



ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ НАУКИ  
ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ ИМ. А.Ф. ИОФФЕ РОССИЙСКОЙ АКАДЕМИИ  
НАУК

# FAINA

Астрофизический код для моделирования наблюдаемых потоков от источников  
излучения

## Руководство пользователя

Санкт-Петербург — 2023

# Содержание

<b>Введение . . . . .</b>	<b>3</b>
Установка и запуск . . . . .	3
Пример расчета излучения . . . . .	3
<b>1 Источники излучения . . . . .</b>	<b>6</b>
1.1 Функции распределения частиц . . . . .	6
1.1.1 Распределения фотонов . . . . .	6
1.1.2 Распределения массивных частиц . . . . .	10
1.1.3 Считывание распределений из файла . . . . .	18
1.2 Источники излучения . . . . .	20
1.2.1 Источники излучения, не зависящие от времени . . . . .	20
1.2.2 Источники излучения, меняющиеся со временем . . . . .	28
<b>2 Расчет излучения . . . . .</b>	<b>30</b>
2.1 Синхротронное излучение . . . . .	31
2.2 Обратное комптоновское рассеяние . . . . .	32
2.3 Распад пионов . . . . .	36
2.4 Тормозное излучение . . . . .	40
<b>3 Оптимизация параметров . . . . .</b>	<b>45</b>
3.1 Вычислители целевой функции . . . . .	45
3.2 Оптимизаторы целевой функции . . . . .	47
<b>4 Формулы расчета излучения . . . . .</b>	<b>53</b>
4.1 Преобразование функции распределения фотонов . . . . .	53
4.1.1 Фотоны . . . . .	55
4.1.2 Массивные частицы . . . . .	56
4.2 Комптоновское рассеяние . . . . .	56
4.3 Синхротронное излучение . . . . .	59
4.4 Распад пионов . . . . .	60
4.5 Тормозное излучение . . . . .	62
<b>Литература . . . . .</b>	<b>64</b>

# Введение

FAINA - численный код, предназначенный для расчетов различных видов электромагнитного излучения от астрофизических источников. Код написан на языке C++ с использованием только стандартной библиотеки. В текущей версии кода реализованы следующие виды излучения: синхротронное излучение, излучение за счет обратного комптоновского рассеяния, излучение распада пионов в результате свободно-свободных столкновений протонов, а так же тормозное излучение. FAINA позволяет вычислять наблюдаемые потоки от источников с заданными параметрами, а так же вычислять параметры источников с помощью фитирования наблюдаемых данных расчетными.

## Установка и запуск

Текущая версия кода доступна на github <https://github.com/VadimRomansky/Faina>. Скачайте архив и разархивируйте его в директорию Faina.

## Windows

Для работы с кодом и его запуска в операционной системе Windows необходимо использовать Microsoft Visual Studio и открыть с помощью неё файл Faina.sin, содержащийся в корневой директории кода. Работоспособность проверялась на версии Visual Studio 2022.

## Linux

Для запуска FAINA в операционной системе Linux предусмотрены два варианта. Рекомендуется использовать среду разработки QtCreator и открыть с помощью неё проектный файл Faina.pro, содержащийся в корневой директории кода.

Так же возможна непосредственная компиляция и запуск из терминала, с помощью команд

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

## Пример расчета излучения

Рассмотрим простейший пример, приведенный в процедуре evaluateSimpleSynchrotron в файле examples.cpp. В данном примере рассматривается синхротронное излучение от

однородного источника в форме плоского диска, с заданной степенной функцией распределения излучающих электронов. Сначала зададим значения магнитного поля и концентрации электронов (в коде используются единицы СГС).

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

После этого нужно создать распределение электронов. Вычисление синхротрона реализовано только для изотропного распределения, поэтому создадим изотропное степенное распределение. Конструктор степенного распределения принимает следующие параметры: массу частиц, подставим константу - массу электрона, степенной индекс (он считается положительным и должен быть больше 1), энергию, с которой начинается спектр, в качестве нее выберем энергию покоя электронов, и концентрацию электронов.

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

Далее создадим источник излучения - однородный плоский диск, параметры его конструктора это распределение электронов, магнитное поле, синус угла наклона магнитного поля к лучу зрения, радиус, толщина и расстояние до него.

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

Последнее, что нам нужно - вычислитель потока излучения. Ему нужно указать рассматриваемый диапазон энергий электронов, в виде числа точек разбиения для интегрирования, минимальной и максимальной энергии. Так же есть параметр, отвечающий за учет синхротронного самопоглощения, по умолчанию его значение true.

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

Синхротронное приближение применимо только при частотах, намного больших циклотронной, поэтому вычислим её

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

Теперь осталось только вычислить само излучение. У класса RadiationEvaluator есть метод, вычисляющий поток излучения в заданном диапазоне энергий и записывающий его в файл. Нужно указать имя файла, источник излучения, минимальную и максимальную энергии фотонов, и желаемое количество точек в этом диапазоне. Вычисление потока и вывод происходит в единицах энергия фотонов - энергетическая плотность потока излучения Вт/эрг см<sup>2</sup> = см<sup>-2</sup>с<sup>-1</sup>. Если необходим вывод в других единицах, то запись в файл нужно переписать самостоятельно.

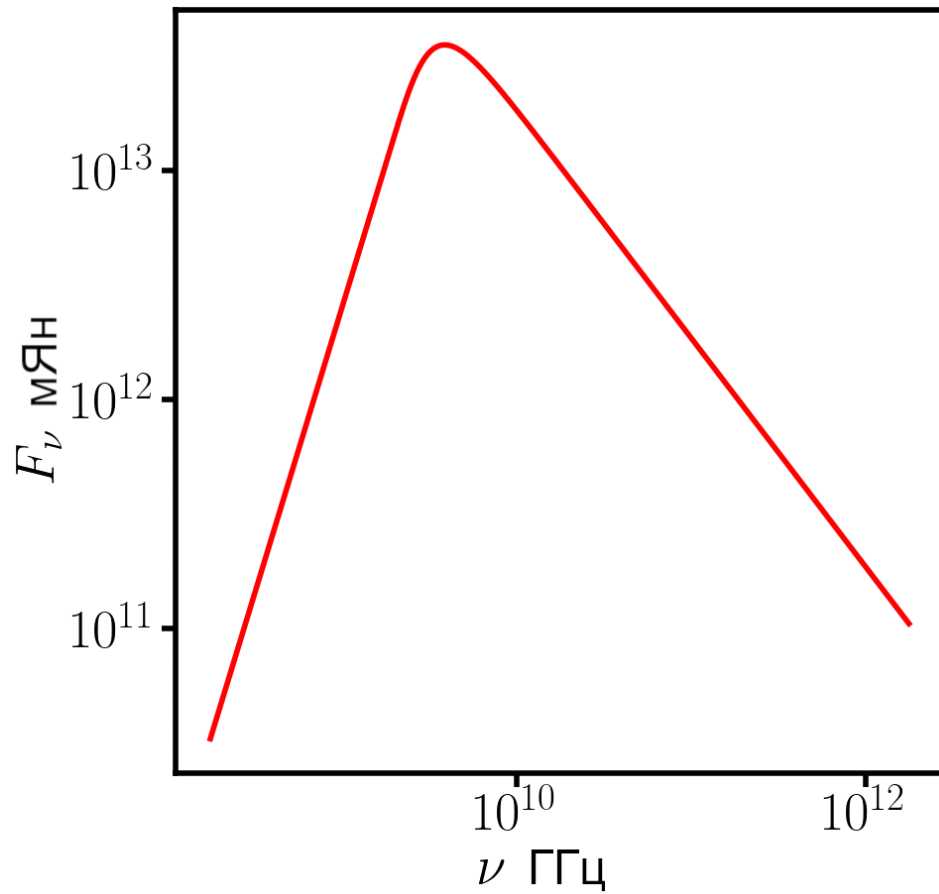


Рисунок 1: Энергетическая плотность потока синхротронного излучения от тестового источника

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

Функция вычисления синхротронного потока источника готова, осталось лишь вызвать её из основной процедуры `main()`. В результате вычисления должен получиться спектр источника, показанный на рисунке [1](#)

# Глава 1

## Источники излучения

FAINA позволяет рассчитывать электромагнитное излучение от источников с заданными функциями распределения излучающих частиц и другими параметрами.

### 1.1 Функции распределения частиц

Важнейшими исходными данными для расчета любого типа излучения является функция распределения излучающих частиц. В коде FAINA для представления распределений используется абстрактный класс `ParticleDistribution` и семейство наследованных от него классов, соответствующих различным конкретным реализациям. Класс `ParticleDistribution` имеет следующие доступные методы, описанные в Таблице 1.1:

Для вычисления излучения необходимо в первую очередь задать распределение излучающих частиц. Для это нужно создать объект из подходящего класса-наследника `ParticleDistribution`. Дерево наследования этого класс делится на две большие ветви - распределения фотонов, представленных абстрактным классом `PhotonDistribution` и распределения массивных частиц - `MassiveParticleDistribution`. Схема наследования этих классов представлена на рисунке 1.1.

Важно отметить, что распределения фотонов не используются для представления результатов расчета излучения. Они нужны как входной параметр для расчета обратного комптоновского рассеяния. Класс `PhotonDistribution` не имеет дополнительных собственных методов и является лишь интерфейсом. Класс `MassiveParticleDistribution` тоже является абстрактным, в нем не задан конкретный вид распределения, но добавлены новые методы, описанные в Таблице 1.2

#### 1.1.1 Распределения фотонов

От абстрактного класса `PhotonDistribution` наследуются следующие классы: абстрактный `PhotonIsotropicDistribution`, предназначенный для представления изотропных распределений фотонов, `PhotonPlankDirectedDistribution`, представляющий планковское распределение по энергии, но сконцентрированное в некотором телесном угле, такое распределение удобно для модели излучения звезды на некотором расстоянии от нее, и

Таблица 1.1: Публичные методы класса ParticleDistribution

<b>ParticleDistribution</b>	Абстрактный класс для любых распределений частиц
double distribution(const double& energy, const double& mu, const double& phi)	возвращает функцию распределения от энергии, косинуса полярного угла и азимутального угла, нормированную на концентрацию
virtual double distributionNormalized(const double& energy, const double& mu, const double& phi)	чисто виртуальный метод, возвращает функцию распределения от энергии, косинуса полярного угла и азимутального угла, нормированную на единицу
virtual double getMeanEnergy()	чисто виртуальный метод, возвращает значение средней энергии частиц в распределении
double getConcentration()	возвращает концентрацию частиц
void resetConcentration(const double& concentration)	изменяет концентрацию на новое значение

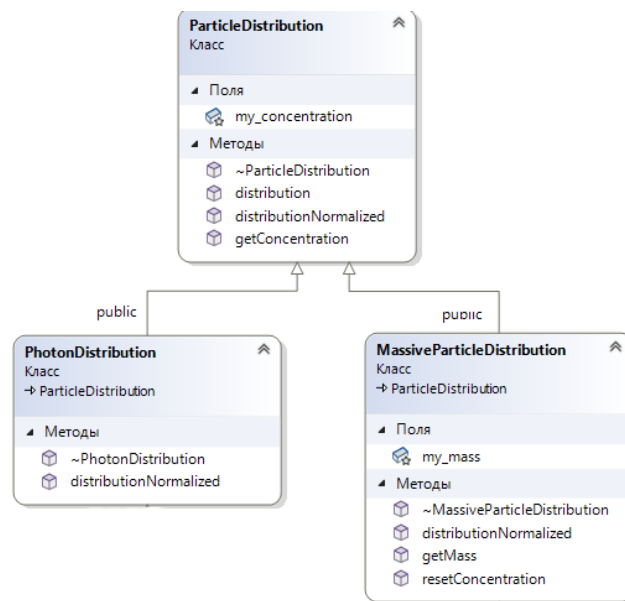


Рисунок 1.1: Схема наследования распределения фотонов и массивных частиц

Таблица 1.2: Публичные методы класса MassiveParticleDistribution

<b>MassiveParticleDistribution</b>	Абстрактный класс для распределений массивных излучающих частиц
virtual double minEnergy()	чисто виртуальный метод, возвращает нижнюю границу энергии частиц в распределении
virtual double maxEnergy()	чисто виртуальный метод, возвращает верхнюю границу энергии частиц в распределении. ОБРАТИТЕ ВНИМАНИЕ! если распределение не ограничено по энергии, возвращается отрицательное число.
double getMass()	возвращает массу частиц

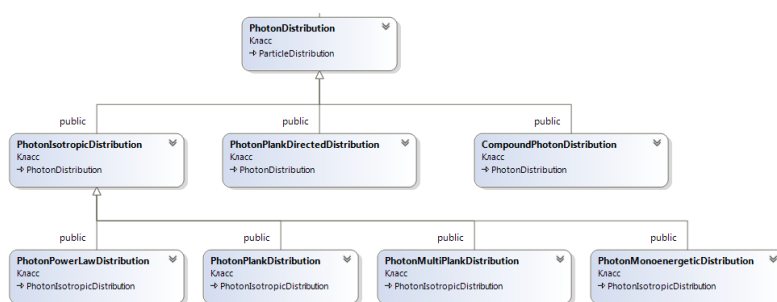


Рисунок 1.2: Схема наследования классов распределений фотонов

CompoundPhotonDistribution, представляющий из себя сумму нескольких распределений фотонов общего вида.

У изотропного распределения PhotonIsotropicDistribution добавляются методы, возвращающие значение функции распределения только в зависимости от энергии. Важно понимать, что это не функция распределения по энергии, а полная функция распределения с отброшенными угловыми аргументами. Другими словами, для получения значения функции распределения по энергии нужно домножить значение, возвращенное данным методом на  $4\pi$ . У класса PhotonIsotropicDistribution есть четыре наследника, которые являются не абстрактными классами, а непосредственно предназначены для создания распределений. Это PhotonPowerLawDistribution для представления степенных распределений, PhotonPlankDistribution, для планковских распределений, PhotonMultiPlankDistribution, для суммы планковских распределений, и PhotonMonoenergeticDistribution для моноэнергетического изотропного распределения. Схема наследования классов фотонных распределений представлена на рисунке 1.2.

Методы класса PhotonDistribution и его наследников перечислены в таблице 1.3



Таблица 1.3: Публичные методы классов распределений фотонов

<b>PhotonIsotropicDistribution</b>	Абстрактный класс для изотропных распределений фотонов
double distribution(const double& energy)	возвращает функцию распределения с отброшенными угловыми аргументами, то есть нормированную на концентрацию, деленную на $4\pi$
virtual double distributionNormalized(const double& energy)	чисто виртуальный метод, возвращает функцию распределения с отброшенными угловыми аргументами, нормированную на $1/4\pi$
void writeDistribution(const char* fileName, int Ne, const double& Emin, const double& Emax)	записывает распределение в файл в виде двух столбцов с точками распределенными логарифмически
<b>PhotonPowerLawDistribution</b>	Класс для степенного распределения фотонов
PhotonPowerLawDistribution(const double& index, const double& E0, const double& concentration)	конструктор, создающий экземпляр с заданными показателем наклона, начальной энергией и полной концентрацией
double getIndex()	возвращает показатель наклона спектра
double getE0()	возвращает минимальную энергию степенного распределения
<b>PhotonPlankDistribution</b>	Класс для планковского распределения фотонов
PhotonPlankDistribution(const double& temperature, const double& amplitude)	конструктор, создающий экземпляр с заданными температурой и амплитудой - то есть отношением концентрации к равновесному планковскому распределению с данной температурой
static PhotonPlankDistribution* getCMBRadiation()	статический метод, возвращающий экземпляр, соответствующий реликтовому излучению (температура $2.725K$ , амплитуда 1)
double getTemperature()	возвращает температуру распределения
<b>PhotonMultiPlankDistribution</b>	Класс для распределения фотонов, состоящего из суммы планковских распределений
PhotonMultiPlankDistribution(int Nplank, const double* const temperatures, const double* const amplitudes)	конструктор, принимающий количество планковских распределений, участвующих в смеси, массив их температур и массив амплитуд

static PhotonMultiPlankDistribution* getGalacticField()	статический метод, возвращающий экземпляр, соответствующий среднегалактическому фотонному распределению, по данным статьи [1]. Данное распределение состоит из пяти планковских компонент, с температурами $2.725K$ , $20K$ , $3000K$ , $4000K$ , $7000K$ и амплитудами $1.0$ , $4 \cdot 10^{-4}$ , $4 \cdot 10^{-13}$ , $1.65 \cdot 10^{-13}$ , $1.0 \cdot 10^{-14}$ соответственно
<b>PhotonMonoenergeticDistribution</b>	Класс для моноэнергетического изотропного распределения фотонов
PhotonMonoenergeticDistribution(const double& Energy, const double& halfWidth, const double& concentration)	конструктор, принимающий среднюю энергию распределения, полуширину разброса вокруг нее и концентрацию
<b>CompoundPhotonDistribution</b>	Класс для распределения фотонов, состоящего из суммы других распределений
CompoundPhotonDistribution(int N, PhotonDistribution** distributions)	конструктор, создающий экземпляр с заданным количеством распределений в смеси и массивом этих распределений
CompoundPhotonDistribution(PhotonDistribution* dist1, PhotonDistribution* dist2)	конструктор, создающий экземпляр содержащий смесь из двух распределений
CompoundPhotonDistribution(PhotonDistribution* dist1, PhotonDistribution* dist2, PhotonDistribution* dist3)	конструктор создающий экземпляр содержащий смесь из трех распределений
<b>PhotonPlankDirectedDistribution</b>	Класс для направленного планковского распределения фотонов
PhotonPlankDirectedDistribution(const double& temperature, const double& amplitude, const double& theta0, const double& phi0, const double& deltaTheta)	конструктор, принимающий температуру, амплитуду, углы задающие направление излучения и угол задающий полуширину раствора конуса излучения
double getTemperature()	возвращает температуру распределения

Пользователь может реализовать другие виды излучений самостоятельно, создав класс, наследующий от PhotonDistribution или PhotonIsotropicDistribution определив необходимый виртуальный метод distributionNormalized.

### 1.1.2 Распределения массивных частиц

Распределения массивных частиц представлены наследниками класса MassiveParticleDistribution. Так же как и в случае с фотонами важную роль играет абстрактный клас для представления изотропных распределений -

MassiveParticleIsotropicDistribution. У этого класса есть методы возвращающие значение функции распределения в зависимости от энергии, и опять же, это не функция распределения, проинтегрированная по углам, а полная функция распределения с отброшенными угловыми аргументами. Для получения значения функции распределения по энергии нужно домножить значение, возвращенное данным методом на  $4\pi$ .

Абстрактный класс изотропных распределений имеет семь наследников, предназначенных для создания конкретных распределений: MassiveParticlePowerLawDistribution - для степенных распределений, MassiveParticleBrokenPowerLawDistribution - для степенных распределений с изломом, MassiveParticlePowerLawCutoffDistribution - для степенных распределений с экспоненциальным завалом, MassiveParticleMaxwellDistribution - для максвелловского распределения (обратите внимание, что в отличие от остальных распределений, максвелловское подразумевает под энергией только кинетическую энергию), MassiveParticleMaxwellJuttnerDistribution - для релятивистского распределения Максвелла-Юттнера, MassiveParticleTabulatedIsotropicDistribution - для таблично заданных распределений и MassiveParticleMonoenergeticDistribution - для моноэнергичного изотропного распределения.

Так же имеется шесть реализаций анизотропных распределений: MassiveParticleTabulatedPolarDistribution - для таблично заданных распределений с зависимостью только от энергии и полярного угла, MassiveParticleAnisotropicDistribution - для таблично заданных распределений с зависимостью от всех переменных, MassiveParticleMonoenergeticDirectedDistribution - для моноэнергичного пучка частиц, с импульсами направленными в заданный телесный угол, MassiveParticleMovingDistribution - для перевода функций распределения в движущуюся систему отсчета, CompoundMassiveParticleDistribution - для суммы распределений общего вида, CompoundWeightedMassiveParticleDistribution - для взвешенной суммы распределений общего вида. В некоторых случаях оперировать весами распределений удобнее, чем непосредственно концентрациями. Полная схема наследования классов распределений массивных частиц представлена на рисунке 1.3.

Список публичных методов классов распределений массивных частиц приведен в Таблице 1.4. Пользователь может сам реализовывать необходимые ему виды распределений излучающих частиц, создав наследника класса MassiveParticleDistribution или MassiveParticleIsotropicDistribution и определив необходимые виртуальные методы.

Таблица 1.4: Публичные методы классов распределений массивных частиц

<b>MassiveParticleIsotropicDistribution</b>	Абстрактный класс для изотропных распределений
---	--

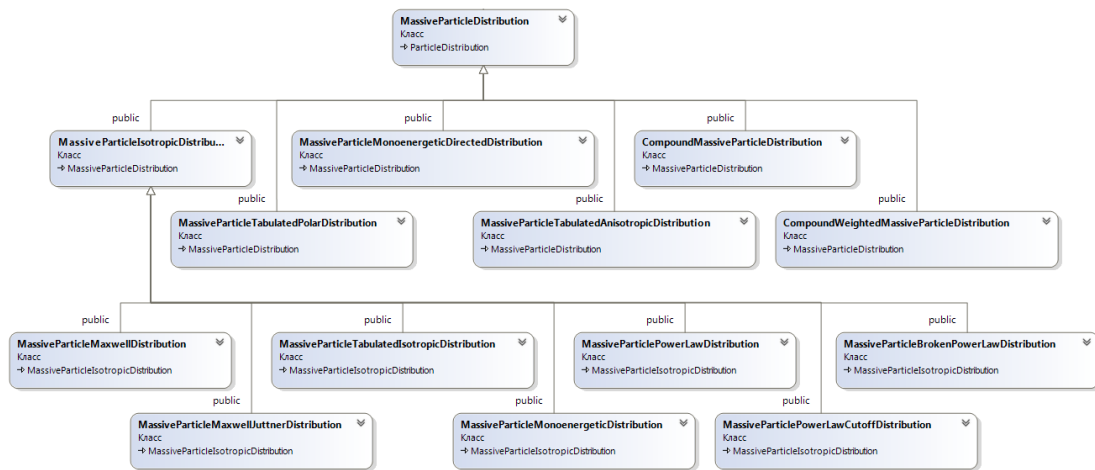


Рисунок 1.3: Схема наследования классов распределения массивных частиц

double distribution(const double& energy)	возвращает функцию распределения с отброшенными угловыми аргументами, то есть нормированную на концентрацию, деленную на $4\pi$
virtual double distributionNormalized(const double& energy)	чисто виртуальный метод, возвращает функцию распределения с отброшенными угловыми аргументами, нормированную на $1/4\pi$
void writeDistribution(const char* fileName, int Ne, const double& Emin, const double& Emax)	записывает распределение в файл с данным именем, в диапазоне между данными минимальной и максимальной энергиями с заданным количеством точек, которые распределяются логарифмически
double distributionNormalizedWithLosses( const double& energy, const double& lossRate, const double& time)	возвращает распределение с учетом эволюции во времени за счет синхротронных потерь до времени $t$ определяемое уравнением $f_t(E) = f\left(\frac{E}{1-El t}\right) \cdot \frac{1}{(1-El t)^2}$ and loss rate $l = 4e^4 B^2 / 9m^4 c^7$
<b>MassiveParticlePowerLawDistribution</b>	Класс для степенного распределения
MassiveParticlePowerLawDistribution( const double& mass, const double& index, const double& E0, const double& concentration)	конструктор, создает экземпляр степенного распределения частиц с заданными массой, степенным индексом, начальной энергией распределения и полной концентрацией

<code>double getIndex()</code>	возвращает степенной индекс распределения
<code>double getE0()</code>	возвращает начальную энергию распределения
<b>MassiveParticleBrokenPowerLawDistribution</b>	Класс для степенного распределения с изломом
<code>MassiveParticleBrokenPowerLawDistribution(const double&amp; mass, const double&amp; index1, const double&amp; index2, const double&amp; E0, const double&amp; Etran, const double&amp; concentration)</code>	конструктор, создает экземпляр степенного распределения с изломом частиц с заданными массой, степенными индексами на низких и высоких энергиях, начальной энергией распределения, энергией соответствующей излому и полной концентрацией
<code>double getIndex1()</code>	возвращает степенной индекс распределения на низких энергиях
<code>double getIndex2()</code>	возвращает степенной индекс распределения на высоких энергиях
<code>double getE0()</code>	возвращает начальную энергию распределения
<code>double getEtran()</code>	возвращает энергию излома
<b>MassiveParticlePowerLawCutoffDistribution</b>	Класс для степенного распределения с экспоненциальным завалом
<code>MassiveParticlePowerLawCutoffDistribution(const double&amp; mass, const double&amp; index, const double&amp; E0, const double&amp; beta, const double&amp; Ecut, const double&amp; concentration)</code>	конструктор, создает экземпляр степенного распределения с экспоненциальным завалом частиц с заданными массой, степенным индексом, начальной энергией распределения, параметром завала, энергией завала и полной концентрацией. $F(E) \propto (E/E_0)^{-index} \cdot \exp(-(E/E_{cut})^\beta)$
<code>double getIndex()</code>	возвращает степенной индекс распределения
<code>double getBeta()</code>	возвращает параметр завала распределения
<code>double getE0()</code>	возвращает начальную энергию распределения
<code>double getEcutoff()</code>	возвращает энергию экспоненциального завала
<b>MassiveParticleMaxwellDistribution</b>	Класс для распределения Максвелла
<code>MassiveParticleMaxwellDistribution(const double&amp; mass, const double&amp; temperature, const double&amp; concentration)</code>	конструктор, создает экземпляр распределения Максвелла частиц с заданными массой, температурой и полной концентрацией

double getTemperature()	возвращает температуру распределения
<b>MassiveParticleMaxwellJuttnerDistribution</b>	Класс для распределения Максвелла-Юттнера
MassiveParticleMaxwellJuttnerDistribution( const double& mass, const double& temperature, const double& concentration)	конструктор, создает экземпляр распределения Максвелла-Юттнера частиц с заданными массой, температурой и полной концентрацией
double getTemperature()	возвращает температуру распределения
<b>MassiveParticleTabulatedIsotropicDistribution</b>	Класс для таблично заданного изотропного распределения
MassiveParticleTabulatedIsotropicDistribution( const double& mass, const char* fileName, const int N, const double& concentration, DistributionInputType inputType)	конструктор, создает экземпляр табличного распределения частиц с заданными массой и полной концентрацией с помощью указанного файла, состоящего из двух колонок с данными указанной длины. Так же указывается формат входных данных.
MassiveParticleTabulatedIsotropicDistribution( const double& mass, const char* energyFileName, const char* distributionFileName, const int N, const double& concentration, DistributionInputType inputType)	конструктор, создает экземпляр табличного распределения частиц с заданными массой и полной концентрацией с помощью указанных двух файлов, состоящих из колонок с данными указанной длины. Так же указывается формат входных данных.
MassiveParticleTabulatedIsotropicDistribution( const double& mass, const double* energy, const double* distribution, const int N, const double& concentration, DistributionInputType inputType)	конструктор, создает экземпляр табличного распределения частиц с заданными массой и полной концентрацией с помощью двух переданных массивов данных указанной длины. Так же указывается формат входных данных.
int getN()	возвращает количество ячеек в таблице задающей функцию
double rescaleDistribution(const double& k)	масштабирует распределение, вытягивая его по оси энергии по формуле $E' = mc^2 + k \cdot (E - mc^2)$ , $F(E') = F(E)/k$ . Данная функция может быть полезна, например, в случае когда исходная функция распределения получена в результате работы численного кода с измененной массой электронов

<code>void addPowerLaw( const double&amp; Epower, const double&amp; index)</code>	добавляет к функции распределения степенной с указанным индексом, начиная с указанной энергии. Функция распределения при этом остается нормированной на указанную ранее концентрацию
<b>MassiveParticleMonoenergeticDistribution</b>	Класс для моноэнергичного изотропного распределения
<code>MassiveParticleMonoenergeticDistribution(const double&amp; mass, const double&amp; Energy, const double&amp; halfWidth, const double&amp; concentration)</code>	конструктор, принимающий массу, среднюю энергию, полуширину разброса по энергии и концентрацию
<b>MassiveParticleTabulatedPolarDistribution</b>	Класс для таблично заданного распределения с зависимостью от полярного угла
<code>MassiveParticleTabulatedPolarDistribution( const double&amp; mass, const char* energyFileName, const char* muFileName, const char* distributionFileName, const int Ne, const int Nmu, const double&amp; concentration, DistributionInputType inputType)</code>	конструктор, создает экземпляр табличного распределения частиц с заданными массой и полной концентрацией с помощью трех указанных файлов, в двух из которых содержатся сетки по энергии и косинусу полярного угла с указанными размерами, а в третьем двумерный массив функции распределения. Так же указывается формат входных данных.
<code>MassiveParticleTabulatedPolarDistribution( const double&amp; mass, const double* energy, const double* mu, const double** distribution, const int Ne, const int Nmu, const double&amp; concentration, DistributionInputType inputType)</code>	конструктор, создает экземпляр табличного распределения частиц с заданными массой и полной концентрацией с помощью трех переданных массивов данных, в двух из которых содержатся сетки по энергии и косинусу полярного угла с указанными размерами, а в третьем двумерный массив функции распределения. Так же указывается формат входных данных.

<code>int getNe()</code>	возвращает количество ячеек по энергии в таблице задающей функцию распределения
<code>int getNmu()</code>	возвращает количество ячеек по полярному углу в таблице задающей функцию распределения
<code>void double rescaleDistribution(const double&amp; k)</code>	масштабирует распределение, вытягивая его по оси энергии по формуле $E' = mc^2 + k \cdot (E - mc^2)$ , $F(E', \mu) = F(E, \mu)/k$ . Данная функция может быть полезна, например, в случае когда исходная функция распределения получена в результате работы численного кода с измененной массой электронов
<b>MassiveParticleTabulatedAnisotropicDistribution</b>	Класс для таблично заданного анизотропного распределения общего вида
<code>MassiveParticleTabulatedAnisotropicDistribution( const double&amp; mass, const char* energyFileName, const char* muFileName, const char* distributionFileName, const int Ne, const int Nmu, const int Nphi, const double&amp; concentration, DistributionInputType inputType)</code>	конструктор, создает экземпляр табличного распределения частиц с заданными массой и полной концентрацией с помощью трех указанных файлов, в двух из которых содержатся сетки по энергии и косинусу полярного угла с указанными размерами, а в третьем двумерный массив функции распределения. Сетка по азимутальному углу считается расномерной и определяется только размером. Так же указывается формат входных данных.



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)	конструктор, создает экземпляр табличного распределения частиц с заданными массой и полной концентрацией с помощью трех переданных массивов данных, в двух из которых содержатся сетки по энергии и косинусу полярного угла с указанными размерами, а в третьем двумерный массив функции распределения. Сетка по азимутальному углу считается равномерной и определяется только размером. Так же указывается формат входных данных.
int getNe()	возвращает количество ячеек по энергии в таблице задающей функцию распределения
int getNmu()	возвращает количество ячеек по полярному углу в таблице задающей функцию распределения
int getNphi()	возвращает количество ячеек по азимутальному углу в таблице задающей функцию распределения
void rescaleDistribution(const double& k)	масштабирует распределение, вытягивая его по оси энергии по формуле $E' = mc^2 + k \cdot (E - mc^2)$ , $F(E', \mu, \phi) = F(E, \mu, \phi)/k$ . Данная функция может быть полезна, например, в случае когда исходная функция распределения получена в результате работы численного кода с измененной массой электронов
<b>MassiveParticleMonoenergeticDirectedDistribution</b>	Класс для моноэнергичного направленного пучка частиц
MassiveParticleMonoenergeticDirectedDistribution(const double& mass, const double& Energy, const double& halfWidth, const double& concentration, const double& theta0, const double& phi0, const double& deltaTheta)	конструктор, принимающий массу частиц, среднюю энергию, полуширину разброса, концентрацию, углы задающие направление пучка и угол полуширины раствора конуса

<b>MassiveParticleMovingDistribution</b>	Класс осуществляющий перевод функций распределения в движущуюся систему отсчета
MassiveParticleMovingDistribution( MassiveParticleDistribution* distribution, const double& velocity)	конструктор, принимающий функцию распределения в собственной системе отсчета и скорость движения этой системы вдоль оси z относительно лабораторной системы
<b>CompoundMassiveParticleDistribution</b>	Класс для распределения, состоящего из суммы других распределений
CompoundMassiveParticleDistribution( int N, MassiveParticleDistribution** distributions)	конструктор, создает экземпляр класса содержащий смесь заданного количества указанных распределений
CompoundMassiveParticleDistribution( MassiveParticleDistribution* dist1, MassiveParticleDistribution* dist2)	конструктор, создает экземпляр класса, содержащий смесь двух распределений
CompoundMassiveParticleDistribution( MassiveParticleDistribution* dist1, MassiveParticleDistribution* dist2, MassiveParticleDistribution* dist3)	конструктор, создает экземпляр класса, содержащий смесь трех распределений
<b>CompoundWeightedMassiveParticleDistribution</b>	Класс для распределения, состоящего из взвешенной суммы других распределений
CompoundWeightedMassiveParticleDistribution( int N, const double* weights, MassiveParticleDistribution** distributions)	конструктор, создает экземпляр класса содержащий смесь заданного количества указанных распределений с заданными весами
CompoundWeightedMassiveParticleDistribution( MassiveParticleDistribution* dist1, const double& w1, MassiveParticleDistribution* dist2, const double& w2)	конструктор, создает экземпляр класса, содержащий смесь двух распределений с указанными весами
CompoundWeightedMassiveParticleDistribution( MassiveParticleDistribution* dist1, const double& w1, MassiveParticleDistribution* dist2, const double& w2, MassiveParticleDistribution* dist3, const double& w3)	конструктор, создает экземпляр класса, содержащий смесь трех распределений с указанными весами

### 1.1.3 Считывание распределений из файла

Классы таблично-заданных распределений, такие как например MassiveParticleTabulatedIsotropicDistribution, имеют конструктор принимающие на вход имена файлов, из которых будет считана функция распределения. Это должны быть

текстовые файлы, содержащие таблицы с данными, причем формат единиц, в которых измеряется функция распределения может быть разным. Для задания формата входных файлов используется перечислимый тип `DistributionInputType`, имеющий пять значений:

- `ENERGY_FE` - во входных файлах заданы энергия и функция распределения по энергии
- `ENERGY_KIN_FE` - заданы кинетическая энергия и функция распределения по энергии
- `GAMMA_FGAMMA` - задан лоренц-фактор и функция распределения по нему
- `GAMMA_KIN_FGAMMA` - задан лоренц-фактор, уменьшенный на единицу, и функция распределения по нему
- `MOMENTUM_FP` - задан импульс и функция распределения по импульсу

Вне зависимости от формата входного файла, функция распределения будет преобразована к единицам энергия - распределение по энергии. С помощью этих параметров можно считывать табличные распределения из файлов, например так:

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

Для облегчения создания распределений из файла в сложных случаях реализован класс `MassiveParticleDistributionFactory`. У него есть несколько методов, позволяющих считывать целые серии распределений из набора пронумерованных файлов. Что может быть полезно, если функция распределения зависит от некоторого параметра, как в примере вычисления синхротронного излучения описанном в следующей главе ???. Считать серию из десяти распределений электронов, содержащихся в файлах с именами `"Fe0.dat"` , `"Fe1.dat"` и так далее, состоящих из двух колонок - лоренц-фактор и функция распределения, и добавить к этим распределениям степенной хвост с показателем 3, начиная с энергий в 100 энергий покоя можно вызовом одной функции:

```
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);

```

Так же у пользователя есть возможность использовать конструкторы табличных распределений, принимающие не имена файлов, а непосредственно массивы со значениями функции распределения, которые пользователь может создать любым удобным ему способом.

## 1.2 Источники излучения

В коде FAINA есть возможность расчета излучения, используя на прямую функции распределения излучающих частиц, с указанием необходимых дополнительных параметров, таких как объем источника, расстояние до него, магнитное поле и других. Но более универсальным и рекомендованным способ является расчет с помощью создания модели источника излучения. При таком подходе возможно учесть геометрическое строение источника, его неоднородности и другие особенности.

Реализованы два базовых класса источников - независящие от времени, представленные абстрактным классом `RadiationSource`, и изменяющиеся со временем, представленные абстрактным классом `RadiationTimeDependentSource`. Эти два класса не связаны между собой через наследование, но объект первого класса содержится внутри объектов второго как приватное поле класса. Схема классов источников излучения представлена на рисунке 1.4.

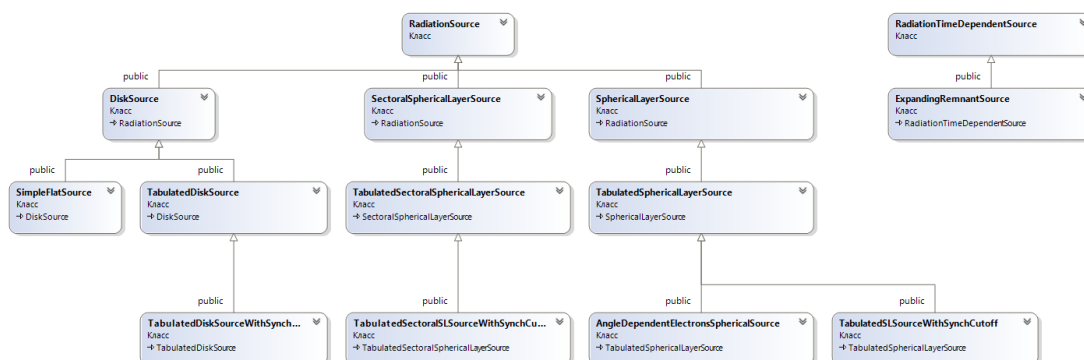


Рисунок 1.4: Схема наследования классов источников излучения

### 1.2.1 Источники излучения, не зависящие от времени

Источники излучения без временной зависимости реализованы с помощью абстрактного класса `RadiationSource`. Геометрически каждый источник задан в виде пространствен-

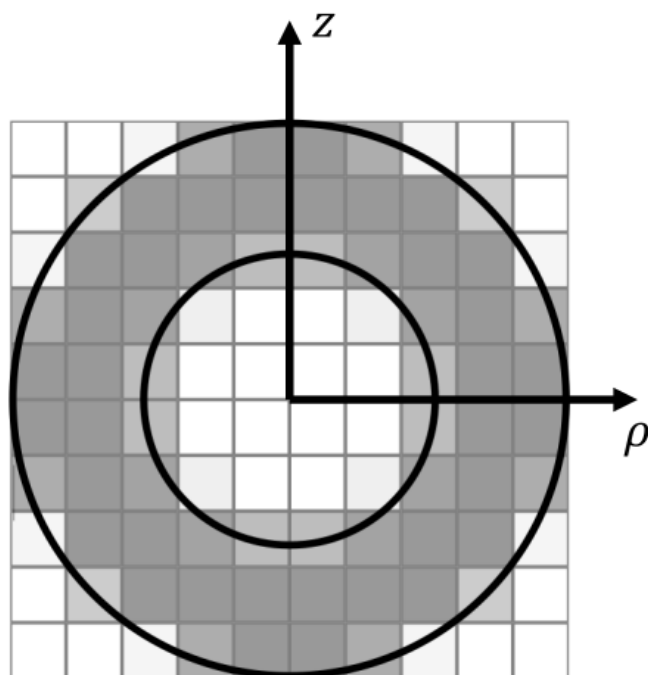


Рисунок 1.5: Модель источника в форме шарового слоя, помещенного в цилиндрическую пространственную координатную сетку. Цвет характеризует долю объема ячейки, заполненную веществом источника.

ной области в цилиндрических координатах, с осью  $z$  направленной вдоль луча зрения к наблюдателю, и характеризуется максимальным радиусом и минимальным и максимальным значением координаты  $z$ . Такая система координат выбрана для удобства учета процессов поглощения при прохождении излучения внутри самого источника вдоль луча зрения. Отличие реальной формы источника от цилиндрической реализовано с помощью долей заполнения веществом источника ячеек пространственной сетки. Модель источника, имеющего форму шарового слоя, в цилиндрической пространственной сетке изображена на рисунке 1.5. Цветом обозначена доля объема ячейки, заполненная веществом источника.

Так же источники излучения имеют следующие важные характеристики, которые могут меняться в различных пространственных ячейках источника: концентрация излучающих частиц, их функция распределения, магнитное поле и угол его наклона к лучу зрения. Большинство методов расчета излучения (все кроме обратного комптоновского рассеяния) реализованы только для изотропных распределений излучающих частиц, поэтому источники содержат только изотропные распределения. Так же у источника должно быть задано расстояние до наблюдателя.

Класс `RadiationSource` имеет три абстрактных класса-наследника: `DiskSource` - для источников в форме диска, перпендикулярного лучу зрения, и `SphericalLayerSource` - для

источников в форме шарового слоя и SectoralSphericalLayerSource - источник, который нужен тогда, когда рассматривается только сектор шарового слоя, "долька апельсина".

Источники в форме диска имеют три реализации: SimpleFlatSource - однородный диск, состоящий из одной пространственной ячейки с заданными параметрами, и TabulatedDiskSource - источник, в котором все характеристики таблично заданы на пространственной сетке и отнаследованный от него TabulatedDiskSourceWithSynchCutoff, который нужен для учета синхротронных потерь функции распределения. В модели данного источника считается, что распределение частиц генерируется на границе источника (верхней грани, соответствующей ударной волне), а в дальнейшем конвекционно переносятся вглубь него, испытывая при этом синхротронные потери. Изменение функции распределения в зависимости от расстояния до границы в случае однородного поля определяется формулой:

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

где  $f(E)$  исходная функция распределения,  $E$  - энергия частицы,  $B$  - магнитное поле,  $l$  - расстояние до границы,  $v$  - скорость конвекционного движения,  $e$  - заряд частицы,  $m$  - масса частицы,  $c$  - скорость света.

Источники в форме шарового слоя имеют следующие реализации: TabulatedSphericalSource - источник, в котором все характеристики таблично заданы на пространственной сетке, и отнаследованные от него TabulatedSLSourceWithSynchCutoff и AngleDependentElectronsSphericalSource. Первый из них нужен для учета синхротронных потерь, аналогично тому как это сделано в TabulatedDiskSourceWithSynchCutoff, а второй - для реализации важного случая, когда функция распределения излучающих частиц зависит от угла наклона магнитного поля по отношению к направлению распространения ударной волны [2, 3, 4, 5?]. В AngleDependentElectronsSphericalSource такие параметры, как концентрация, магнитное поле и его угол наклона к лучу зрения заданы таблично на пространственной сетке, а функция распределения излучающих частиц - в виде таблицы по углам наклона магнитного поля к направлению распространения ударной волны, которая в данном случае считается сферически симметричной. Функция распределения в каждой ячейке выбирается в зависимости от вычисленного угла наклона магнитного поля к ударной волне.

Источники в форме шарового слоя имеют следующие реализации: TabulatedSectoralSphericalLayerSource - источник, в котором все характеристики таблично заданы на пространственной сетке, и отнаследованный от него TabulatedSectoralSLSourceWithSynchCutoff, учитывающий потери энергии частиц аналогично тому, как это реализовано в классе TabulatedDiskSourceWithSynchCutoff.

Публичные методы классов источников излучения без зависимости от времени перечислены в Таблице 1.5.

Таблица 1.5: Публичные методы классов источников излучения без зависимости от времени

<b>RadiationSource</b>	абстрактный класс для источников излучения общего вида
virtual double getMaxRho()	чисто виртуальный метод, возвращает границу источника по радиальной оси в цилиндрических координатах
virtual double getMinZ()	чисто виртуальный метод, возвращает минимальную границу источника по оси z
virtual double getMaxZ()	чисто виртуальный метод, возвращает максимальную границу источника по оси z
virtual double getMaxB()	чисто виртуальный метод, возвращает максимальное магнитное поле
virtual double getAverageSigma()	чисто виртуальный метод, возвращает среднюю магнетизацию $\sigma = \frac{B^2}{4\pi m_p c^2}$
virtual double getAverageConcentration()	чисто виртуальный метод, возвращает среднюю концентрацию
virtual double getRho(int irho)	чисто виртуальный метод, возвращает радиальную координату данной ячейки
virtual double getZ(int iz)	чисто виртуальный метод, возвращает z координату данной ячейки
virtual double getPhi(int iphi)	чисто виртуальный метод, возвращает азимутальную координату данной ячейки
virtual int getRhoIndex(const double& rho)	чисто виртуальный метод, возвращает радиальный индекс ячейки по координате
virtual bool isSource(int irho, int iphi)	чисто виртуальный метод, возвращает логическое значение - учитывать ли ячейки с данными радиальными и азимутальными координатами при расчете излучения всего источника
int getNrho()	возвращает количество пространственных ячеек по радиальной оси цилиндрических координат
int getNz()	возвращает количество пространственных ячеек по оси z цилиндрических координат
int getNphi()	возвращает количество пространственных ячеек по азимутальному углу цилиндрических координат

<code>double getDistance()</code>	возвращает расстояние до источника
<code>getArea(int irho)</code>	возвращает поперечное сечение данной пространственной ячейки
<code>getVolume(int irho, int iz, int iphi)</code>	возвращает объем ячейки, занятый веществом источника. Этот метод согласован с методами <code>getArea</code> и <code>getLength</code> и возвращает их произведение
<code>virtual getB(int irho, int iz, int iphi)</code>	чисто виртуальный метод, возвращает значение магнитного поля в ячейке
<code>virtual getConcentration(int irho, int iz, int iphi)</code>	чисто виртуальный метод, возвращает значение концентрации в ячейке
<code>virtual getSinTheta(int irho, int iz, int iphi)</code>	чисто виртуальный метод, возвращает синус угла наклона магнитного поля к лучу зрения
<code>virtual void getVelocity(int irho, int iz, int iphi, double&amp; velocity, double&amp; theta, double&amp; phi)</code> чисто виртуальный метод, возвращает скорость данной ячейки источника	
<code>virtual getTotalVolume()</code>	чисто виртуальный метод, возвращает полный объем источника
<code>virtual getLength(int irho, int iz, int iphi)</code>	чисто виртуальный метод, возвращает среднюю толщину ячейки, заполненную веществом источника
<code>virtual resetParameters(const double* parameters, const double* normalizationUnits)</code>	чисто виртуальный метод, меняющий параметры источника. Список параметров, их количество, их влияние на источник определяются пользователем в конкретных реализациях класса. Принимет массив параметров и массив единиц в которых они измерены. Данный метод используется в процедурах оптимизации, либо при учете изменения источника со временем
<code>virtual getParticleDistribution(int irho, int iz, int iphi)</code>	чисто виртуальный метод, возвращает распределение излучающих частиц в ячейке
<b>DiskSource</b>	Абстрактный класс для источников в форме диска
<b>SimpleFlatSource</b>	Класс для источников в форме однородного диска
<code>SimpleFlatSource( MassiveParticleDistribution* electronDistribution, const double&amp; B, const double&amp; sinTheta, const double&amp; rho, const double&amp; z, const double&amp; distance, const double&amp; velocity = 0)</code>	конструктор, возвращает экземпляр с заданными распределением частиц, магнитным полем, синусом угла его наклона, радиусом диска, толщиной диска, расстоянием до источника и скоростью движения вещества



<b>TabulatedDiskSource</b>	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
<b>TabulatedDiskSourceWithSynchCutoff</b>	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

<b>SphericalLayerSource</b>	Абстрактный класс для источников в форме шарового слоя
double getInnerRho()	возвращает внутренний радиус шарового слоя
<b>TabulatedSphericalLayerSource</b>	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
<b>AngleDependentElectronsSphericalSource</b>	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
<b>TabulatedSLSourceWithSynchCutoff</b>	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
<b>SectoralSphericalLayerSource</b>	абстрактный класс для источников в форме сектора шарового слоя (дольки апельсина)
double getRhoIn()	возвращает внутренний радиус шарового слоя
<b>TabulatedSectoralSphericalLayerSource</b>	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 также заданными внутренним и внешним радиусом шарового слоя, углом раствора сектора, расстоянием до источника и скоростью движения вещества
<b>TabulatedSectoralSLSourceWithSynchCutoff</b>	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 также 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 также 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 абстрактным 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.

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

<b>RadiationTimeDependentSource</b>	Абстрактный класс для учета изменений источников излучения со временем
<code>virtual resetParameters(const double* parameters, const double* normalizationUnits)</code>	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
<code>virtual getRadiationSource(double&amp; time, const double* normalizationUnits)</code>	возвращает источник излучения с параметрами соответствующими заданному моменту времени. Так же принимает на вход массив единиц, в которых измеряются параметры этого источника.
<b>ExpandingRemnantSource</b>	класс, представляющий модель расширяющегося с постоянной скоростью остатка сверхновой, имеющего форму шарового слоя постоянной толщины с однородными концентрацией и магнитным полем
<code>ExpandingRemnantSource(const double&amp; R0, const double&amp; B0, const double&amp; concentration0, const double&amp; v, const double&amp; widthFraction, RadiationSource* source, const double&amp; t0)</code>	конструктор, создает экземпляр класса расширяющейся сферической оболочки с заданными в момент $t_0$ радиусом, магнитным полем, концентрацией, скоростью расширения, отношением толщины оболочки к радиусу и моделью источника. Для корректного учета изменения источника во времени важно, чтобы конкретная реализация метода <code>source-&gt;resetParameters</code> соответствовала той, что используется в методе <code>getRadiationSource</code> . В данном случае подходят все перечисленные выше реализации источников не зависящих от времени

## Глава 2

# Расчет излучения

В коде FAINA построены модели следующих типов излучения: синхротронного, обратного комптоновского рассеяния, пионного распада в результате свободно-свободного взаимодействия протонов и тормозного излучения.

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  $\text{см}^{-2}\text{с}^{-1}$ . 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 рассчитывается излучение подробно описаны в Главе 4.

Таблица 2.1: Публичные методы класса `RadiationEvaluator`

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

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 $\text{см}^{-2}\text{с}^{-1}$ 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 $\text{эрг}\text{см}^{-2}\text{с}^{-1}$ в диапазоне от минимальной до максимальной энергии, проинтегрированным 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 $\text{см}^{-2}\text{с}^{-1}$
<b>RadiationSumEvaluator</b>	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 в

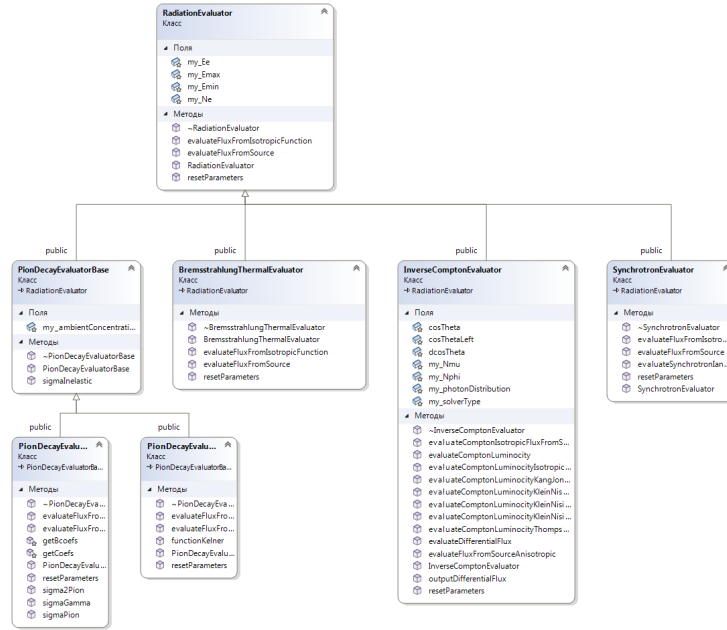


Рисунок 2.1: Схема наследования классов вычислителей излучения.

магнитном поле. Реализован случай только изотропной функции распределения излучающих частиц. Так же возможен учет синхротронного самопоглощения. Физическая сторона вопроса рассмотрена в разделе 4.3. 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 синхротронного 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, формулы для которых приведены в разделе 4.2, для обозначения которых используется перечислимый тип ComptonSolverType, имеющий следующие значения:



Таблица 2.2: Публичные методы класса SynchrotronEvaluator

<b>SynchrotronEvaluator</b>	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 доплеровского 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 томсоновском режиме. Реализовано только для степенного распределения электронов и теплового фотонов [6] глава 17, с. 466
- ANISOTROPIC\_KLEIN\_NISHINA - модель рассчитывающее излучение напрямую из сечения Клейна-Нишины, возможен учет анизотропных функций распределения [7, 8]
- ISOTROPIC\_KLEIN\_NISHINA - модель рассчитывающее 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.

Таблица 2.3: Публичные методы класса InverseComptonEvaluator

<b>InverseComptonEvaluator</b>	класс предназначенный для вычисления излучения рождающегося в результате обратного комптоновского рассеяния
--------------------------------	---

<code>InverseComptonEvaluator( int Ne, int Nmu, int Nphi, double Emin, double Emax, PhotonDistribution* photonDistribution, ComptonSolverType solverType)</code>	<p>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</p>
<code>evaluateComptonFluxKleinNishinaAnisotropic const double&amp; photonFinalEnergy, const double&amp; photonFinalTheta, const double&amp; photonFinalPhi, PhotonDistribution* photonDistribution, MassiveParticleDistribution* electronDistribution, const double&amp; volume, const double&amp; distance)</code>	<p>возвращает энергетическую плотность потока энергии в заданном направлении, излучением созданным заданными функциями распределения фотонов и рассеивающих частиц (которые могут быть анизотропными) в заданном объеме на данном расстоянии</p>
<code>evaluateFluxFromSourceAnisotropic( const double&amp; photonFinalEnergy, const double&amp; photonFinalTheta, const double&amp; photonFinalPhi, PhotonDistribution* photonDistribution, RadiationSource* source)</code>	<p>возвращает энергетическую плотность потока энергии в заданном направлении, излучением созданным заданными функциями распределения фотонов и источником, содержащим распределения рассеивающих частиц</p>
<b>InverseComptonEvaluatorWithSource</b>	<p>класс предназначенный для вычисления излучения рождающегося в результате обратного комптоновского рассеяния с учетом зависимости функции распределения фотонов от расстояния до их источника</p>
<code>InverseComptonEvaluatorWithSource(int Ne, int Nmu, int Nphi, double Emin, double Emax, double Ephmin, double Ephmax, PhotonDistribution* photonDistribution, ComptonSolverType solverType, const double&amp; sourceR, const double&amp; sourceZ, const double&amp; sourcePhi)</code>	<p>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</p>

Пример вычисления излучения от обратного комптоновского рассеяния содержится в процедуре `evaluateComptonWithPowerLawDistribution()` в файле `examples.cpp`. В ней рассчитывается 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 распределение 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<sup>-2</sup>с<sup>-1</sup>. 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, переводя 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 абстрактный класс `PionDecayEvaluatorBase` и два 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 [\[13\]](#). Формулы расчета излучения подробно описаны в разделе [4.4](#). 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.

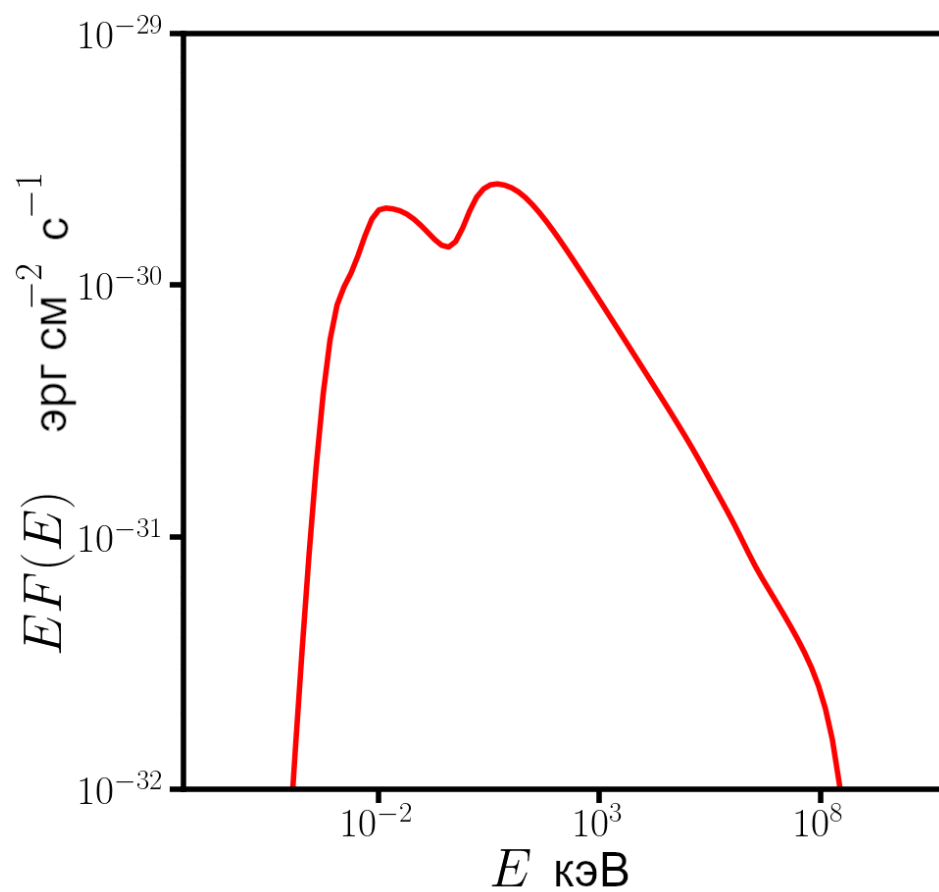


Рисунок 2.2: Энергетическая плотность потока синхротронного излучения от тестового источника

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

Таблица 2.4: Публичные методы класса `PionDecayEvaluatorBase` и его наследников

<b>PionDecayEvaluatorBase</b>	абстрактный класс для вычисления гамма излучения от распада пионов
<code>sigmaInelastic(const double&amp; energy)</code>	возвращает полное сечение неупругого взаимодействия протонов в лабораторной системе, принимает кинетическую энергию движущегося протона
<b>PionDecayEvaluatorKelner</b>	класс для вычисления гамма излучения от распада пионов по методу из статьи [12]
<code>PionDecayEvaluatorKelner(int Ne, double Emin, double Emax, const double&amp; ambientConcentration)</code>	конструктор, создает экземпляр с заданным рассматриваемым диапазоном энергии и концентрацией фоновых протонов
<b>PionDecayEvaluator</b>	класс для вычисления гамма излучения от распада пионов по методу из статьи [13]
<code>PionDecayEvaluator(int Ne, double Emin, double Emax, const double&amp; ambientConcentration)</code>	конструктор, создает экземпляр с заданным рассматриваемым диапазоном энергии и концентрацией фоновых протонов
<code>sigmaGamma(const double&amp; photonEnergy, const double&amp; protonEnergy)</code>	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 файл `examples.cpp`. В нем рассмотрено моделирование излучение объекта Кокон Лебеда 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 Лебеда - 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);

```

Создадим сетку по энергии для излученных гамма фотонов

```

int Nnu = 200;
double* E = new double[Nnu];
double* F = new double[Nnu];
double Ephmin = 0.01 * Emin;
double Ephmax = 1E16 * 1.6E-12;
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;
}

```

теперь вычислим спектральную плотность энергии и запишем ее в файл, переведя энергию в килоэлектрон-вольты, а поток в число фотонов на единицу энергии

```

for (int i = 0; i < Nnu; ++i) {
    F[i] = pionDecayEvaluator->evaluateFluxFromSource(
        E[i], source);
}
FILE* output_ev_dNdE = fopen("outputPionE.dat", "w");

```

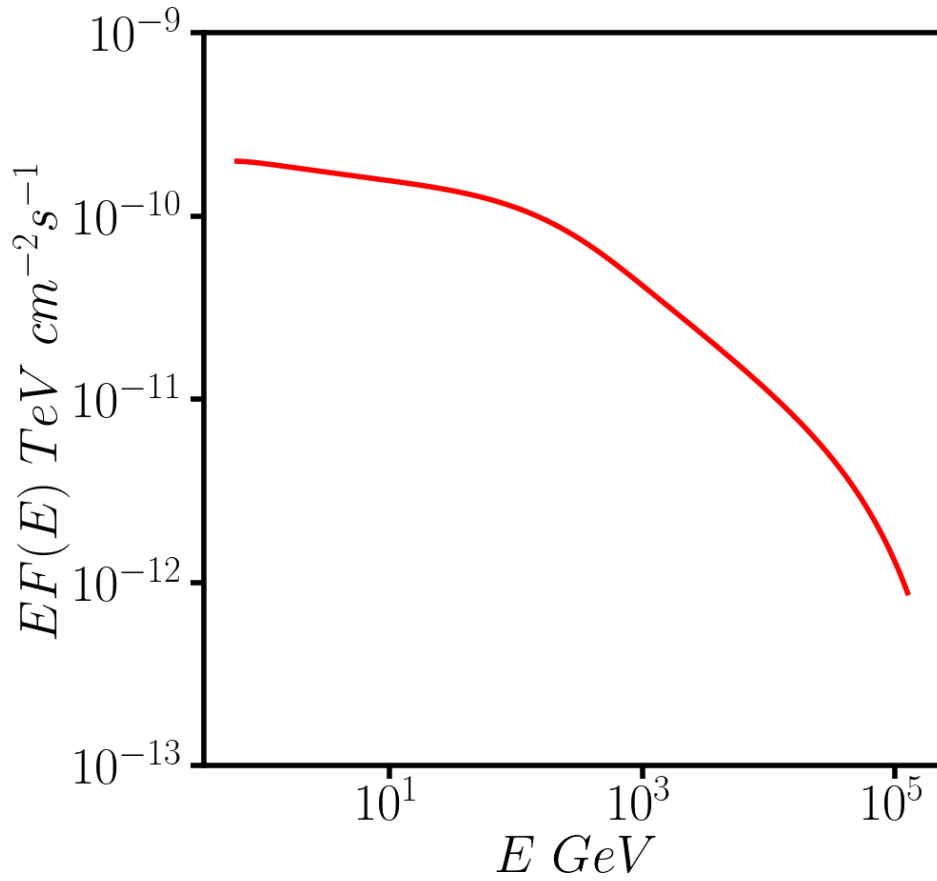


Рисунок 2.3: Расчетная энергетическая плотность потока гамма излучения Кокона Лебедя и данные наблюдений

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

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 Кокон Лебедя на Fermi LAT, ARGO и HAWC [15, 16, 17] приведены на рисунке 2.3

## 2.4 Тормозное излучение

Для вычисления тормозного излучения производимого в процессе электрон-электронных и электрон-протонных рассеяний, в коде реализованы два класса, наследованных от `RadiationEvaluator`. `BremsstrahlungThermalEvaluator` использует простую ана-



литическую модель, описанную в [18], применимую в случае тепловых распределений излучающих частиц. Этот класс следует использовать главным образом для отладки и калибровки следующего - BremsstrahlungEvaluator, который использует более общий подход и может применяться для любых изотропных распределений излучающих частиц, рассеивающихся на медленных фоновых частицах. Для электрон-ионного рассеяния используется формула Бёте-Хайтлера [19], детально описанная в [20]. Электрон-электронное рассеяние рассчитывается с помощью формул, описанных в работе [21]. Подробное описание формул расчета тормозного излучения приведено в разделе 4.5. Публичные методы классов вычислителей описаны в Таблице 2.5

Таблица 2.5: Публичные методы классов BremsstrahlungThermalEvaluator и BremsstrahlungEvaluator

<b>BremsstrahlungThermalEvaluator</b>	класс, предназначенный для вычисления тормозного излучения плазмы с тепловым распределением
BremsstrahlungThermalEvaluator()	конструктор по-умолчанию, не принимает никаких параметров
<b>BremsstrahlungEvaluator</b>	класс, предназначенный для вычисления тормозного излучения от произвольных изотропных распределений рассеивающихся электронов
BremsstrahlungEvaluator( int Ne, const double& Emin, const double& Emax)	конструктор, создает экземпляр класса с заданным диапазоном энергий рассматриваемых электронов и заданным числом точек для интегрирования в нем. При использовании этого конструктора концентрация ионов считается нулевой и учитываются только электрон-электронные столкновения
BremsstrahlungEvaluator( int Ne, const double& Emin, const double& Emax, double protonsRelativeConcentration)	конструктор, создает экземпляр класса с заданным диапазоном энергий рассматриваемых электронов, заданным числом точек для интегрирования в нем и указанной относительной концентрации протонов (при значении 1 она считается равной концентрации электронов).
BremsstrahlungEvaluator( int Ne, const double& Emin, const double& Emax, int ionNumber, double* ionConcentrations, int* ionCharges)	конструктор, создает экземпляр класса с заданным диапазоном энергий рассматриваемых электронов, заданным числом точек для интегрирования в нем, количеством типов ионов, их относительными концентрациями и зарядами в единицах элементарного заряда. ОБРАТИТЕ ВНИМАНИЕ! Электронейтральность системы не проверяется и является ответственностью пользователя.

double evaluateSigma( const double& gammaE, const double& epsilonG)	возвращает дифференциальное сечение излучения фотона с энергией $\epsilon_G$ в единицах $m_e c^2$ в диапазоне $d\epsilon_G$ электроном с Лоренц-фактором $\gamma_e$ , проинтегрированное по всем угловым переменным
--	---

Пример вычисления тормозного излучения приведен в функции evaluateBremsstrahlung в файле examples.cpp. В этом примере сравниваются результаты вычисления тормозного излучения тепловой плазмы, полученные двумя описанными выше классами. Как обычно, вычисление начинается с задания параметров источника. В этот раз к параметрам добавляется температура горячей релятивистской плазмы.

```
double electronConcentration = 1;
double rmax = 1.3E17;
double B = 0;
double temperature = 1E10;

double distance = 150 * 3.08 * 1.0E24;
double Emin = me_c2;
double Emax = me_c2 + 100 * kBoltzman * temperature;
```

Далее создадим распределение Максвелла-Юттнера излучающих электронов и источник излучения в виде однородного диска

```
MassiveParticleMaxwellJuttnerDistribution* electrons = new
MassiveParticleMaxwellJuttnerDistribution(
massElectron, temperature, electronConcentration);
RadiationSource* source = new
SimpleFlatSource(electrons, 0, 0, rmax, rmax, distance);
```

Так же создадим два вычислителя излучения. Концентрацию протонов укажем равной концентрации электронов.

```
int Ne = 500;
BremsstrahlungThermalEvaluator* bremsstrahlungEvaluator1 = new
BremsstrahlungThermalEvaluator();
BremsstrahlungEvaluator* bremsstrahlungEvaluator2 = new
BremsstrahlungEvaluator(Ne, Emin, Emax, 1.0);
```

Далее как обычно зададим сетку энергий излученных фотонов. Минимальное и максимальное значение зависят от температуры плазмы.

```
int Nnu = 200;
double* E = new double[Nnu];

double Ephmin = 0.001 * kBoltzman * temperature;
```

```

double Ephmax = 100 * kBoltzman * temperature;
double factor = pow(Ephmax / Ephmin, 1.0 / (Nnu - 1));
E[0] = Ephmin;
for (int i = 1; i < Nnu; ++i) {
    E[i] = E[i - 1] * factor;
}

```

Вычислим энергетическую плотность потока фотонов двумя разными вычислителями и запишем результаты в файл

```

FILE* output_ev_EFE = fopen("outputBremE.dat", "w");
FILE* output_GHz_Jansky = fopen("outputBremNu.dat", "w");
for (int i = 0; i < Nnu; ++i) {
    double nu = E[i] / hplank;
    printf("%d\n", i);
    printLog("%d\n", i);
    double F1 = bremsstrahlungEvaluator1->evaluateFluxFromSource(
E[i], source);
    double F2 = bremsstrahlungEvaluator2->evaluateFluxFromSource(
E[i], source);
    fprintf(output_ev_EFE, "%g_%g_%g\n",
E[i] / (1.6E-12), E[i] * F1, E[i] * F2);
    fprintf(output_GHz_Jansky, "%g_%g_%g\n",
nu / 1E9, 1E26 * hplank * F1, 1E26 * hplank * F2);
}
fclose(output_ev_EFE);
fclose(output_GHz_Jansky);

```

Результат расчета энергетической плотности потока энергии тормозного излучения показан на рисунке 2.4. В пределах своей применимости простая аналитическая модель дает результаты близкие к более сложной модели.

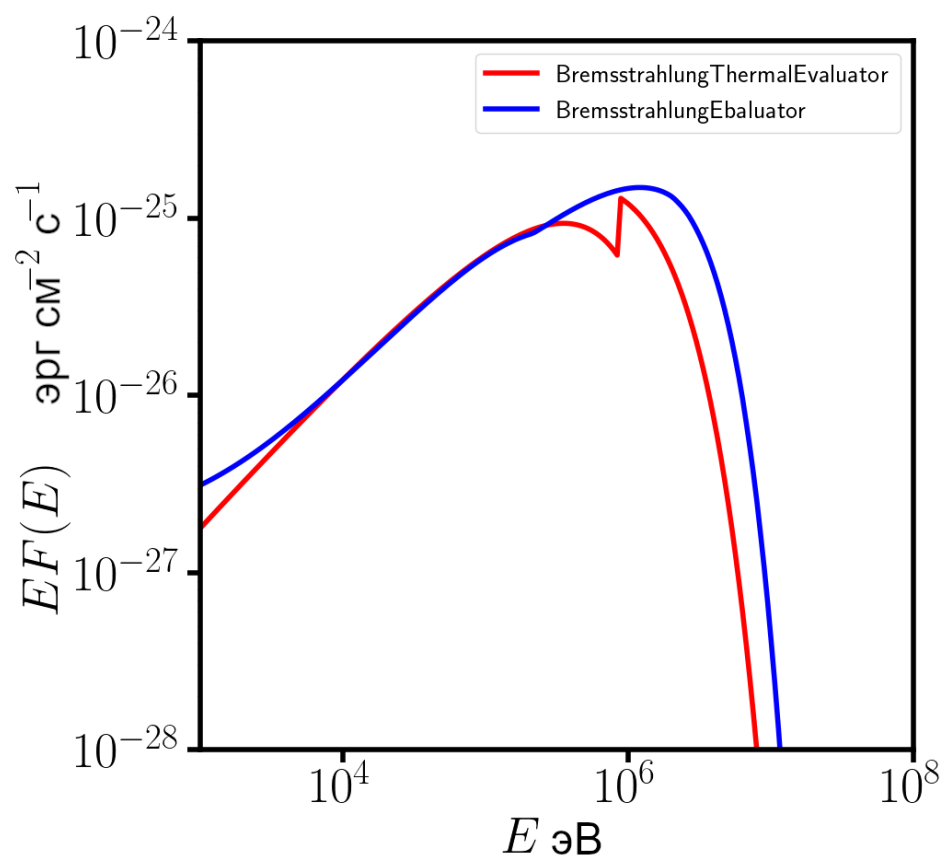


Рисунок 2.4: Расчетный спектр тормозного излучения горячей плазмы

## Глава 3

# Оптимизация параметров

Код FAINA позволяет не только рассчитывать 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 некоем фиксированном значении параметра  $x_i$  (например при данной частоте),  $F_{obs,i}$  - наблюдаемая величина этой функции,  $\sigma_i$  - 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.

Таблица 3.1: Публичные методы классов вычислителей целевых функций

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

<b>SpectrumLossEvaluator</b>	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, $\sigma_i$ - 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 <b>в каких единицах?</b>
<b>TimeDependentSpectrumLossEvaluator</b>	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, $\sigma_{ij}$ - 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 <b>в каких единицах?</b>
<b>RadialProfileLossEvaluator</b>	класс, в котором целевая функция характеризует отличие радиальной зависимости яркости источника в заданном диапазоне 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, $\sigma_i$ - 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 виртуальная 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 `RadiationOptimizer`: `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 наследования 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.

Таблица 3.2: Публичные методы классов оптимизаторов параметров источников

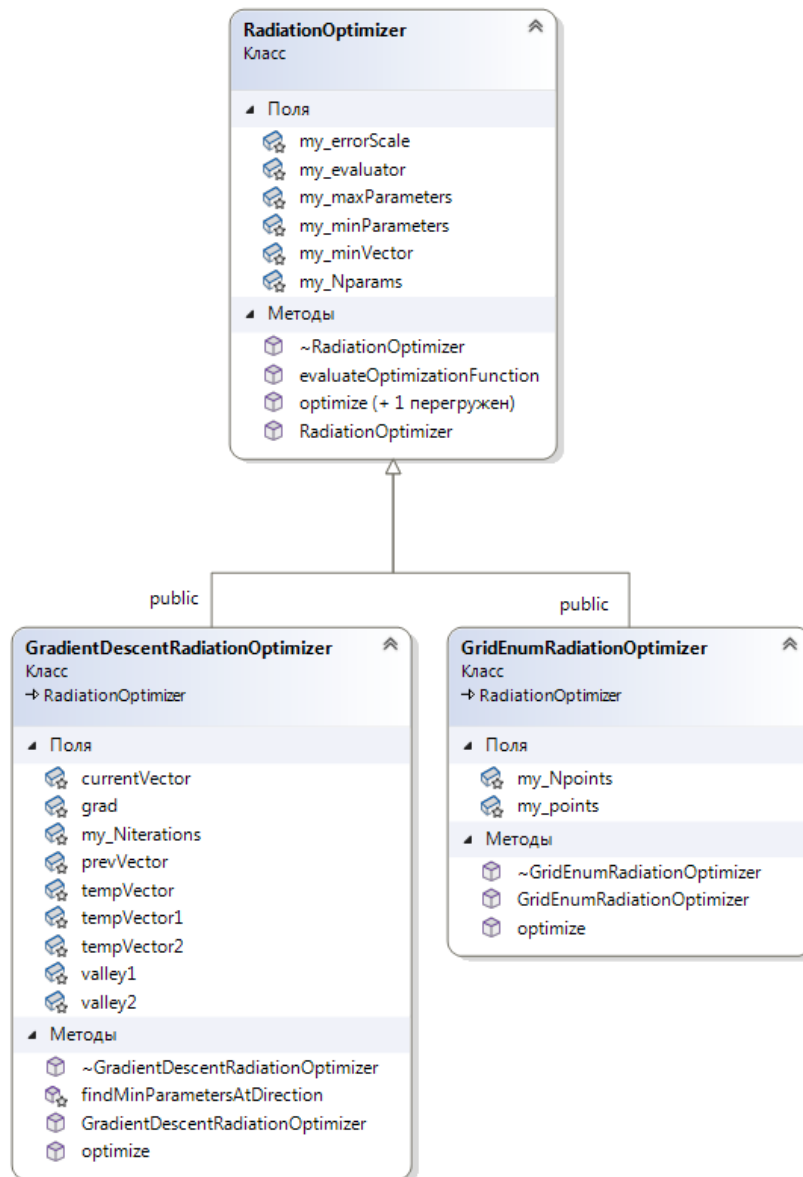


Рисунок 3.1: Схема наследования классов оптимизаторов



<b>RadiationOptimizer</b>	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 пробегает соответствующую плоскость, а остальные оптимизируются
<b>GridEnumRadiationOptimizer</b>	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.
<b>GradientDescentRadiationOptimizer</b>	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

<b>CombinedRadiationOptimizer</b>	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 `fitCSS161010withPowerLawDistribution` 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 данных 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  $\text{см}^{-2}\text{с}^{-1}$ ).

```
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 итерации 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, observedFlux,
    observedError);
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 параметры источника равны: радиус диска  $R = 1.8 \times 10^{17}$  см, магнитное поле  $B = 1.6$  Гс, концентрация электронов  $n = 2.3 \text{ см}^{-3}$ , 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.

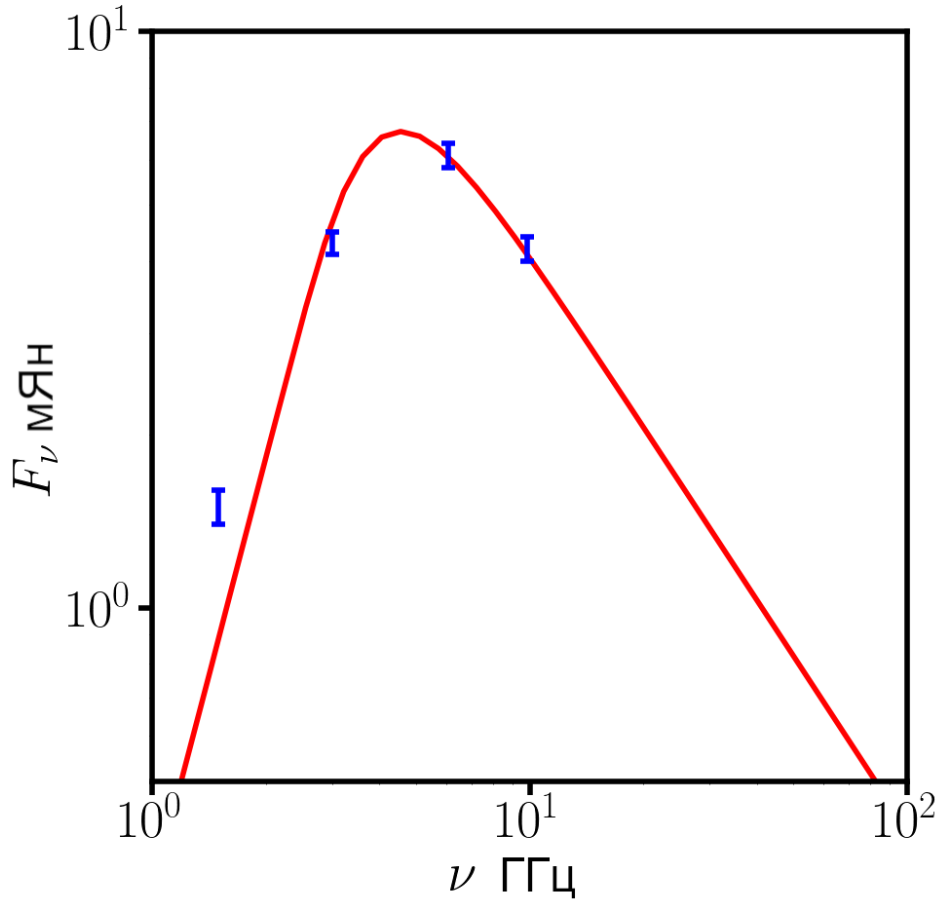


Рисунок 3.2: Наблюдаемый и расчетный спектр радиоизлучения объекта CSS161010 на 98 день после вспышки

## Глава 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.

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

Для вычисления функции распределения движущейся системе отсчета необходимо вычислить якобиан матрицы перехода в движущуюся систему отсчета. С учетом того, что азимутальный угол  $\phi' = \phi$  не изменяется при преобразованиях, и энергия и полярный угол не зависят от объема, занимаемого частицами, в общем виде матрица перехода выглядит как

$$J = \begin{pmatrix} \frac{d\epsilon'}{d\epsilon} & \frac{d\epsilon'}{d\mu} & 0 & 0 \\ \frac{d\mu'}{d\epsilon} & \frac{d\mu'}{d\mu} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{dV'}{d\epsilon} & \frac{dV'}{d\mu} & 0 & \frac{dV'}{dV} \end{pmatrix} \quad (4.2)$$

Определитель такой матрицы можно разложить по последнему столбцу и получить, что

$$|J| = \frac{dV'}{dV} \left( \frac{d\epsilon'}{d\epsilon} \frac{d\mu'}{d\mu} - \frac{d\epsilon'}{d\mu} \frac{d\mu'}{d\epsilon} \right) \quad (4.3)$$

Начнем с преобразования пространственного объема  $V$ . Один способ, используемый в п.10 т.2 Ландау-Лифшица [22], связан с переходом в собственную систему отсчета пучка движущихся частиц. Из этого можно получить, что объем преобразуется обратно энергии  $dV'/dV = \epsilon/\epsilon'$ . Этот результат правильный, но вывод не корректен для безмассовых частиц, у которых не существует собственной системы отсчета.

Поэтому мы вычислим изменение объема, содержащего рассматриваемые частицы. непосредственно из преобразований Лоренца. Рассмотрим поток частиц, равномерно распределенных вдоль оси  $z$ , с расстоянием  $L$  между соседними частицами, и движущихся

со одной скоростью  $v$ , направленной под углом  $\theta$  к оси  $z$  и введем величину  $\mu = \cos\theta$ , как показано на рисунке 4.1.

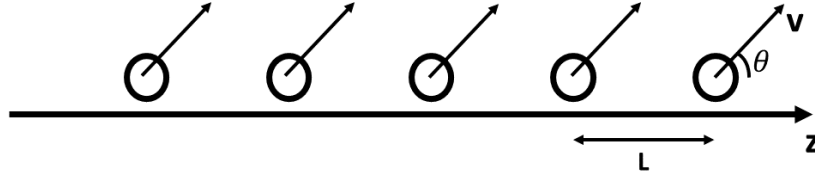


Рисунок 4.1: Пучок равномерно распределенных частиц, движущихся с одинаковыми скоростями

В лабораторной системе  $i$ -тая частица в момент времени  $t$  имеет  $z$ -координату  $z_i = i \cdot L + \mu vt$ . Вычислим ее координаты в движущейся системе отсчета

$$\begin{pmatrix} ct' \\ z'_i \end{pmatrix} = \begin{pmatrix} \gamma & -\beta\gamma \\ -\beta\gamma & \gamma \end{pmatrix} \times \begin{pmatrix} ct \\ z_i \end{pmatrix} \quad (4.4)$$

из этой системы можно получить, что значение координаты в движущейся системе равно  $z'_i = \gamma z_i + (\gamma\mu v - c\beta\gamma)t$ , но эти значения соответствуют одному моменту времени  $t$  в лабораторной системе отсчета, в то время как для определения объема или концентрации в движущейся системе, нам нужны координаты частиц в один момент времени относительно движущейся системы. Поэтому выразим  $t$  через  $z_i$  и  $t'$  и подставим в выражение для координат  $z'_i$

$$t = \frac{t' + \gamma\beta z_i/c}{\gamma - \beta\mu v/c} \quad (4.5)$$

и

$$z'_i = \gamma z_i + (\gamma\mu v - c\beta\gamma) \frac{t' + \gamma\beta z_i/c}{\gamma - \beta\mu v/c} = z_i \frac{1}{\gamma(1 - \beta\mu v/c)} + t' \frac{\mu v/c - \beta}{1 - \beta\mu v/c} \quad (4.6)$$

Второй член, содержащий  $t'$  дает стандартную формулу для релятивистского сложения скоростей. А первый характеризует сокращение расстояний между частицами  $L' = z'_{i+1} - z'_i = L/(\gamma(1 - \beta\mu v/c))$ . Расстояния вдоль поперечных направлений не изменяются, поэтому изменение объема будет таким же

$$V' = V/(\gamma(1 - \beta\mu v/c)) \quad (4.7)$$

это тот же результат, что и приводимый Ландау и Лифшицем [22]

Далее нужно найти выражения для  $\epsilon'$  и  $\mu'$ , но их лучше рассмотреть в двух отдельных случаях - для безмассовых частиц и для массивных.

### 4.1.1 Фотоны

Рассмотрим преобразование вектора четырех-импульса для безмассовых частиц (фотонов), учитывая что поперечные компоненты не изменяются, а временная и продольная связаны как  $p_z = \mu\epsilon$ :

$$\begin{pmatrix} \epsilon' \\ \mu'\epsilon' \end{pmatrix} = \begin{pmatrix} \gamma & -\beta\gamma \\ -\beta\gamma & \gamma \end{pmatrix} \times \begin{pmatrix} \epsilon \\ \mu\epsilon \end{pmatrix} \quad (4.8)$$

Из первой строчки матрицы получаем уравнение для доплеровского сдвига энергии

$$\epsilon' = \gamma(1 - \mu\beta)\epsilon \quad (4.9)$$

Вычислим производные новой энергии по старым координатам

$$\frac{d\epsilon'}{d\epsilon} = \gamma(1 - \mu\beta) \quad (4.10)$$

$$\frac{d\epsilon'}{d\mu} = -\gamma\beta\epsilon \quad (4.11)$$

Из второй строчки матрицы получаем  $\mu'\epsilon' = -\beta\gamma\epsilon + \gamma\mu\epsilon$ . Подставив значение  $\epsilon'$  из 4.9 и сократив  $\epsilon$  получим уравнение аберрации света

$$\mu' = \frac{\mu - \beta}{1 - \mu\beta} \quad (4.12)$$

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

$$\frac{d\mu'}{d\mu} = \frac{d}{d\mu} \frac{1}{\beta} \frac{\beta\mu - 1 + 1 - \beta^2}{1 - \mu\beta} = \frac{d}{d\mu} \frac{1}{\beta} \frac{1 - \beta^2}{1 - \mu\beta} = \frac{1 - \beta^2}{(1 - \mu\beta)^2} = \frac{1}{\gamma^2(1 - \mu\beta)^2} \quad (4.13)$$

Матрица якоби преобразования координат выглядит следующим образом

$$J = \begin{pmatrix} \frac{d\epsilon'}{d\epsilon} & \frac{d\epsilon'}{d\mu} & 0 & 0 \\ 0 & \frac{d\mu'}{d\mu} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \frac{dV'}{d\mu} & 0 & \frac{dV'}{dV} \end{pmatrix} \quad (4.14)$$

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

$$\frac{D(\epsilon', \mu', \phi', V')}{D(\epsilon, \mu, \phi, V)} = \frac{d\epsilon'}{d\epsilon} \frac{d\mu'}{d\mu} \frac{dV'}{dV} = \gamma(1 - \mu\beta) \frac{1}{\gamma^2(1 - \mu\beta)^2} \frac{1}{\gamma(1 - \mu\beta)} = \frac{1}{\gamma^2(1 - \mu\beta)^2} \quad (4.15)$$

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

$$n'_{ph}(\epsilon', \mu', \phi') = \frac{n_{ph}(\epsilon, \mu, \phi)}{\frac{D(\epsilon', \mu', \phi', V')}{D(\epsilon, \mu, \phi, V)}} = \gamma^2(1 - \mu\beta)^2 n_{ph}(\epsilon, \mu, \phi) \quad (4.16)$$

### 4.1.2 Массивные частицы

Для массивных частиц выражения для  $\epsilon'$  и  $\mu'$  намного сложнее. В этом случае  $p_z = \mu\sqrt{\epsilon^2 - m^2 c^4}/c$  где  $m$  масса частиц, и преобразования Лоренца выглядят следующим образом

$$\begin{pmatrix} \epsilon' \\ \mu' \sqrt{\epsilon'^2 - m^2 c^4} \end{pmatrix} = \begin{pmatrix} \gamma & -\beta\gamma \\ -\beta\gamma & \gamma \end{pmatrix} \times \begin{pmatrix} \epsilon \\ \mu \sqrt{\epsilon^2 - m^2 c^4} \end{pmatrix} \quad (4.17)$$

и выражения для  $\epsilon'$  и  $\mu'$

$$\epsilon' = \gamma\epsilon - \beta\gamma\mu\sqrt{\epsilon^2 - m^2 c^4} \quad (4.18)$$

$$\mu' = \frac{-\beta\gamma\epsilon + \gamma\mu\sqrt{\epsilon^2 - m^2 c^4}}{\sqrt{\epsilon^2 - m^2 c^4}} \quad (4.19)$$

Выражения для преобразования объемов переменных  $\epsilon$  и  $\mu$

$$V' = \frac{V}{\gamma(1 - \mu\beta\sqrt{\epsilon^2 - m^2 c^4}/\epsilon)} \quad (4.20)$$

Выражения для частных производных  $\epsilon'$ ,  $\mu'$ ,  $V'$  и особенно для якобиана выглядят устрашающе, поэтому здесь мы приведем только финальный результат преобразования функции распределения в единицах  $c = 1$

$$\frac{n'_m(\epsilon', \mu', \phi')}{n_m(\epsilon, \mu, \phi)} = \frac{\gamma(\epsilon - \mu\sqrt{\epsilon^2 - m^2}\beta)(\gamma^2\epsilon^2 - m^2 + \mu^2((\epsilon^2 - m^2)(\gamma^2 - 1)) - 2\mu\epsilon\gamma^2\beta\sqrt{\epsilon^2 - m^2})^{3/2}}{\epsilon(((\gamma^2 - 1)(\epsilon^2 - m^2)\mu^2 + \gamma^2\epsilon^2 - m^2)\sqrt{\epsilon^2 - m^2} - 2\mu\epsilon\gamma(\epsilon^2 - m^2)\sqrt{\gamma^2 - 1})} \quad (4.21)$$

## 4.2 Комптоновское рассеяние

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

$$\frac{d\sigma}{d\epsilon'_1 d\Omega'_1} = \frac{r_e^2}{2} \left( \frac{\epsilon'_1}{\epsilon'_0} \right)^2 \left( \frac{\epsilon'_1}{\epsilon'_0} + \frac{\epsilon'_0}{\epsilon'_1} - \sin^2 \Theta' \right) \delta \left( \epsilon'_1 - \frac{\epsilon'_0}{1 + \frac{\epsilon'_0}{m_e c^2} (1 - \cos \Theta')} \right) \quad (4.22)$$

Где  $r_e$  - классический радиус электрона,  $\epsilon'_0$  и  $\epsilon'_1$  - энергии начального и конечного фотона, соответственно,  $\Theta'$  - 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

$$\epsilon'_1 = \frac{\epsilon'_0}{1 + \frac{\epsilon'_0}{m_e c^2} (1 - \cos \Theta')} \quad (4.23)$$



$$\epsilon'_0 = \frac{\epsilon'_1}{1 - \frac{\epsilon'_1}{m_e c^2}(1 - \cos \Theta')} \quad (4.24)$$

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

$$\frac{dN'}{dt' d\epsilon'_1 d\Omega'_1} = \int c \frac{d\sigma}{d\epsilon'_1 d\Omega'_1} \frac{dn'}{d\epsilon'_0 d\Omega'_0} d\Omega'_0 d\epsilon'_0 \quad (4.25)$$

Перепишем дельта-функцию через энергию начального фотона с помощью соотношения

$$\delta(f(x)) = \sum \frac{\delta(x - x_k)}{|f'(x_k)|} \quad (4.26)$$

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

$$\frac{d\epsilon'_1}{d\epsilon'_0} = \frac{1}{(1 + \frac{\epsilon'_0}{m_e c^2}(1 - \cos \Theta'))^2} \quad (4.27)$$

и она сократится с квадратом отношения энергий в формуле для сечения. Функцию распределения начальных фотонов выразим в лабораторной системе с помощью выражения 4.16.

$$\frac{dN'}{dt' d\epsilon'_1 d\Omega'_1} = \int \frac{r_e^2 c}{2} \gamma_e^2 (1 - \mu_0 \beta_e)^2 \left( \frac{\epsilon'_1}{\epsilon'_0} + \frac{\epsilon'_0}{\epsilon'_1} - \sin^2 \Theta' \right) \frac{dn}{d\epsilon_0 d\Omega_0} \delta \left( \epsilon'_0 - \frac{\epsilon'_1}{1 - \frac{\epsilon'_1}{m_e c^2}(1 - \cos \Theta')} \right) d\epsilon'_0 d\mu'_0 d\phi'_0 \quad (4.28)$$

Теперь избавимся от дельта-функции, проинтегрировав по  $\epsilon'_0$ .

$$\frac{dN'}{dt' d\epsilon'_1 d\Omega'_1} = \int \frac{r_e^2 c}{2} \gamma_e^2 (1 - \mu_0 \beta_e)^2 \left( 1 + \cos^2 \Theta' + \left( \frac{\epsilon'_1}{m_e c^2} \right)^2 \frac{(1 - \cos \Theta')^2}{1 - \frac{\epsilon'_1}{m_e c^2}(1 - \cos \Theta')} \right) \frac{dn}{d\epsilon_0 d\Omega_0} d\mu'_0 d\phi'_0 \quad (4.29)$$

Осталось перевести количество рассеянных 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

$$\frac{dN}{dt d\epsilon_1 d\Omega_1} = \int \frac{r_e^2 c}{2} \frac{(1 - \mu_0 \beta_e)^2}{1 - \mu_1 \beta_e} \left( 1 + \cos^2 \Theta' + \left( \frac{\epsilon'_1}{m_e c^2} \right)^2 \frac{(1 - \cos \Theta')^2}{1 - \frac{\epsilon'_1}{m_e c^2}(1 - \cos \Theta')} \right) \frac{dn}{d\epsilon_0 d\Omega_0} d\mu'_0 d\phi'_0 \quad (4.30)$$

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

$$\frac{dN}{dt d\epsilon_1 d\Omega_1} = \int \frac{r_e^2 c}{2} \frac{1}{\gamma_e^2 (1 - \mu_1 \beta_e)} \left( 1 + \cos^2 \Theta' + \left( \frac{\epsilon'_1}{m_e c^2} \right)^2 \frac{(1 - \cos \Theta')^2}{1 - \frac{\epsilon'_1}{m_e c^2}(1 - \cos \Theta')} \right) \frac{dn}{d\epsilon_0 d\Omega_0} d\mu_0 d\phi_0 \quad (4.31)$$

При интегрировании нужно выразить углы в лабораторной системе отсчета  $\mu_0, \phi_0$  через переменные интегрирования  $\mu'_0, \phi'_0$ . Для расчета рассеяния на распределении электронов нужно проинтегрировать формулу 4.30 или 4.31 с функцией распределения электронов, нормированной на концентрацию частиц. При этом надо учесть разные направления движения электронов и произвести повороты углов.

В коде реализовано вычисления потоков в терминах энергетической плотности потока энергии в единицах  $\text{см}^{-2}\text{с}^{-1}$ . Для вычисления этой величины нужно домножить число рассеянных фотонов на энергию, поделить на квадрат расстояния до источника и проинтегрировать по объему источника.

$$F(\epsilon_1) = \frac{\epsilon_1}{D^2} \int \frac{dN}{dt d\epsilon_1 d\Omega_1} \frac{dn_e}{d\epsilon_e d\Omega_e} dV d\epsilon_e d\Omega_e \quad (4.32)$$

При рассмотрении процессов, связанных с электронами высоких энергий  $\gamma_e \approx 10^8$  относительные численные погрешности вычислений могут быть очень велики, так как  $\beta_e$  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:

$$\delta_e = 1 - \beta_e \quad (4.33)$$

$$\text{versin } \theta = 1 - \cos \theta \quad (4.34)$$

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

$$1 - \mu\beta_e = \text{versin } \theta + \delta_e - \text{versin } \theta \delta_e \quad (4.35)$$

а выражение для угла между конечным и начальным фотоном как

$$1 - \cos \Theta' = \text{versin } \theta'_0 + \text{versin } \theta'_1 - \text{versin } \theta'_0 \text{versin } \theta'_1 - \sin \theta'_0 \sin \theta'_1 \cos(\phi'_1 - \phi'_0) \quad (4.36)$$

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

$$F(\epsilon_1) = \frac{\epsilon_1}{D^2} \int \frac{2\pi r_e^2 \beta_e c}{\epsilon_0 \gamma_e^2} \frac{dn_{ph}}{d\epsilon_0} \frac{dn_e}{d\epsilon_e} (2q \ln(q) + 1 + q - 2q^2 + \frac{q^2(1-q)\Gamma^2}{2(1+q\Gamma)}) d\epsilon_0 d\epsilon_e dV \quad (4.37)$$

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 хорошо известен и описан в классических работах, например [6]. Но так же возможен и процесс синхротронного поглощения. Сечение этого процесса описано в работе Гизеллини и Свенсона [23]. В коде вычисляются спектральная плотность мощности излучения единицы объема и линейный коэффициент поглощения, описанные в работе [24]. Спектральная плотность мощности излучения единицы объема вещества определяется формулой

$$I(\nu) = \int_{E_{min}}^{E_{max}} dE \frac{\sqrt{3}e^3 n F(E) B \sin(\phi)}{m_e c^2} \frac{\nu}{\nu_c} \int_{\frac{\nu}{\nu_c}}^{\infty} K_{5/3}(x) dx, \quad (4.38)$$

где  $\phi$  это угол между вектором магнитного поля и лучом зрения,  $\nu_c$  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, распространяющихся вдоль луча зрения равен

$$k(\nu) = \int_{E_{min}}^{E_{max}} dE \frac{\sqrt{3}e^3}{8\pi m_e \nu^2} \frac{n B \sin(\phi)}{E^2} \frac{d}{dE} E^2 F(E) \frac{\nu}{\nu_c} \int_{\frac{\nu}{\nu_c}}^{\infty} K_{5/3}(x) dx. \quad (4.39)$$

Чтобы вычислить наблюдаемый поток в случае пренебрежимо малого самопоглощения, необходимо проинтегрировать мощность излучения 4.38 по объему источника и поделить результат на квадрат расстояния до источника  $D$ . Если самопоглощение значительно, нужно сначала решить дифференциальное уравнение для потока излучения  $\Phi = \frac{dW}{dt d\nu dS}$  вдоль луча зрения,

$$\frac{d\Phi(\nu, z)}{dz} = \frac{dW(\nu)}{dt d\nu dV} - k(\nu)\Phi(\nu, z) \quad (4.40)$$

а затем проинтегрировать по поперечным направлениям

$$F(\nu) = \frac{1}{D^2} \int dS(x, y) \Phi(\nu, x, y, z_{max}) \quad (4.41)$$

где  $z_{max}$  - координата поверхности источника.

## Синхротронные потери

В случае изотропного релятивистского распределения электронов, средняя потеря одним электроном описывается как

$$\frac{dE}{dt} = -\frac{4}{3} \sigma_T c \frac{B^2}{8\pi} \gamma_e^2 \quad (4.42)$$

где  $\sigma_T$  Томпсоновское сечение, а  $\gamma_e$  Лоренц-фактор электрона. Можно переписать это выражение как

$$\frac{dE}{dt} = -\frac{4}{9} \frac{e^4 B^2}{m^4 c^7} E^2 \quad (4.43)$$

Введем параметр, характеризующий скорость потерь  $l = \frac{4}{9} \frac{e^4 B^2}{m^4 c^7}$ , тогда в момент времени  $t$  энергия частицы с начальной энергией  $E_0$  будет равна

$$E(t) = \frac{E_0}{1 + E_0 l t} \quad (4.44)$$

Теперь вычислим как изменяется функция распределения электронов из-за синхротронных потерь. Если начальная функция распределения равна  $F_0(E_0)$ , и потери энергии каждой частицы описываются уравнением 4.43, тогда частицы мигрируют к более низким энергиям и с учетом сохранения числа частиц, функция распределения меняется как

$$F(E, t) = F_0(E_0(E, t)) dE_0/dE = F_0\left(\frac{E}{1 - E l t}\right) \frac{1}{(1 - E l t)^2} \quad (4.45)$$

Если величина  $l$  не постоянна во времени, тогда во всех уравнениях выше нужно заменить множитель  $l t$  на интеграл  $\int_0^t l(t') dt'$ .

## 4.4 Распад пионов

В коде реализовано два способа вычисления спектра гамма фотонов, излученных в процессе распада пионов образовавшихся в результате протон-протонного рассеяния. Более простой описан в работе [12]. Количество фотонов излученных в результате одного рассеяния считается пропорциональным полному сечению неупругого протон-протонного рассеяния, которое можно аппроксимировать следующим образом [13]

$$\sigma_{inel} = (30.7 - 0.96 \ln(E_p/E_{th}) + 0.18 \ln^2(E_p/E_{th})) \times (1 - (E_{th}/E_p)^{1.9})^3 \cdot 10^{-27} \text{см}^2 \quad (4.46)$$

где  $E_p$  кинетическая энергия ускоренного протона,  $E_{th}$  пороговая энергия реакции  $E_{th} = 2m_\pi + m_\pi^2/2m_p \approx 0.2797 \text{ GeV}$

Количество фотонов излученных в единицу энергии  $(\epsilon, \epsilon + d\epsilon)$  в единицу объема в единицу времени равно

$$\frac{dN}{dt d\epsilon dV} = c n_{amb} \int_\epsilon^{\text{inf}} \sigma_{inel}(E_p) F(E_p) F_\gamma(\epsilon/E_p, E_p) dE_p/E_p \quad (4.47)$$

где  $n_{amb}$  концентрация фоновых протонов, а функция  $F_\gamma(\epsilon/E_p, E_p)$  определена как

$$F_\gamma(x, E_p) = B \frac{\ln(x)}{x} \left( \frac{1 - x^\beta}{1 + k x^\beta (1 - x^\beta)} \right)^4 \times \left( \frac{1}{\ln(x)} - \frac{4\beta x^\beta}{1 - x^\beta} - \frac{4k\beta x^\beta (1 - 2x^\beta)}{1 + k x^\beta (1 - x^\beta)} \right) \quad (4.48)$$

Параметры  $B, \beta, k$  зависят только от энергии ускоренного протона и в диапазоне энергий  $0.1 \text{ ТэВ} < E_p < 10^5 \text{ ТэВ}$  могут быть выражены как  $B = 1.3 + 0.14L + 0.011L^2$ ,  $\beta = 1.0 / (1.79 + 0.11L + 0.008L^2)$  и  $k = 1.0 / (0.801 + 0.049L + 0.014L^2)$ , где  $L = \ln(E_p / 1 \text{ ТэВ})$

В более сложной модели, описанной в работе [13], на низких энергиях ( $E_p < 2 \text{ ГэВ}$ ) используется непосредственно сечение рождение пионов, в то время как на высоких используется доля полного сечения неупругого рассеяния  $\sigma_{inel}$ , но с более точными поправочными коэффициентами.

На низких энергиях сечение рождения пионов  $\sigma_\pi$  складывается из сечений трех процессов - с рождением одного пиона  $pp \rightarrow pp\pi^0$  и с рождением двух пионов  $pp \rightarrow pp2\pi^0$  и  $pp \rightarrow \{pn, D\}\pi^+\pi^0$ . Первое сечение выражается как

$$\sigma_{1\pi} = 7.66 \times 10^{-30} \eta^{1.95} (1 + \eta + \eta^5) \times (f_{BW}(\sqrt{s}))^{1.86} \text{ см}^2 \quad (4.49)$$

где  $s = 2m_p(E_p + 2m_p)$  квадрат энергии в системе центра инерции,  $\eta$  нормированный максимальный импульс пиона

$$\eta = \frac{\sqrt{(s - m_\pi^2 - 4m_p^2)^2 - 16m_\pi^2 m_p^2}}{2m_\pi \sqrt{s}} \quad (4.50)$$

и  $f_{BW}$  релятивистское распределение Брейта-Вигнера

$$f_{BW}(\sqrt{s}) = \frac{m_p K}{\left((\sqrt{s} - m_p)^2 - M_{res}^2\right)^2 + M_{res}^2 \Gamma_{res}^2} \quad (4.51)$$

где  $K = \sqrt{8} M_{res} \Gamma_{res} \gamma / \pi \sqrt{M_{res}^2 + \gamma}$  и  $\gamma = \sqrt{M_{res}^2 (M_{res}^2 + \Gamma_{res}^2)}$ . Значения используемых параметров равны  $M_{res} = 1.1883 \text{ ГэВ}$  и  $\Gamma_{res} = 0.2264 \text{ ГэВ}$ .

Полное сечение процессов, в которых рождаются два пиона, равно

$$\sigma_{2\pi} = \frac{5.7 \cdot 10^{-27}}{1 + \exp(-9.3(E_p / 1 \text{ GeV} - 1.4))} \quad (4.52)$$

Для энергий  $E_p > 0.56 \text{ GeV}$ , а для более низких энергий оно равно нулю.

Для энергий выше 2 ГэВ, сечение рождения пиона вычисляется через среднее число рождения пионов при одном столкновении и полное сечение неупругого рассеяния **4.46**  $\sigma_\pi = \langle n_\pi \rangle \sigma_{inel}$ . Для энергий меньше 5 ГэВ среднее число рожденных пионов равно  $\langle n_\pi \rangle = -0.006 + 0.237Q - 0.023Q^2$  где  $Q = (E_p - E_{th}) / m_p$ . Для энергий выше 5 ГэВ is  $\langle n_\pi \rangle = a_1 \xi^{a_4} (1 + \exp(-a_2 \xi^{a_5})) (1 - \exp(-a_3 \xi^{1/4}))$ , где  $\xi = (E_p / 1 \text{ ГэВ} - 3) / m_p$  и коэффициенты  $a_1 - a_5$  вычислены с численным моделированием с помощью кода GEANT [25],  $a_1 = 0.728, a_2 = 0.596, a_3 = 0.491, a_4 = 0.2503, a_5 = 0.117$ .

Теперь, зная  $\sigma_\pi$ , можно выразить сечение излучения фотона в единицу энергии

$$\frac{d\sigma_\gamma(E_p, \epsilon)}{d\epsilon} = A_{max}(E_p) \times F(E_p, \epsilon) \quad (4.53)$$

где  $A_{max}(E_p)$  максимум сечения по всем энергиям фотона, который зависит только от энергии протона, а  $F(E_p, \epsilon)$  описывает форму спектра. Для явного выражения  $F(E_p, \epsilon)$  нужно сначала ввести несколько величин, описывающих кинематику.  $Y = \epsilon + \frac{m_\pi^2}{4\epsilon}$ ,  $Y_{max} = \epsilon_{max} + \frac{m_\pi^2}{4\epsilon_{max}}$ , где  $\epsilon_{max}$  максимальная энергия излученного фотона при данной энергии протона,  $X = \frac{Y - m_\pi}{Y_{max} - m_\pi}$ . Энергия пиона в системе центра инерции равна  $E_{CM} = \frac{s - 4m_p^2 + m_\pi^2}{s\sqrt{s}}$ , а максимальная энергия пиона в лабораторной системе -  $E_{max} = \gamma_{CM} \left( E_{CM} + \sqrt{E_{CM}^2 - m_\pi^2} \beta_{CM} \right)$ , где лоренц-фактор центра инерции в лабораторной системе  $\gamma_{CM} = \frac{E_p + 2m_p}{\sqrt{s}}$  и  $\beta_{CM} = \sqrt{1 - 1/\gamma_{CM}^2}$ . Теперь выражение для  $F(E_p, \epsilon)$  можно записать как

$$F(E_p, \epsilon) = \frac{(1 - X^{\alpha(E_p)})^{\beta(E_p)}}{(1 + XY_{max}/\lambda m_\pi)^{\gamma(E_p)}} \quad (4.54)$$

Явные выражения для  $\lambda, \alpha(E_p), \beta(E_p), \gamma(E_p)$  можно найти в работе [13].

Величина  $A_{max}$  может быть выражена как  $5.9 \times \frac{\sigma_\pi}{E_{max}}$  для протонов с кинетическими энергиями меньше  $< 1$  ГэВ, и как  $b_1 (E_p/m_p)^{-b_2} \exp(b_3 \ln^2(E_p/m_p)) \times \frac{\sigma_\pi}{m_p}$  для высоких энергий. Выражения для  $b_1, b_2, b_3$  так же можно найти в работе [13]. Полное число излученных фотонов в единице времени в единице объема в единицу энергии в интервале энергий  $(\epsilon, \epsilon + d\epsilon)$  может быть найдено интегрированием  $\frac{d\sigma_\gamma(E_p, \epsilon)}{d\epsilon}$  с функцией распределения протонов.

$$\frac{dN}{dt d\epsilon dV} = cn_{amb} \int_\epsilon^{\inf} \frac{d\sigma_\gamma(E_p, \epsilon)}{d\epsilon} F(E_p) dE_p \quad (4.55)$$

## 4.5 Тормозное излучение

Для простого случая тормозного излучения горячей плазмы мы используем формулы, описанные в книге [18]. Количество энергии, излучаемой в единицу энергии, в единицу времени единицей объема равно

$$\frac{dW}{dt d\epsilon dV} = \frac{2^5 \pi e^6}{3 h m c^3} \sqrt{\frac{2\pi}{3 k m}} T^{-1/2} Z^2 n_e n_i \exp\left(-\frac{\epsilon}{kT}\right) \bar{G} \quad (4.56)$$

где  $T$  температура плазмы,  $n_e, n_i$  концентрации электронов и ионов соответственно,  $Z$  зарядовое число ионов и  $\bar{G}$  гаунт-фактор, усредненный по скоростям частиц. Приближенные выражения для  $\bar{G}$  приведены в книге Новикова и Торна [26] и исправлены Райбеки и Лайтманом [18]. Формулы для разных режимов рассеяния - малоуглового и рассеяния на большие углы, а так же для классического и квантового режима, когда становится важен принцип неопределенности (П.Н.), приведены на рисунке 4.2, где  $R_y$  - постоянная Ридберга и  $\xi$  постоянная Эйлера-Маскерони.

Для случая произвольных изотропных распределений рассеивающихся электронов мы используем уравнения приведенные в работе [21]. Количество фотонов, излученных в еди-

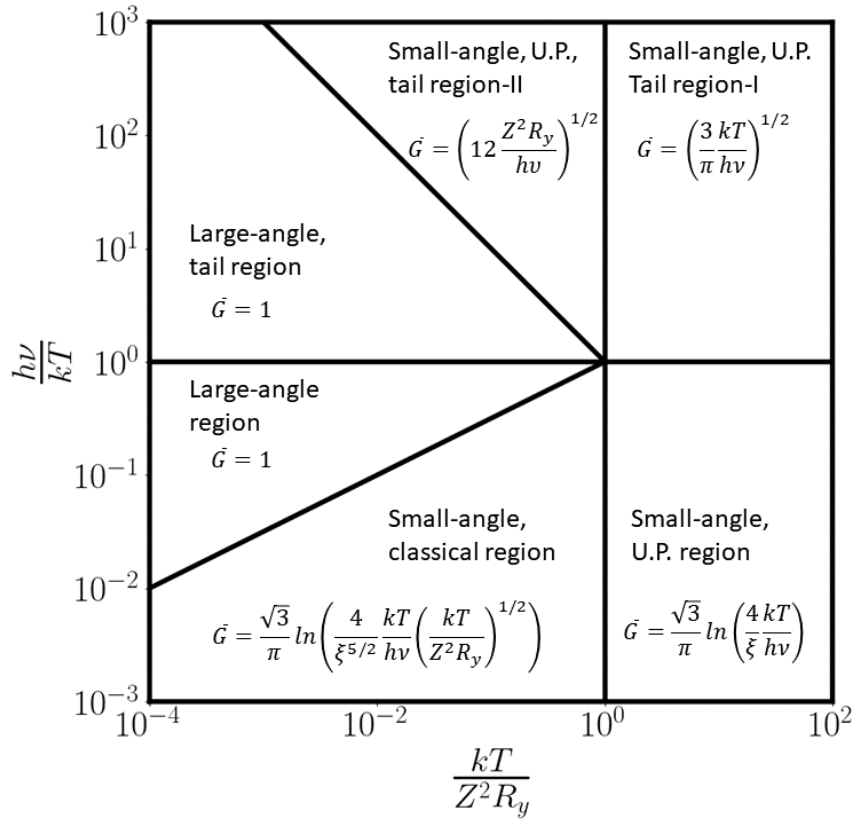


Рисунок 4.2: Усредненные гаунт-факторы, цитируется по Райбеки и Лайтман [18], выведены Новиковым и Торном [26]

ницу времени в единицу энергии единицей объема заданным распределением электронов определяется как

$$\frac{dN}{dt d\epsilon dV} = \int \frac{dN_\gamma(E_e, \epsilon)}{dt d\epsilon} F(E_e) dE_e \quad (4.57)$$

где  $\frac{dN_\gamma(E_e, \epsilon)}{dt d\epsilon}$  число фотонов излученных одним лектроном с энергией  $E_e$ . Эту величину можно выразить как

$$\frac{dN_\gamma(E_e, \epsilon)}{dt d\epsilon} = v_e \left( \left( \sum n_i Z_i^2 \right) \sigma_{e-p} + n_e \sigma_{e-e} \right) \quad (4.58)$$

где  $v_e$  скорость электрона,  $n_i, Z_i$  концентрации и зарядовые числа каждого типа ионов, а  $\sigma_{e-p}$  и  $\sigma_{e-e}$  сечения электрон-протонного и электрон-электронного рассеяния соответственно.

Электрон-электронное сечение в релятивистском режиме ( $E_e > 2$  МэВ) может быть выражено следующим образом, как показано в работе [27]

$$\sigma_{e-e}(E_e, \epsilon) = (\sigma_1 + \sigma_2) A(\epsilon, E_e/m_e c^2) \quad (4.59)$$

Первый член в скобках определяется как

$$\sigma_1 = \frac{4r_e^2 \alpha}{\epsilon} \left( 1 + \left( \frac{1}{3} - \frac{\epsilon}{\gamma_e} \right) \left( 1 - \frac{\epsilon}{\gamma_e} \right) \right) \left( \ln \left( 2\gamma_e \frac{\gamma_e - \epsilon}{\epsilon} \right) - \frac{1}{2} \right) \quad (4.60)$$

Выражение для  $\sigma_2$  более сложное и разделяется на два случая. Для  $\epsilon < 1/2$ :

$$\sigma_2 = \frac{r_e^2 \alpha}{3\epsilon} \left( 16(1 - \epsilon + \epsilon^2) \ln \frac{\gamma_e}{\epsilon} - \frac{1}{\epsilon^2} + \frac{3}{\epsilon} - 4 + 4\epsilon - 8\epsilon^2 - 2(1 - 2\epsilon) \ln(1 - 2\epsilon) \left( \frac{1}{4\epsilon^3} - \frac{1}{2\epsilon^2} + \frac{3}{\epsilon} - 2 + 4\epsilon \right) \right) \quad (4.61)$$

и для  $\epsilon > 1/2$ :

$$\sigma_2 = \frac{r_e^2 \alpha}{3\epsilon} \frac{2}{\epsilon} \left( \left( 4 - \frac{1}{\epsilon} + \frac{1}{4\epsilon^2} \right) \ln(2\gamma_e) - 2 + \frac{2}{\epsilon} - \frac{5}{8\epsilon^2} \right) \quad (4.62)$$

где  $r_e$  классический радиус электрона и  $\alpha$  постоянная тонкой структуры. Поправочный фактор  $A(\epsilon, \gamma_e)$  определяется как

$$A(\epsilon, \gamma_e) = 1 - \frac{8}{3} \frac{(\gamma_e - 1)^5}{\gamma_e + 1} \left( \frac{\epsilon}{\gamma_e} \right)^{1/3} \quad (4.63)$$

В нерелятивистском режиме электрон-электронное сечение равно

$$\sigma_{NR} = \frac{4r_e^2 \alpha}{15\epsilon} F \left( \frac{4\epsilon}{\gamma_e^2 - 1} \right) \quad (4.64)$$

где

$$F(x) = \left( 1 + \frac{1}{2} (\gamma_e^2 - 1) \right) \left( 17 - \frac{3x^2}{(2-x)^2} - \frac{10x\gamma_e\beta_e(2 + \gamma_e\beta_e)}{1 + x^2(\gamma_e^2 - 1)} \right) \sqrt{1-x} + \left( 12(2-x) - \frac{7x^2}{2-x} - \frac{3x^4}{(2-x)^3} \right) \ln \frac{1 + \sqrt{1-x}}{\sqrt{x}} \quad (4.65)$$

Сечение для электрон-протонного рассеяния приведено Бёте и Хайтлером в работе [19]. Можно заметить, что в ультрарелятивистском режиме оно равно слагаемому  $\sigma_1$  в формуле для электрон-электронного рассеяния 4.60 [20]. В общем случае сечение Бёте-Хайтлера равно

$$\sigma_{e-p} = \frac{r_e^2 \alpha p'}{\epsilon p} \left( \frac{4}{3} - \frac{2}{(\beta_e \beta'_e)^2} \left( \frac{\gamma_e}{\gamma'_e} + \frac{\gamma'_e}{\gamma_e} - \frac{2}{\gamma_e \gamma'_e} \right) + \left( l \frac{\gamma'_e}{\gamma_e^2 - 1} + l' \frac{\gamma_e}{\gamma_e'^2 - 1} - ll' \right) + L \left( \frac{8}{3} \gamma_e \gamma'_e + \epsilon^2 \left( \frac{1}{(\beta_e \beta'_e)^2 + 1} \right) + \frac{1}{2} \left( l \left( 1 + \frac{\gamma'_e}{\beta_e^2 \gamma_e} \right) - l' \left( 1 + \frac{\gamma_e}{\beta_e'^2 \gamma'_e} \right) + 2 \frac{\epsilon}{\beta_e^2 \gamma_e \beta_e'^2 \gamma'_e} \right) \right) \right) \quad (4.66)$$

В этой формуле используются следующие вспомогательные величины:  $\gamma_e$  - лоренц-фактор начального электрона,  $\beta_e$  - его скорость в единицах скорости света,  $l = \frac{1}{\beta_e \gamma_e} \ln \frac{1+\beta_e}{1-\beta_e}$ , штрихованные величины относятся к рассеянному электрону и они связаны соотношением  $\gamma_e = \gamma'_e + \epsilon$ . Так же введена дополнительная величина  $L = \frac{2}{\beta_e \gamma_e \beta'_e \gamma'_e} \ln \frac{\gamma_e \gamma'_e (1 + \beta_e \beta'_e) - 1}{\epsilon}$ .



# 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. Bethe H., Heitler W. On the Stopping of Fast Particles and on the Creation of Positive Electrons // *Proceedings of the Royal Society of London Series A*. — 1934. — Vol. 146, no. 856. — P. 83–112.
20. Jauch J. M., Rohrlich F. [The theory of photons and electrons. The relativistic quantum field theory of charged particles with spin one-half](#). *Texts and Monographs in Physics*. — 2nd ed. edition. — Berlin : Springer, 1980. — ISBN: [978-3-642-80953-8](#), [978-3-642-80951-4](#).
21. Baring Matthew G., Ellison Donald C., Reynolds Stephen P. et al. Radio to Gamma-Ray Emission from Shell-Type Supernova Remnants: Predictions from Nonlinear Shock Acceleration Models // *ApJ*. — 1999. — Vol. 513, no. 1. — P. 311–338.
22. Landau L. D., Lifshitz E. M. *The classical theory of fields*. — 1975.
23. Ghisellini Gabriele, Svensson Roland. The synchrotron and cyclo-synchrotron absorption cross-section // *MNRAS*. — 1991. — Vol. 252. — P. 313–318.
24. Ghisellini Gabriele. [Radiative Processes in High Energy Astrophysics](#). — 2013. — Vol. 873.
25. Allison J., Amako K., Apostolakis J. et al. Geant4 developments and applications // *IEEE Transactions on Nuclear Science*. — 2006. — Vol. 53, no. 1. — P. 270–278.
26. Novikov I. D., Thorne K. S. *Astrophysics of black holes*. // *Black Holes (Les Astres Occlus)*. — 1973. — P. 343–450.
27. Baïer V. N., Fadin V. S., Khoze V. A. Photon Bremsstrahlung in Collision of High-energy Electrons // *Soviet Journal of Experimental and Theoretical Physics*. — 1967. — Vol. 24. — P. 760.