEvaluator - для фитирования зависимости яркости различных точек источника в зависимости от радиуса. Публичные методы этих классов перечислены в Таблице 3.1.

Table 3.1: Публичные методы классов вычислителей целевых функций

LossEvaluator	абстрактный класс вычислителя целевой функции
virtual double evaluate(const double* vector, const double* maxParameters, RadiationEvaluator* evaluator)	чисто виртуальный метод, возвращающий значение целевой функции при данных параметрах. Так же на вход принимает вектор нормироваочных значений и вычислитель излучения

SpectrumLossEvaluator	класс, в котором целевая функция характеризует отличие вычисленной спектральной плотности излучения от наблюдательных данных, $f = \sum \frac{(F(E_i) - F_{obs,i})^2}{\sigma_i^2}$, где $F(E_i)$ - расчетная спектральная плотность потока излучения при данной энергии E_i, $F_{obs,i}$ - наблюдаемая спектральная плотность потока излучения, σ_i - её погрешность.
SpectrumLossEvaluator(double* energy, double* observedFlux, double* observedError, int Ne, RadiationSource* radiationSource)	конструктор, принимает на вход наблюдаемые значения спектральной плотности энергии, их количество и источник, для которого нужно вычислять излучение в каких единицах?
TimeDependentSpectrumLossEvaluator	класс, в котором целевая функция характеризует отличие вычисленной спектральной плотности излучения от наблюдательных данных, собранных в различные моменты времени, $f = \sum \frac{(F(E_{ij}, t_j) - F_{obs,i,j})^2}{\sigma_{ij}^2}$, где $F(E_{ij}, t_j)$ - расчетная спектральная плотность потока излучения при данной энергии E_{ij} в момент времени t_j, $F_{obs,i,j}$ - наблюдаемая спектральная плотность потока излучения, σ_{ij} - её погрешность. ОБРАТИТЕ ВНИМАНИЕ, что количество наблюдательных точек в разные моменты времени может быть разным
TimeDependentSpectrumLossEvaluator(double** energy, double** observedFlux, double** observedError, int* Ne, double* times, int Ntimes, RadiationTimeDependentSource* radiationSource)	конструктор, принимает на вход двумерные массивы наблюдаемых значений спектральной плотности энергии, массив количества точек в разные моменты времени и зависящий от времени источник, для которого нужно вычислять излучение в каких единицах?
RadialProfileLossEvaluator	класс, в котором целевая функция характеризует отличие радиальной зависимости яркости источника в заданном дипазоне от наблюдательных данных. $f = \sum \frac{(F(R_i) - F_{obs,i})^2}{\sigma_i^2}$, где $F(R_i)$ - расчетная яркость источника при данном радиусе R_i, $F_{obs,i}$ - наблюдаемая яркость, σ_i - её погрешность

RadialProfileLossEvaluator(double energy, double* observedFlux, double* observedError, double* rhoPoints, int Nrho, RadiationSource* radiaionSource)	конструктор, принимающий на вход значение энергии, для которого нужно вычислять яркость, наблюдаемые потоки, погрешности и соответствующие значения радиуса, количество точек и источник, для которого нужно рассчитывать излучение

3.2 Оптимизаторы целевой функции

Для фитирования постоянных во времени кривых блеска предназначен абстрактный класс RadiationOptimizer. В нем определена виртуальныя функция optimize(double* vector, bool* optPar), которая и производит процесс оптимизации. Входными параметрами являются: vector - массив подбираемых параметров, в который будет записан результат работы программы, optPar - массив булевских переменных, определяющих оптимизировать соответствующий параметр, или считать его фиксированным. Функция изменения параметров источника source->resetParameters, который будет использоваться в процессе оптимизации, описанная в разделе 1.2.1, должна быть согласована с массивом оптимизируемых параметров vector, так как в процессе оптимизации он будет передаваться в нее в качестве аргумента.

В коде реализованы три наследника класса RadiationOptimazer: GridEnumRadiationOptimizer - производящий поиск минимума простым перебором по сетке параметров с заданным количеством распределенных равномерно логарифмически точек, GradientDescentRadiationOptimizer - в котором минимум находится методом градиентного спуска, и CombinedRadiationOptimizer, который выполняет оптимизацию двумя этими методами последовательно, используя результат работы первого как начальную точку для второго. Схема насследования классов оптимизаторов показана на рисунке 3.1, а список их публичных методов приведен в Таблице 3.2. Реализованные методы оптимизации применимы для всех описанных выше типов источников, видов электромагнитного излучения и вычислителей целевых функций.

Table 3.2: Публичные методы классов оптимизаторов параметров источников

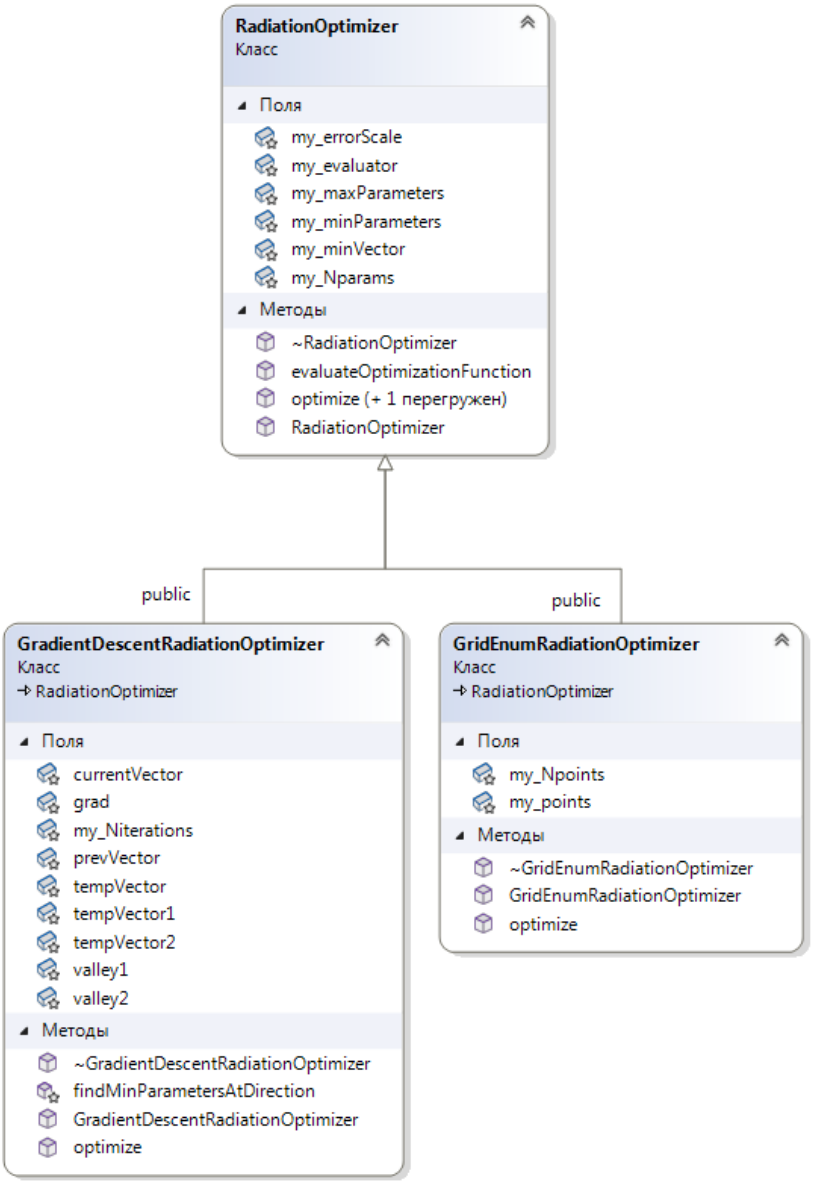

Figure 3.1: Схема наследования классов оптимизаторов

RadiationOptimizer	абстрактный класс для оптимизации параметров источника
double evaluateOptimizationFunction(const double* vector)	вычисляет целевую функцию - взвешенную при данном векторе параметров
void optimize(double* vector, bool* optPar)	функция, осуществляющая оптимизацию, принимает на вход массив подбираемых параметров, в который будет записан результат и массив булевских переменных, определяющих оптимизировать соответствующий параметр, или считать его фиксированным
void outputProfileDiagrams(const double* vector, int Npoints)	функция, которая строит и записывает в файлы двухмерные сечения целевой функции по всем комбинациям параметров, проходящие через точку определяемую заданным вектором параметров
void outputOptimizedProfileDiagram(const double* vector, bool* optPar, int Npoints, int Nparam1, int Nparam2)	функция, которая строит и записывает в файлы двухмерные сечения целевой функции, в которых два параметра с номерами Nparam1 и Nparam2 фиксируются и пробегают соответствующую плоскость, а остальные оптимизируются
GridEnumRadiationOptimizer	класс предназначенный для оптимизации параметров с помощью перебора по сетке
GridEnumRadiationOptimizer(RadiationEvaluator* evaluator, const double* minParameters, const double* maxParameters, int Nparams, const int* Npoints, LossEvaluator* lossEvaluator)	конструктор, создает экземпляр класса с указанным вычислителем излучения, минимальными и максимальными значениями оптимизируемых параметров, количеством этих параметров, массивом с количеством перебираемых точек по каждому параметру и вычислителем целевой функции. При переборе точки будут распределены логарифмически по оси.
GradientDescentRadiationOptimizer	класс, предназначенный для оптимизации параметров методом градиентного спуска
GradientDescentRadiationOptimizer(RadiationEvaluator* evaluator, const double* minParameters, const double* maxParameters, int Nparams, int Niterations, LossEvaluator* lossEvaluator)	конструктор, создает экземпляр класса с указанным вычислителем излучения, минимальными и максимальными значениями оптимизируемых параметров, количеством этих параметров, максимальным количеством итераций градиентного спуска и вычислителем целевой функции

CombinedRadiationOptimizer	класс, предназначенный для совместного использования сеточного поиска и градиентного спуска
CombinedRadiationOptimizer(RadiationEvaluator* evaluator, const double* minParameters, const double* maxParameters, int Nparams, int Niterations, const int* Npoints, LossEvaluator* lossEvaluator)	конструктор, создает экземпляр класса с указанным вычислителем излучения, минимальными и максимальными значениями оптимизируемых параметров, количеством этих параметров, максимальным количеством итераций градиентного спуска, количеством точек для сеточного поиска и вычислителем целевой функции

Пример фитирования параметров источника по наблюдательным данным приведен в функции fitCSS161010withPowerLawDistribition в файле examples.cpp. Следуя авторам работы [11] произведем расчет синхротронного излучения источника с учетом самопоглощения, считая функцию распределения электронов чисто степенной с показателем 3.6. Но мы не будем накладывать дополнительную связь на параметры и предполагать равенство распределения энергии между магнитным полем и ускоренными частицами, вместо этого магнитное поле и концентрация электронов будут независимыми параметрами.

Подберем параметры Быстрого Оптического Голубого Транзиента CSS161010 на 98 день после вспышки на основе радиоизлучения. Зададим параметры источника на основе дынных статьи [11], которые будут использоваться в качестве начального приближения, а так же расстояние до него.

```
double electronConcentration = 25;
double B = 0.6;
double R = 1.4E17;
double fraction = 0.5;
const double distance = 150 * 1E6 * parsec;
```

Далее зададим степенное распределение электронов, с показателем 3.6 и источник в форме плоского диска, перпендикулярного лучу зрения, и вычислитель синхротронного излучения.

```
double Emin = me_c2;
double Emax = 10000 * me_c2;
double index = 3.6;

SynchrotronEvaluator* synchrotronEvaluator = new
    SynchrotronEvaluator(200, Emin, Emax);

MassiveParticlePowerLawDistribution* electrons =
    new MassiveParticlePowerLawDistribution(
    massElectron, index, Emin, electronConcentration);
```

```
SimpleFlatSource* source = new
    SimpleFlatSource(electrons, B, pi/2, R, fraction * R, distance);
```

Теперь определим вектор оптимизируемых параметров - это размер, магнитное поле, концентрация электронов и доля толщины, показывающая какую долю от радиуса диска составляет его толщина. И именно такие параметры ожидает функция resetParameters у источника SimpleFlatSource. Так же нужно указать минимальные и максимальные значения параметров, которые ограничат область поиска. Максимальные значения так же будут использоваться как константы нормировки.

```
const int Nparams = 4;
double minParameters[Nparams] = { 1E17, 0.01, 0.5, 0.1 };
double maxParameters[Nparams] = { 2E17, 10, 200, 1.0 };
double vector[Nparams] = { R, B, electronConcentration, fraction};
for (int i = 0; i < Nparams; ++i) {
        vector[i] = vector[i] / maxParameters[i];
}
```

Зададим наблюдательные данные, которые и будем фитировать. Обратите внимание, что частоты нужно перевести в энергии, а спектральную плотность потока - в энергетическую (в единицы $см^{-2}с^{-1}$).

```
const int Nenergy1 = 4;
double energy1[Nenergy1] = { 1.5E9*hplank, 3.0E9 * hplank,
    6.1E9 * hplank, 9.8E9 * hplank };
double observedFlux[Nenergy1] = { 1.5/(hplank*1E26),
    4.3/(hplank*1E26), 6.1/(hplank*1E26), 4.2 /(hplank*1E26)};
double observedError[Nenergy1] = { 0.1 / (hplank * 1E26),
    0.2/(hplank*1E26), 0.3/(hplank*1E26), 0.2/(hplank*1E26)};
```

Далее создадим вычислитель целевой функции, фитирующий спектр и комбинированный оптимизатор, и укажем количество точек для перебора и количество итераци градиентного спуска. Так же укажем, что оптимизируем все параметры.

```
bool optPar[Nparams] = { true, true, true, true };
int Niterations = 20;
int Npoints[Nparams] = { 10,10,10,10 };

LossEvaluator* lossEvaluator = new SpectrumLossEvaluator(energy1, observe
RadiationOptimizer* optimizer = new CombinedRadiationOptimizer(
    synchrotronEvaluator,minParameters,maxParameters,Nparams, Niterations
```

Применим функцию optimize и изменим параметры источника на оптимальные

```
optimizer->optimize(vector, optPar, energy1, observedFlux,
```

```
        observedError, Nenergy1, source);
    source->resetParameters(vector, maxParameters);
```

Полученные в результате оптимизации парметры источника равны: радиус диска $R = 1.8 \times 10^17$ см, магнитное поле $B = 1.6$ Гс, концентрация электронов $n = 2.3$ см$^{-3}$, доля толщины $fraction = 0.54$. Значение целевой функции $f \approx 50$. Модельный спектр излучения с данными параметрами и наблюдательные данные изображены на рисунке 3.2.

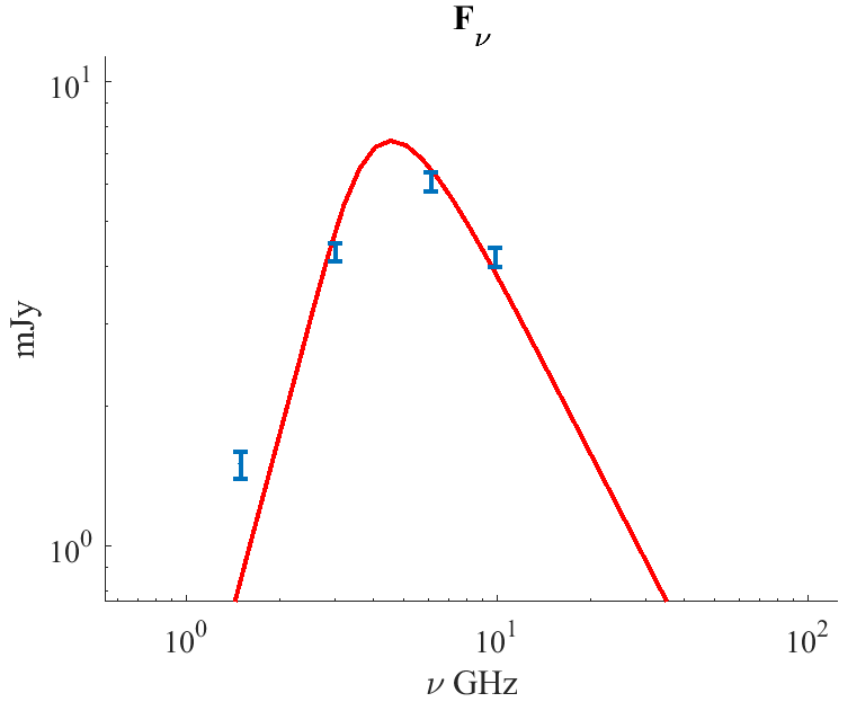

Figure 3.2: Наблюдаемый и расчетный спектр радиоизлучения объекта CSS161010 на 98 день после вспышки

Chapter 4

Формулы расчета излучения

4.1 Преобразование функции распределения фотонов

Функция распределения фотонов задана в сферических координатах $n_{ph}(\epsilon, \mu, \phi)$. Рассмотрим переход в систему отсчета, движущуюся в направлении оси z с лоренц-фактором $\gamma = 1/\sqrt{1-\beta^2}$. Количество частиц в элементе фазового пространства N - инвариант.

$$N = n_{ph}(\epsilon, \mu, \phi) d\epsilon d\mu d\phi dV = n'_{ph}(\epsilon', \mu', \phi') d\epsilon' d\mu' d\phi' dV' \tag{4.1}$$

Рассмотрим преобразование вектора четырех-импульса. Поперечные компоненты не изменяются, а временная и продольная меняются следующим образом, учитывая что $p_z = \mu\epsilon$:

$$\begin{pmatrix} \epsilon' \\ \mu'\epsilon' \end{pmatrix} = \begin{pmatrix} \gamma & -\beta\gamma \\ -\beta\gamma & \gamma \end{pmatrix} \times \begin{pmatrix} \epsilon \\ \mu\epsilon \end{pmatrix} \tag{4.2}$$

Из первой строчки матрицы получаем уравнение для допплеровского сдвига энергии

$$\epsilon' = \gamma(1 - \mu\beta)\epsilon \tag{4.3}$$

Вычислим производные новой энергии по старым координатам

$$\frac{d\epsilon'}{d\epsilon} = \gamma(1 - \mu\beta) \tag{4.4}$$

$$\frac{d\epsilon'}{d\mu} = -\gamma\beta\epsilon \tag{4.5}$$

Из второй строчки матрицы получаем $\mu'\epsilon' = -\beta\gamma\epsilon + \gamma\mu\epsilon$. Подставив значение ϵ' из 4.3 и сократив ϵ получим уравнение аберрации света

$$\mu' = \frac{\mu - \beta}{1 - \mu\beta} \tag{4.6}$$

Заметим, что угол наклона луча в новой системе не зависит от энергии в старой системе. Вычислим частную производную $\frac{d\mu'}{d\mu}$

$$\frac{d\mu'}{d\mu} = \frac{d}{d\mu}\frac{1}{\beta}\frac{\beta\mu - 1 + 1 - \beta^2}{1 - \mu\beta} = \frac{d}{d\mu}\frac{1}{\beta}\frac{1-\beta^2}{1-\mu\beta} = \frac{1-\beta^2}{(1-\mu\beta)^2} = \frac{1}{\gamma^2(1-\mu\beta)^2} \tag{4.7}$$

Азимутальный угол не зависит от системы отсчета $\phi' = \phi$. Преобразование элемента объема описывается выражением $\frac{dV'}{dV} = \frac{\epsilon}{\epsilon'}$ см. ЛЛ Т2 параграф 10, вот только там используется переход в собственную систему. То есть

$$\frac{dV'}{dV} = \frac{1}{\gamma(1 - \mu\beta)} \tag{4.8}$$

Матрица якоби преобразования координат выглядит следующим образом

$$J = \begin{pmatrix} \frac{d\epsilon'}{d\epsilon} & \frac{d\epsilon'}{d\mu} & 0 & 0 \\ 0 & \frac{d\mu'}{d\mu} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \frac{dV'}{d\mu} & 0 & \frac{dV'}{dV} \end{pmatrix} \tag{4.9}$$

При такой матрице якобиан, к счастью, равен произведению диагональных членов

$$\frac{D(\epsilon', \mu', \phi', V')}{D(\epsilon, \mu, \phi, V)} = \frac{d\epsilon'}{d\epsilon}\frac{d\mu'}{d\mu}\frac{dV'}{dV} = \gamma(1-\mu\beta)\frac{1}{\gamma^2(1-\mu\beta)^2}\frac{1}{\gamma(1-\mu\beta)} = \frac{1}{\gamma^2(1-\mu\beta)^2} \tag{4.10}$$

И в итоге функция распределения фотонов преобразуется с помощью деления на вычисленный якобиан

$$n'_{ph}(\epsilon', \mu', \phi') = \frac{n_{ph}(\epsilon, \mu, \phi)}{\frac{D(\epsilon', \mu', \phi', V')}{D(\epsilon, \mu, \phi, V)}} = \gamma^2(1-\mu\beta)^2 n_{ph}(\epsilon, \mu, \phi) \tag{4.11}$$

4.2 Комптоновское рассеяние

Рассмотрим рассеяние фотонов на одном электроне, движущемся вдоль ось z, см [8]. Сечение Клейна-Нишины в системе покоя электрона равно

$$\frac{d\sigma}{d\epsilon'_1 d\Omega'_1} = \frac{{r_e}^2}{2}\left(\frac{\epsilon'_1}{\epsilon'_0}\right)^2 \left(\frac{\epsilon'_1}{\epsilon'_0} + \frac{\epsilon'_0}{\epsilon'_1} - \sin^2\Theta'\right)\delta\left(\epsilon'_1 - \frac{\epsilon'_0}{1 + \frac{\epsilon'_0}{m_e c^2}(1-\cos\Theta')}\right) \tag{4.12}$$

Где r_e - классический радиус электрона, ϵ'_0 и ϵ'_1 - энергии начального и конечного фотона, соответственно, Θ' - угол между начальным и конечным фотоном, определяемый выражением $\cos\Theta' = \cos\theta'_0\cos\theta'_1 + \sin\theta'_0\sin\theta'_1\cos(\phi'_1 - \phi'_0)$. Штрихованные индексы относятся к системе отсчета электрона. При этом начальная и конечная энергии фотонов оказываются связаны соотношениями

$$\epsilon'_1 = \frac{\epsilon'_0}{1 + \frac{\epsilon'_0}{m_e c^2}(1-\cos\Theta')} \tag{4.13}$$

$$\epsilon'_0 = \frac{\epsilon'_1}{1 - \frac{\epsilon'_1}{m_e c^2}(1-\cos\Theta')} \tag{4.14}$$

Число фотонов, рассеявшихся в заданный телесный угол в единицу времени в промежуток энергии в системе покоя электрона равно

$$\frac{dN'}{dt' d\epsilon'_1 d\Omega'_1} = \int c\frac{d\sigma}{d\epsilon'_1 d\Omega'_1}\frac{dn'}{d\epsilon'_0 d\Omega'_0} d\Omega'_0 d\epsilon'_0 \tag{4.15}$$

Перепишем дельта-функцию через энергию начального фотона с помощью соотношения

$$\delta(f(x)) = \sum \frac{\delta(x - x_k)}{|f'(x_k)|} \tag{4.16}$$

где x_k - корни функции $f(x)$. Производная выражения внутри дельта-функции равна

$$\frac{d\epsilon_1'}{d\epsilon_0'} = \frac{1}{(1+\frac{\epsilon_0'}{m_e c^2}(1-\cos\Theta'))^2} \tag{4.17}$$

и она сократится с квадратом отношения энергий в формуле для сечения. Функцию распределения начальных фотонов выразим в лабораторной системе с помощью выражения 4.11.

$$\frac{dN'}{dt' d\epsilon_1' d\Omega_1'} = \int \frac{r_e^2 c}{2}\gamma_e^2(1-\mu_0\beta_e)^2(\frac{\epsilon_1'}{\epsilon_0'}+\frac{\epsilon_0'}{\epsilon_1'}-\sin^2\Theta')\frac{dn}{d\epsilon_0 d\Omega_0}\delta(\epsilon_0'-\frac{\epsilon_1'}{1-\frac{\epsilon_1'}{m_e c^2}(1-\cos\Theta')})d\epsilon_0' d\mu_0' d\phi_0' \tag{4.18}$$

Теперь избавимся от дельта-функции, проинтегрировав по ϵ_0'.

$$\frac{dN'}{dt' d\epsilon_1' d\Omega_1'} = \int \frac{r_e^2 c}{2}\gamma_e^2(1-\mu_0\beta_e)^2(1+\cos^2\Theta'+(\frac{\epsilon_1'}{m_e c^2})^2\frac{(1-\cos\Theta')^2}{1-\frac{\epsilon_1'}{m_e c^2}(1-\cos\Theta')})\frac{dn}{d\epsilon_0 d\Omega_0}d\mu_0' d\phi_0' \tag{4.19}$$

Осталось перевести поток рассеяных фотонов в лабораторную систему отсчета $\frac{dN}{dt d\epsilon_1 d\Omega_1} = \frac{dN'}{dt' d\epsilon_1' d\Omega_1'}\frac{dt'}{dt}\frac{d\epsilon_1'}{d\epsilon_1}\frac{d\Omega_1'}{d\Omega_1}$. Используя то, что $dt = \gamma_e dt'$, $\epsilon = \frac{1}{\gamma_e(1-\mu_1\beta_e)}\epsilon'$ и $\mu_1' = \frac{\mu_1-\beta_e}{1-\mu_1\beta_e}$ получим

$$\frac{dN}{dt d\epsilon_1 d\Omega_1} = \int \frac{r_e^2 c}{2}\frac{(1-\mu_0\beta_e)^2}{1-\mu_1\beta_e}(1+\cos^2\Theta'+(\frac{\epsilon_1'}{m_e c^2})^2\frac{(1-\cos\Theta')^2}{1-\frac{\epsilon_1'}{m_e c^2}(1-\cos\Theta')})\frac{dn}{d\epsilon_0 d\Omega_0}d\mu_0' d\phi_0' \tag{4.20}$$

При интегрировании нужно выразить углы в лабораторной системе отсчета μ_0, ϕ_0 через переменные интегрирования μ_0', ϕ_0'. Для расчета рассеяния на распределении электронов нужно проинтегрировать формулу 4.20 с функцией распределения электронов, нормированной на количество частиц. При этом надо учесть разные направления движения электронов и произвести повороты углов.

Так же может быть удобно интегрировать в переменных лабораторной системы расчета, тогда выражение для потока фотонов будет следующим

$$\frac{dN}{dt d\epsilon_1 d\Omega_1} = \int \frac{r_e^2 c}{2}\frac{1}{\gamma_e^2(1-\mu_1\beta_e)}(1+\cos^2\Theta'+(\frac{\epsilon_1'}{m_e c^2})^2\frac{(1-\cos\Theta')^2}{1-\frac{\epsilon_1'}{m_e c^2}(1-\cos\Theta')})\frac{dn}{d\epsilon_0 d\Omega_0}d\mu_0 d\phi_0 \tag{4.21}$$

При рассмотрении процессов, связанных с электронами высоких энергий $\gamma_e \approx 10^8$ относительные численные погрешности вычислений могут быть очень велики, так как β_e и $\mu_0, \mu_1, \cos\Theta'$ оказываются слишком близки к единице и стандартный тип double может не разрешать это отличие. Поэтому для численных вычислений оказывается полезным ввести следующие вспомогательные величины:

$$\delta_e = 1 - \beta_e \tag{4.22}$$

$$\text{versin}\ \theta = 1 - \cos\theta \tag{4.23}$$

Тогда выражения вида $1-\mu\beta_e$ в этих величинах перепишется как

$$1 - \mu\beta_e = \text{versin}\ \theta + \delta_e - \text{versin}\ \theta\ \delta_e \tag{4.24}$$

а выражение для угла между конечным и начальным фотоном как

$$1 - \cos\Theta' = \text{versin}\ \theta'_0 + \text{versin}\ \theta'_1 - \text{versin}\ \theta'_0 \text{versin}\ \theta'_1 - \sin\theta'_0 \sin\theta'_1 \cos(\phi'_1 - \phi'_0) \qquad (4.25)$$

С использованием данных выражений значительно повышается точность и максимальные доступные к рассмотрению энергии фотонов и электронов.

В случае изотропных функций распределения фотонов и релятивистских электронов можно произвести аналитическое интегрирование по угловым переменным [9, 10], и тогда для вычисления излучения достаточно лишь провести интегрирования по энергиям по формуле

$$\frac{dN}{dtd\epsilon_1 d\Omega_1} = \int \frac{2\pi r_e^2 m_e c^3}{\epsilon_0 \gamma_e^2} \frac{dn_{ph}}{d\epsilon_0} \frac{dn_e}{d\epsilon_e} (2q\ \ln(q) + 1 + q - 2q^2 + \frac{q^2(1-q)\Gamma^2}{2(1+q\Gamma)}) d\epsilon_0 d\epsilon_e \qquad (4.26)$$

где $\Gamma = 4\epsilon_0\gamma_e/m_e c^2$, $q = \epsilon_1/((\gamma_e m_e c^2 - \epsilon_1)\Gamma)$.

4.3 Синхротронное излучение

Процесс синхротронного излучения хороши известен и описан в классических работах. Но с точки зрения квантовой электродинамки, любому процессу излучения можно так же сопоставить процесс поглощения. Сечение процесса синхротронного самопоглощения описано в работе Гизеллини и Свенсона [19]. Спектральная плотность мощности излучения единицы объема вещества определеяется формулой

$$I(\nu) = \int_{E_{min}}^{E_{max}} dE \frac{\sqrt{3} e^3 n F(E) B \sin(\phi)}{m_e c^2} \frac{\nu}{\nu_c} \int_{\frac{\nu}{\nu_c}}^{\infty} K_{5/3}(x) dx, \qquad (4.27)$$

где ϕ это угол между вектором магнитного поля и лучом зрения, ν_c критическая частота, определяемая выражением $\nu_c = 3e^2 B \sin(\phi) E^2/4\pi {m_e}^3 c^5$, и $K_{5/3}$ - функция МакДональда. Коэффициент поглощения для фотонов, распростроняющихся вдоль луча зрения равен

$$k(\nu) = \int_{E_{min}}^{E_{max}} dE \frac{\sqrt{3} e^3}{8\pi m_e \nu^2} \frac{nB\sin(\phi)}{E^2} \frac{d}{dE} E^2 F(E) \frac{\nu}{\nu_c} \int_{\frac{\nu}{\nu_c}}^{\infty} K_{5/3}(x) dx. \qquad (4.28)$$

References

1. Mathis J. S., Mezger P. G., Panagia N. Interstellar radiation field and dust temperatures in the diffuse interstellar medium and in giant molecular clouds // Astron. Astrophys.. — 1983. — Vol. 128. — P. 212–229.

2. Sironi Lorenzo, Spitkovsky Anatoly. Particle Acceleration in Relativistic Magnetized Collisionless Pair Shocks: Dependence of Shock Acceleration on Magnetic Obliquity // ApJ. — 2009. — Vol. 698, no. 2. — P. 1523–1549.

3. Guo Xinyi, Sironi Lorenzo, Narayan Ramesh. Non-thermal Electron Acceleration in Low Mach Number Collisionless Shocks. I. Particle Energy Spectra and Acceleration Mechanism // ApJ. — 2014. — Vol. 794, no. 2. — P. 153.

4. Crumley P., Caprioli D., Markoff S., Spitkovsky A. Kinetic simulations of mildly relativistic shocks - I. Particle acceleration in high Mach number shocks // MNRAS. — 2019. — Vol. 485, no. 4. — P. 5105–5119.

5. Romansky V. I., Bykov A. M., Osipov S. M. Electron and ion acceleration by relativistic shocks: particle-in-cell simulations // Journal of Physics Conference Series. — Vol. 1038 of Journal of Physics Conference Series. — 2018. — P. 012022.

6. Ginzburg V. L. Theoretical physics and astrophysics. Additional chapters. — 1975.

7. Klein O., Nishina Y. The Scattering of Light by Free Electrons according to Dirac's New Relativistic Dynamics // Nature. — 1928. — Vol. 122, no. 3072. — P. 398–399.

8. Dubus G., Cerutti B., Henri G. The modulation of the gamma-ray emission from the binary LS 5039 // Astron. Astrophys.. — 2008. — Vol. 477, no. 3. — P. 691–700.

9. Jones Frank C. Calculated Spectrum of Inverse-Compton-Scattered Photons // Physical Review. — 1968. — Vol. 167, no. 5. — P. 1159–1169.

10. Bykov A. M., Chevalier R. A., Ellison D. C., Uvarov Yu. A. Nonthermal Emission from a Supernova Remnant in a Molecular Cloud // ApJ. — 2000. — Vol. 538, no. 1. — P. 203–216.

11. Coppejans D. L., Margutti R., Terreran G. et al. A Mildly Relativistic Outflow from the Energetic, Fast-rising Blue Optical Transient CSS161010 in a Dwarf Galaxy // ApJ Lett.. — 2020. — may. — Vol. 895, no. 1. — P. L23.

12. Kelner S. R., Aharonian F. A., Bugayov V. V. Energy spectra of gamma rays, electrons, and neutrinos produced at proton-proton interactions in the very high energy regime // Phys. Rev. D. — 2006. — Vol. 74, no. 3. — P. 034018.

13. Kafexhiu Ervin, Aharonian Felix, Taylor Andrew M., Vila Gabriela S. Parametrization of gamma-ray production cross sections for p p interactions in a broad proton energy range from the kinematic threshold to PeV energies // Phys. Rev. D. — 2014. — Vol. 90, no. 12. — P. 123014.

14. Bykov A. M., Kalyashova M. E. Modeling of GeV-TeV gamma-ray emission of Cygnus Cocoon // Advances in Space Research. — 2022. — Vol. 70, no. 9. — P. 2685–2695.

15. Ackermann M., Ajello M., Allafort A. et al. A Cocoon of Freshly Accelerated Cosmic Rays Detected by Fermi in the Cygnus Superbubble // Science. — 2011. — Vol. 334, no. 6059. — P. 1103.

16. Bartoli B., Bernardini P., Bi X. J. et al. Identification of the TeV Gamma-Ray Source ARGO J2031+4157 with the Cygnus Cocoon // ApJ. — 2014. — Vol. 790, no. 2. — P. 152.

17. Abeysekara A. U., Albert A., Alfaro R. et al. HAWC observations of the acceleration of very-high-energy cosmic rays in the Cygnus Cocoon // Nature Astronomy. — 2021. — Vol. 5. — P. 465–471.

18. Rybicki George B., Lightman Alan P. Radiative Processes in Astrophysics. — 1986.

19. Ghisellini Gabriele, Svensson Roland. The synchrotron and cyclo-synchrotron absorption cross-section // MNRAS. — 1991. — Vol. 252. — P. 313–318.

Литература

1. Mathis J. S., Mezger P. G., Panagia N. Interstellar radiation field and dust temperatures in the diffuse interstellar medium and in giant molecular clouds // Astron. Astrophys.. — 1983. — Vol. 128. — P. 212–229.

2. Sironi Lorenzo, Spitkovsky Anatoly. Particle Acceleration in Relativistic Magnetized Collisionless Pair Shocks: Dependence of Shock Acceleration on Magnetic Obliquity // ApJ. — 2009. — Vol. 698, no. 2. — P. 1523–1549.

3. Guo Xinyi, Sironi Lorenzo, Narayan Ramesh. Non-thermal Electron Acceleration in Low Mach Number Collisionless Shocks. I. Particle Energy Spectra and Acceleration Mechanism // ApJ. — 2014. — Vol. 794, no. 2. — P. 153.

4. Crumley P., Caprioli D., Markoff S., Spitkovsky A. Kinetic simulations of mildly relativistic shocks - I. Particle acceleration in high Mach number shocks // MNRAS. — 2019. — Vol. 485, no. 4. — P. 5105–5119.

5. Romansky V. I., Bykov A. M., Osipov S. M. Electron and ion acceleration by relativistic shocks: particle-in-cell simulations // Journal of Physics Conference Series. — Vol. 1038 of Journal of Physics Conference Series. — 2018. — P. 012022.

6. Ginzburg V. L. Theoretical physics and astrophysics. Additional chapters. — 1975.

7. Klein O., Nishina Y. The Scattering of Light by Free Electrons according to Dirac's New Relativistic Dynamics // Nature. — 1928. — Vol. 122, no. 3072. — P. 398–399.

8. Dubus G., Cerutti B., Henri G. The modulation of the gamma-ray emission from the binary LS 5039 // Astron. Astrophys.. — 2008. — Vol. 477, no. 3. — P. 691–700.

9. Jones Frank C. Calculated Spectrum of Inverse-Compton-Scattered Photons // Physical Review. — 1968. — Vol. 167, no. 5. — P. 1159–1169.

10. Bykov A. M., Chevalier R. A., Ellison D. C., Uvarov Yu. A. Nonthermal Emission from a Supernova Remnant in a Molecular Cloud // ApJ. — 2000. — Vol. 538, no. 1. — P. 203–216.

11. Coppejans D. L., Margutti R., Terreran G. et al. A Mildly Relativistic Outflow from the Energetic, Fast-rising Blue Optical Transient CSS161010 in a Dwarf Galaxy // ApJ Lett.. — 2020. — may. — Vol. 895, no. 1. — P. L23.

12. Kelner S. R., Aharonian F. A., Bugayov V. V. Energy spectra of gamma rays, electrons, and neutrinos produced at proton-proton interactions in the very high energy regime // Phys. Rev. D. — 2006. — Vol. 74, no. 3. — P. 034018.

13. Kafexhiu Ervin, Aharonian Felix, Taylor Andrew M., Vila Gabriela S. Parametrization of gamma-ray production cross sections for p p interactions in a broad proton energy range from the kinematic threshold to PeV energies // Phys. Rev. D. — 2014. — Vol. 90, no. 12. — P. 123014.

14. Bykov A. M., Kalyashova M. E. Modeling of GeV-TeV gamma-ray emission of Cygnus Cocoon // Advances in Space Research. — 2022. — Vol. 70, no. 9. — P. 2685–2695.

15. Ackermann M., Ajello M., Allafort A. et al. A Cocoon of Freshly Accelerated Cosmic Rays Detected by Fermi in the Cygnus Superbubble // Science. — 2011. — Vol. 334, no. 6059. — P. 1103.

16. Bartoli B., Bernardini P., Bi X. J. et al. Identification of the TeV Gamma-Ray Source ARGO J2031+4157 with the Cygnus Cocoon // ApJ. — 2014. — Vol. 790, no. 2. — P. 152.

17. Abeysekara A. U., Albert A., Alfaro R. et al. HAWC observations of the acceleration of very-high-energy cosmic rays in the Cygnus Cocoon // Nature Astronomy. — 2021. — Vol. 5. — P. 465–471.

18. Rybicki George B., Lightman Alan P. Radiative Processes in Astrophysics. — 1986.

19. Ghisellini Gabriele, Svensson Roland. The synchrotron and cyclo-synchrotron absorption cross-section // MNRAS. — 1991. — Vol. 252. — P. 313–318.