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

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. Так же возможен учет эволюции источников и их излучения во времени.

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 дирректории кода.

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

```
$ g++ -o faina *.cpp
$ ./faina
```

Быстрый старт

Рассмотрим простейший пример, приведенный в процедуре evaluateSimpleSynchrotron в файле examples.cpp. В данном примере рассмытривается 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, парметры 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

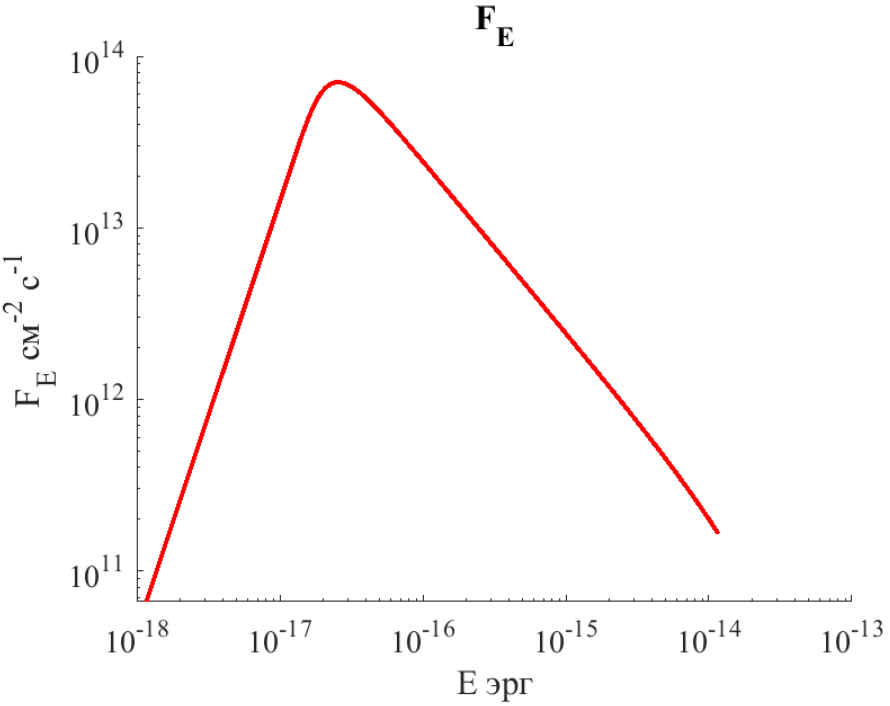

Рисунок 1: Энергетическая плотность потока синхротронного излучения от тестового источника

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^2 = см$^{-2}$с$^{-1}$. Если необходим вывод в других единицах, то запись в файл нужно переписать самостоятельно.

```
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

Расчет излучения источников

FAINA позволяет рассчитывать электромагнитное излучение от источников с заданными функциями распределения излучающих частиц и другими параметрами. Построены модели следующих типов излучения: синхротронного, обратного комптоновского рассеяния, пионного распада в результате свободно-свободного взаимодействия протонов и тормозного излучения.

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 рас-

Таблица 1.1: Публичные методы класса ParticleDistribution

ParticleDistribution	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

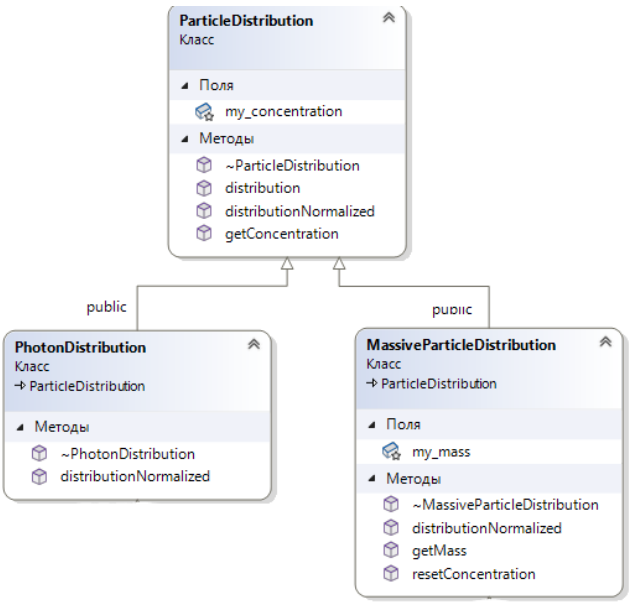

Рисунок 1.1: Схема наследования распределения фотонов и массивных частиц

Таблица 1.2: Публичные методы класса MassiveParticleDistribution

MassiveParticleDistribution	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
void resetConcentration(const double& n)	позволяет изменить полную концентрацию частиц в распределении

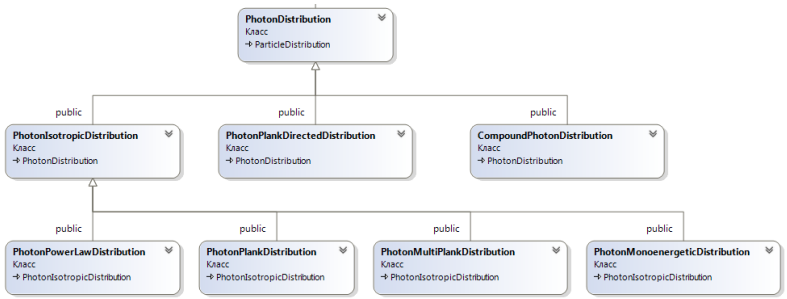

Рисунок 1.2: Схема наследования классов распределений фотонов

пределение по энергии, но сконцентрированное в некотором телесном угле, такое распределение удобно для модели излучения звезды на некотором расстоянии от нее, и CompoundPhotonDistribution, представляющий из себя сумму нескольких распределений фотонов общего вида. Схема наследования классов фотонных распределений представлена на рисунке 1.2.

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 4π.

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 моноэнер-

гетичного изотропного распределения. Метода класса PhotonIsotropicDistribution и его наследников перечислены в таблице 1.3

Таблица 1.3: Публичные методы классов изотропных распределений фотонов

PhotonIsotropicDistribution	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 4π
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
PhotonPowerLawDistribution	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
PhotonPlankDistribution	Класс для планковского распределения фотонов
PhotonPlankDistribution(const double& temperature, const double& amplitude)	конструктор, создающий экземпляр с заданными температурой и апмплитудой - 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
PhotonMultiPlankDistribution	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
PhotonMonoenergeticDistribution	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

Класс CompoundPhotonDistribution предназначен для представления смеси различных распределений фотонов, не обязательно планковских, как PhotonMultiPlankDistribution, и не обязательно изотропных. Его методы описаны в Таблице 1.4

Таблица 1.4: Публичные методы класса CompoundPhotonDistribution

CompoundPhotonDistribution	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)	конструкторб создающий экземпляр содержащий смесь из трех распределений

Единственное реализованное в коде анизотропное распределение фотонов - это PhotonPlankDirectedDistribution, представляющий направленное планковское излучение. Пользователь может реализовать другие виды анизотропных излучений самостоятельно, создав класс, наследующий от PhotonDistribution и определив необходимый виртуальный метод distributionNormalized(const double& energy, const double& mu, const double& phi). Методы класса PhotonPlankDirectedDistribution описаны в Таблице 1.5

Таблица 1.5: Публичные методы класса PhotonPlankDirectedDistribution

PhotonPlankDirectedDistribution	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

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 4π.

Таблица 1.6: Публичные методы класса MassiveParticleIsotropicDistribution

MassiveParticleIsotropicDistribution	Абстрактный класс для изотропных распределений
double distribution(const double& energy)	возвращает функцию распределения с отброшенными угловыми аргументами, то есть нормированную на концентрацию, деленную на 4π
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 междду данными минимальной и максимальной енергиями с заданным количеством точек, которые распределяются логарифмически

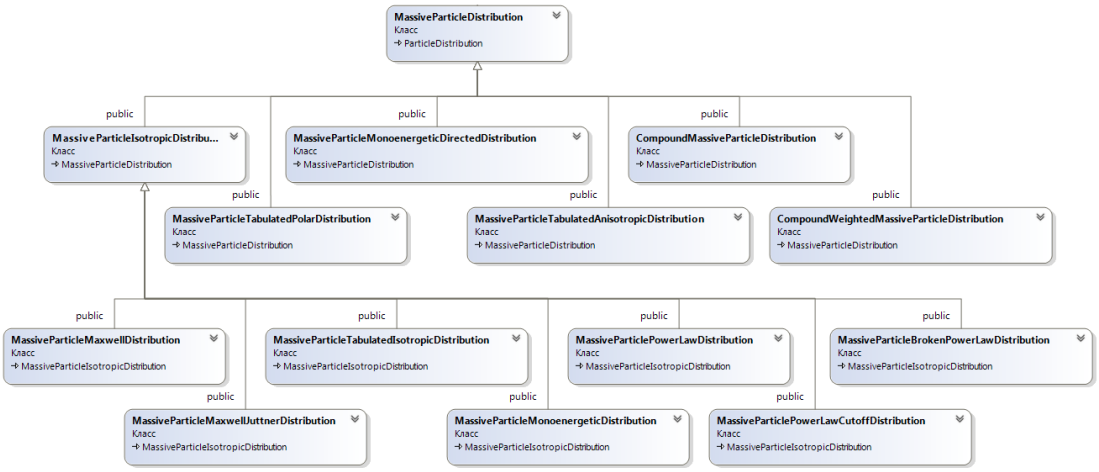

Рисунок 1.3: Схема наследования классов распределения массивных частиц

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 распредлений 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 1.7. Пользователь может сам реализовывать необходимые ему виды распределений излучающих частиц, создав наследника класса MassiveParticleDistribution или MassiveParticleIsotropicDistribution и определив необходимые виртуальные методы.

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 перечислимы 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.

Таблица 1.7: Публичные методы классов распределений массивных частиц

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

double getIndex()	возвращает степенной индекс распределения
double getBeta()	возвращает параметр завала распределения
double getE0()	возвращает начальную энергию распределения
double getEcutoff()	возвращает энергию экспоненциального завала
MassiveParticleMaxwellDistribution	Класс для распределения Максвелла
MassiveParticleMaxwellDistribution(const double& mass, const double& temperature, const double& concentration)	конструктор, создает экземпляр распределния Максвелла частиц с заданными массой, температурой и полной концентрацией
double getTemperature()	возвращает температуру распределения
MassiveParticleMaxwellJuttnerDistribution	Класс для распределения Максвелла-Юттнера
MassiveParticleMaxwellJuttnerDistribution(const double& mass, const double& temperature, const double& concentration)	конструктор, создает экземпляр распределния Максвелла-Юттнера частиц с заданными массой, температурой и полной концентрацией
double getTemperature()	возвращает температуру распределения
MassiveParticleTabulatedIsotropicDistribution	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 распределния частиц с заданными массой и полной концентрацией с помощью указанного файла, состоящего из двух колнок 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 распределния частиц с заданными массой и полной концентрацией с помощью указанных двух файлов, состоящих из колнок 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 распределния 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 getEmin()	возвращает минимальную энергию распределения
double getEmax()	возвращает максимальную энергию распределения
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
void addPowerLaw(const double& Epower, const double& index)	добавляет к функции распределения степенной с указанным индексом, начиная с указанной энергии. Функция распределения при этом остается нормированной на указанную ранее концентрацию
MassiveParticleMonoenergeticDistribution	Класс для моноэнергичного изотропного распределения
MassiveParticleMonoenergeticDistribution(const double& mass, const double& Energy, const double& halfWidth, const double& concentration)	конструктор, принимающий массу, среднюю энергию, полуширину разброса по энергии и концентрацию
MassiveParticleTabulatedPolarDistribution	Класс для таблично заданного распределения с зависимостью от полярного угла
MassiveParticleTabulatedPolarDistribution(const double& mass, const char* energyFileName, const char* muFileName, const char* distributionFileName, const int Ne, const int Nmu, const double& concentration, DistributionInputType inputType)	конструктор, создает экземпляр табличного распределния 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.

MassiveParticleTabulatedPolarDistribution(const double& mass, const double* energy, const double* mu, const double** distribution, const int Ne, const int Nmu, const double& concentration, DistributionInputType inputType)	конструктор, создает экземпляр табличного распределния 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.
int getNe()	возвращает количество ячеек по энергии в таблице задающей функцию распределения
double getEmin()	возвращает минимальную энергию распределения
double getEmax()	возвращает максимальную энергию распределения
int getNmu()	возвращает количество ячеек по полярному углу в таблице задающей функцию распределения
void double rescaleDistribution(const double& k)	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
MassiveParticleTabulatedAnisotropicDistribution	Класс для таблично заданного анизотропного распределения общего вида
MassiveParticleTabulatedAnisotropicDistribution(const double& mass, const char* energyFileName, const char* muFileName, const char* distributionFileName, const int Ne, const int Nmu, const int Nphi, const double& concentration, DistributionInputType inputType)	конструктор, создает экземпляр табличного распределния 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 распределния 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 расномерной и определяется только размером. Так же указывается формат входных данных.
int getNe()	возвращает количество ячеек по энергии в таблице задающей функцию распределения
double getEmin()	возвращает минимальную энергию распределения
double getEmax()	возвращает максимальную энергию 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
MassiveParticleMonoenergeticDirectedDistribution	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

MassiveParticleMovingDistribution	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
CompoundMassiveParticleDistribution	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
CompoundWeightedMassiveParticleDistribution	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

1.2 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 моде-

ли 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.

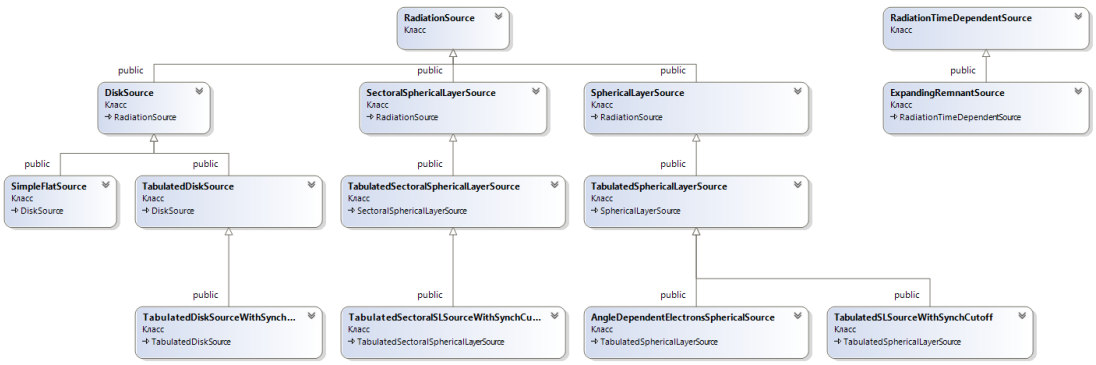

Рисунок 1.4: Схема наследования классов источников излучения

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 пространственной 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 цилиндрическо пространственной сетке изображена на рисунке 1.5. Цветом обозначена доля объема ячейки, заполненная веществом источника.

Так же источники излучения имеют следующие важные характеристки, 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.

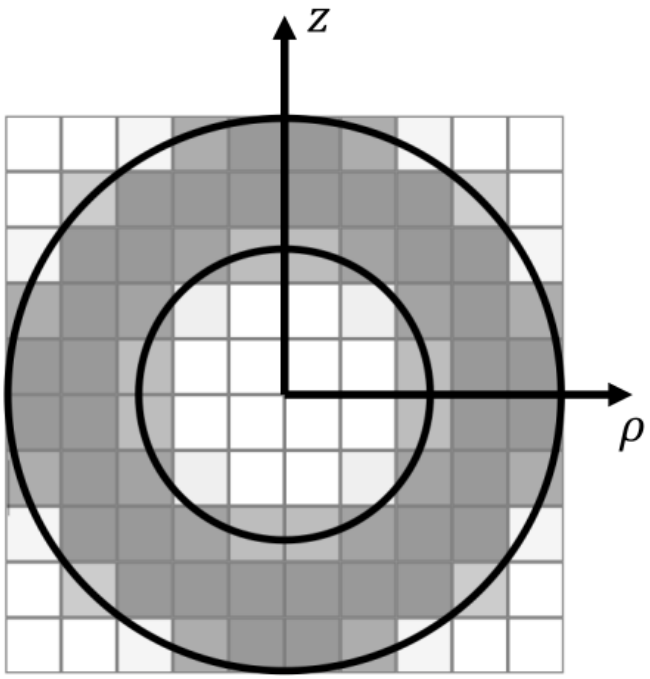

Рисунок 1.5: Модель источника в форме шарового слоя, помещенного в цилиндрическую пространственную координатную сетку. Цвет характеризует долю объема ячейки, заполненную веществом источника.

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 можели данного источника считается, что распределение частиц генерирутся 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 определеяется формулой:

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

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 частицц, 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 1.8.

Таблица 1.8: Публичные методы классов источников излучения без зависимости от времени

RadiationSource	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 $\sigma = \frac{B^2}{4\pi n m_p c^2}$
virtual double getAverageConcentration()	чисто виртуальный метод, возвращает среднюю конценрацию
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 по азимутальному углу цилиндрических координат
double getDistance()	возвращает расстояние до источника
getArea(int irho)	возвращает поперечное сечение данной пространственной ячейки
getVolume(int irho, int iz, int iphi)	возвращает объем ячейки, занятый веществом источника. Этот метод согласован с методами getArea и getLength и возвращает их произведение
virtual getB(int irho, int iz, int iphi)	чисто виртуальный метод, возвращает значение магнитного поля в ячейке
virtual getConcentration(int irho, int iz, int iphi)	чисто виртуальный метод, возвращает значение концентрации в ячейке
virtual getSinTheta(int irho, int iz, int iphi)	чисто виртуальный метод, возвращает синус угла наклона магнитного поля к лучу зрения
virtual void getVelocity(int irho, int iz, int iphi, double& velocity, double& theta, double& phi) чисто виртуальный метод, возвращает скорость данной ячейки источника	

virtual getTotalVolume()	чисто виртуальный метод, возвращает полный объем источника
virtual getLength(int irho, int iz, int iphi)	чисто виртуальный метод, возвращает среднюю толщину ячейки, заполненную веществом источника
virtual resetParameters(const double* parameters, const double* normalizationUnits)	чисто виртуальный метод, меняющий параметры источника. Список параметров, их количество, их влияние на источник определяются пользователем в конкретных реализациях класса. Принимет массив параметров и массив единиц в которых они измерены. Данный метод используется в процедурах оптимизации, либо при учете изменения источника со временем
virtual getParticleDistribution(int irho, int iz, int iphi)	чисто виртуальный метод, возвращает распределение излучающих частиц в ячейке
DiskSource	Абстрактный класс для источников в форме диска
SimpleFlatSource	Класс для источников в форме однородного диска
SimpleFlatSource(MassiveParticleDistribution* electronDistribution, const double& B, const double& sinTheta, const double& rho, const double& z, const double& distance, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными распределением частиц, магнитным полем, синусом угла его наклона, радиусом диска, толщиной диска, расстоянием до источника и скоростью движения вещества
TabulatedDiskSource	Класс для источников в форме диска с таблично заданными значениями параметров
TabulatedDiskSource(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, double*** B, double*** sinTheta, double*** concentration, const double& rho, const double& z, const double& distance, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными с помощью массивов распределением частиц, магнитным полем, синусом угла его наклона, а так же заданными радиусом диска, толщиной диска, расстоянием до источника и скоростью движения вещества
TabulatedDiskSource(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, const double& B, const double& sinTheta, const double& concentration , const double& rho, const double& z, const double& distance, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными однородными распределением частиц, магнитным полем, синусом угла его наклона, а так же заданными радиусом диска, толщиной диска, расстоянием до источника и скоростью движения вещества

TabulatedDiskSourceWithSynchCutoff	Класс для источников в форме диска с таблично заданными значениями параметров и учетом синхротронных потерь энергии частиц
TabulatedDiskSourceWithSynchCutoff(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, double*** B, double*** theta, double*** concentration, const double& rho, const double& z, const double& distance, const double& downstreamVelocity, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными с помощью массивов распределением частиц, магнитным полем, синусом угла его наклона, а так же заданными радиусом диска, толщиной диска, расстоянием до источника, скоростью конвекции частиц и скоростью движения вещества
TabulatedDiskSourceWithSynchCutoff(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, const double& B, const double& concentration, const double& theta, const double& rho, const double& z, const double& distance, const double& downstreamVelocity, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными однородными распределением частиц, магнитным полем, синусом угла его наклона, а так же заданными радиусом диска, толщиной диска, расстоянием до источника, скоростью конвекции частиц и скоростью движения вещества
SphericalLayerSource	Абстрактный класс для источников в форме шарового слоя
double getInnerRho()	возвращает внутренний радиус шарового слоя
TabulatedSphericalLayerSource	Класс для источников в форме шарового слоя с таблично заданными значениями параметров
TabulatedSphericalLayerSource(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, double*** B, double*** sinTheta, double*** concentration, const double& rho, const double& rhoin, const double& distance, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными с помощью массивов распределением частиц, магнитным полем, синусом угла его наклона к лучу зрения, а так же заданными внешним и внутренним радиусом шарового слоя, расстоянием до источника и скоростью движения вещества
TabulatedSphericalLayerSource(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, const double& B, const double& concentration, const double& sinTheta, const double& rho, const double& rhoin, const double& distance, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными однородными распределением частиц, магнитным полем, синусом угла его наклона, а так же заданными внутренним и внешним радиусом шарового слоя, расстоянием до источника и скоростью движения вещества

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

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

TabulatedSectoralSLSourceWithSynchCutoff(int Nrho, int Nz, int Nphi, MassiveParticleDistribution* electronDistribution, const double& B, const double& concentration, const double& theta, const double& rho, const double& rhoin, const double& minrho, const double& phi, const double& distance, const double& downstreamVelocity, const double& velocity = 0)	конструктор, возвращает экземпляр с заданными однородными распределением частиц, магнитным полем, синусом угла его наклона, а так же заданными внутренним и внешним радиусом шарового слоя, углом раствора сектора, расстоянием до источника, скоростью конвекции частиц и скоростью движения вещества

1.2.2 Источники излучения, меняющиеся со временем

Источники излучения, учитывающие зависимость от времени, представлены абастрактным классом классом RadiationTimeDependentSource. Этот класс не является наследником класса RadiationSource, но содержит экземпляр такого класса внутри себя, чтобы использовать его для расчета излучения в конкретный момент времени. Для этого пользователь должен самостоятельно создать имплементацию виртуальной функции getRadiationSource, в которой будут вычислены параметры источника в зависимости от времени. В текущей версии кода реализован только один наследник RadiationTimeDependentSource - ExpandingRemnantSource, представляющий собой модель расширяющегося остатка сверхновой. В данной модели предполагается, что размер источника 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 1.9.

Таблица 1.9: Публичные методы классов источников излучения учитывающих зависимость от времени

RadiationTimeDependentSource	Абстрактный класс для учета изменений источников излучения со временем
virtual resetParameters(const double* parameters, const double* normalizationUnits)	чисто виртуальный метод, меняющий параметры источника. Список параметров, их количество, их влияние на источник определяются пользователем в конкретных реализациях класса. Принимает массив параметров и массив единиц в которых они измерены. Данный метод применяется в процедурах оптимизации

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

1.3 Вычисление излучения

Для расчета излучения источников используется класс RadiationEvaluator и его наследники. Список публичных методов этого класса приведен в Таблице 1.10. 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), вычисляющую энергетическую плотность потока излучения источника на данной энергии принимаемого фотона в единицах $см^{-2}с^{-1}$. Далее в данном разделе описаны реализации класса RadiationEvaluator для конкретных видов излучения. Схема наследования классов вычислителей излучения представлена на рисунке 1.6. Физическая сторона вопроса, формулы по которым расчитывается излучение подробно описаны в Главе 3.

1.3.1 Синхротронное излучение

Для расчета синхротронного излучения используется класс SynchrotronEvaluator. В нем используется приближение непрерывного спектра, то есть рассматриваемые частоты

Таблица 1.10: Публичные методы класса RadiationEvaluator

RadiationEvaluator	абстрактный класс для вычисления излучения
virtual evaluateFluxFromIsotropicFunction(const double& photonFinalEnergy, MassiveParticleIsotropicDistribution* electronDistribution, const double& volume, const double& distance)	чисто виртуальный метод, возвращает энергетическую плотность потока излучаемого элементарным объемом с источника с данным распределением, на данном расстоянии от наблюдателя в единицах $см^{-2}с^{-1}$. Данный метод лучше не использовать самостоятельно, использовать вместо него расчет излучения от источников
virtual evaluateFluxFromSource(const double& photonFinalEnergy, RadiationSource* source)	чисто виртуальный метод, возвращает энергетическую плотность потока излучаемого данным источником в единицах $см^{-2}с^{-1}$
virtual resetParameters(const double* parameters, const double* normalizationUnits)	чисто виртуальный метод, позволяет изменить внутренние параметры вычислителя излучения. Список параметров, их количество, их влияние на источник определяются в конкретных реализациях класса, данный метод используется при оптимизации
writeFluxFromSourceToFile(const char* fileName, RadiationSource* source, const double& Ephmin, const double& Ephmax, const int Nph)	записывает в файл с данным именем излучение источника в единицах $см^{-2}с^{-1}$ в диапазоне от минимальной до максимальной энергии, с заданным количеством точек, распределенных логарифмически

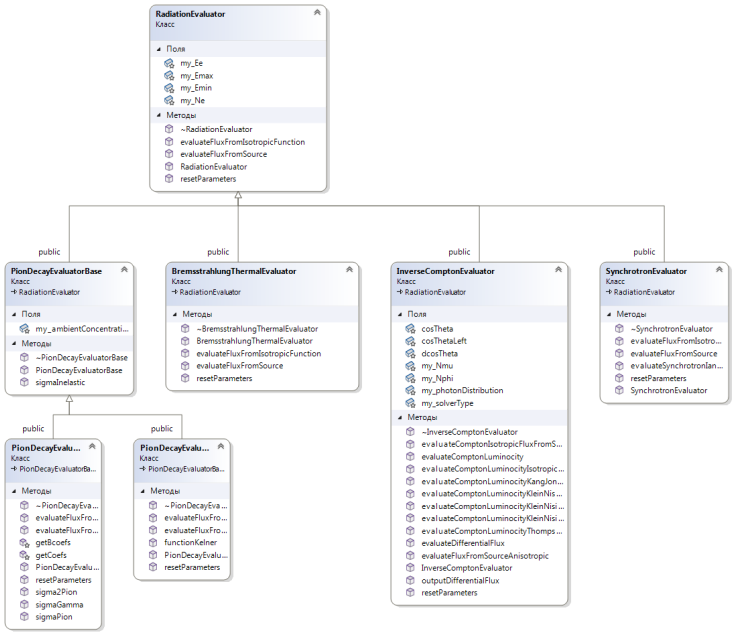

Рисунок 1.6: Схема наследования классов вычислителей излучения.

Таблица 1.11: Публичные методы класса SynchrotronEvaluator

SynchrotronEvaluator	класс предназначенный для вычисления синхротронного излучения
SynchrotronEvaluator(int Ne, double Emin, double Emax, bool selfAbsorption = true, const double& defaultB = 0, const double& defaultSinTheta = 1.0, const double& defaultLength = 0)	конструктор, создает экземпляр с указанным диапазоном рассматриваемых энергий, параметром учета самопоглощения, и значениями по умолчанию магнитного поля, его наклона и толщины
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 излучательной способности и коэффициента поглощения распределения с данной концентрацией в данном магнитном поле

фотонов предполагаются намного большими, чем частота вращения излучающих частиц в магнитном поле. Реализован случай только изотропной функции распределения излучающих частиц. Так же возможен учет синхротронного самопоглощения. Используемая геометрия источников, показанная на рисунке 1.5, позволяет легко интегрировать излучение по лучу зрения, и учитывать при этом поглощение внутри источника. При создании объекта класса необходимо указать рассматриваемый диапазон энергий частиц и количество точек в нем, параметр отвечающиза 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 1.11. Пример вычисления синхотронного излучения приведен в разделе .

1.3.2 Обратное комптоновское рассеяние

Для расчета излучения, получающегося в результате процесса обратного комптоновского рассеяния, используется класс InverseComptonEvaluator. Внутри него реализованы четыре различных метода расчета излучения, для обозначения которых используется перечислимый тип ComptonSolverType, имеющий следующие значения:

- ISOTROPIC_THOMSON - модель рассеяния в томсоновсков режиме. Реализовано только для степенного распределения электронов и теплового фотонов [6] глава 17, с 466

- ANISOTROPIC_KLEIN_NISHINA - модель расчитывающее излучение напрямую из сечения Клейна-Нишины, возможен учет анизотропных функций распределения [7, 8]

- ISOTROPIC_KLEIN_NISHINA - модель расчитывающее излучение напрямую из сечения Клейна-Нишины, но для изотропных функций распределения, что позволяет уменьшить количество интегрирований

- ISOTROPIC_JONES - модель, использующая аналитически проинтегрированное по углам сечение Клейна-Нишины [9, 10]

При создании объекта класса InverseComptomEvaluator 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 класса SynchrotronEvaluator перечислены в Таблице 1.12.

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

```
double electronConcentration = 150;
double sinTheta = 1.0;
double rmax = 1.3E17;
double B = 0.0;
double distance = 150*1E6*parsec;

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

Таблица 1.12: Публичные методы класса InverseComptonEvaluator

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

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

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

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

```
RadiationSource* source = new SimpleFlatSource(
```

```
    electrons, B, sinTheta, rmax, rmax, distance);

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

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

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

после этого вычислим в цикле желаемые потоки излучения

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

и запишем их в файл, переведя в желаемые единицы

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

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

fclose(output_ev_EFE);
```

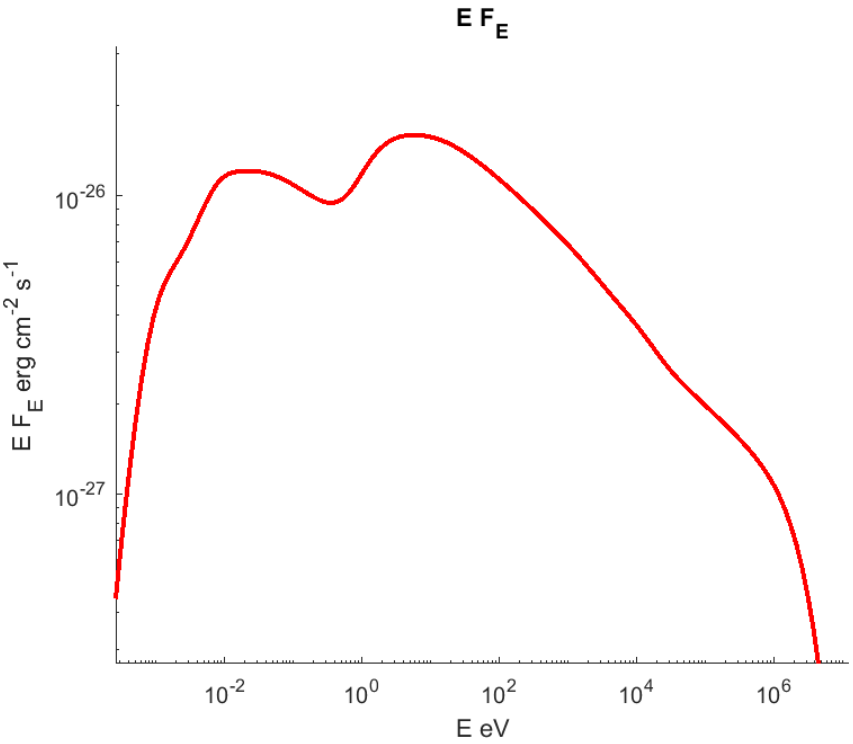

Рисунок 1.7: Энергетическая плотность потока синхротронного излучения от тестового источника

Спектр излучения, полученный в результате работы данной программы приведен на рисунке 1.7

1.3.3 Распад пионов

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

При создании объекта класса PionDecayEvaluator необходимо указать рассматриваемый диапазон энергий частиц и количество точек в нем, а так же концентрацию 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 1.13

Таблица 1.13: Публичные методы класса PionDecayEvaluatorBase и его наследников

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

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

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

double distance = 1400 * parsec;
```

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

После этого создадим распределение протонов и источник излучения

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

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

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

Как и в предыдущих случаях далее необходимо внутри цикла вычислить излучение в интересующем диапазоне энергий, используя функцию evaluateFluxFromSource, и вывести результат в файл в удобных единицах. Спектр излучения, полученный в результате работы данной программы и результаты наблюдений Кокона Лебедя на Fermi LAT, ARGO и HAWC [15, 16, 17] приведены на рисунке 1.8

1.3.4 Тормозное излучение

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

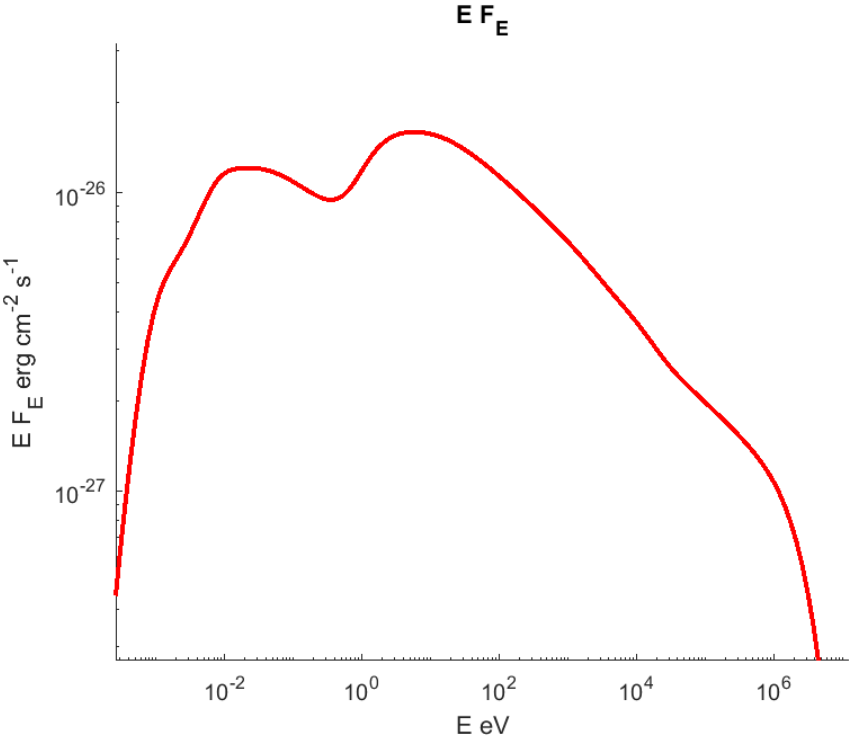

Рисунок 1.8: Расчетная энергетическая плотность потока гамма излучения Кокона Лебедя и данные наблюдений

Глава 2

Оптимизация параметров

Код FAINA позволяет не только расчитывать излучение заданных источников, но и фитировать наблюдательные данные модельными, подбирая необходимые параметры. Реализованы методы оптимизации, пригодные для произвольного числа параметров и широкого класса моделей источников. В качестве целевой функции используется взвешенная сумма квадратов отклонений по всем наблюдательным точкам $f = \sum \frac{(F_i - F_{obs,i})^2}{\sigma_i^2}$, где F_i - расчетная спектральная плотность потока излучения, $F_{obs,i}$ - наблюдаемая спектральная плотность потока излучения, σ_i - её погрешность. В текущей версии учитывается лишь погрешность измеряемого потока, ширина бина и неопределенность энергии, на которой принят сигнал не учитываются.

Реализованные методы оптимизации делятся на два типа - те, которые рассматривают излучение в один момент времени, либо постоянные во времени, и те, которые учитывают эволюцию источников и используют наблюдения в разные моменты времени. В последнем случае пользователю необходимо самостоятельно указывать, как меняются параметры источника со временем, см. раздел 1.2.2.

2.1 Фитирование источников, не зависящих от времени

Для фитирования постоянных во времени кривых блеска предназначен абстрактный класс RadiationOptimizer. В нем определена виртуальныя функция optimize(double* vector, bool* optPar, double* energy, double* observedFlux, double* observedError, int Ne, RadiationSource* source), 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, energy - массив энергий, на которых производились наблюдения, observedFlux - соответствующие наблюдаемые потоки в единицах $см^{-2}с^{-1}$, Ne - количество наблюдательных точек, source - источник излучения. Функция изменения параметров источника source->resetParameters, 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.

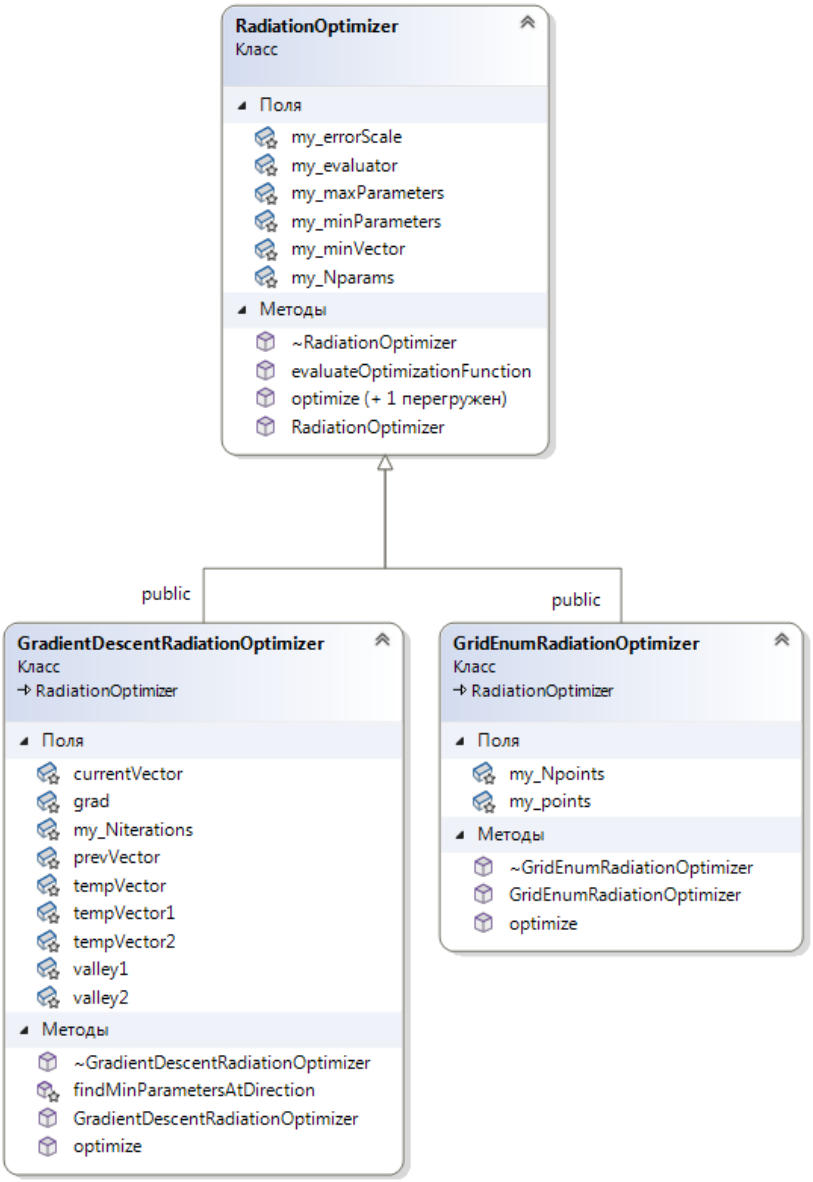

Рисунок 2.1: Схема наследования классов оптимизаторов

В коде реализованы два наследника класса RadiationOptimazer: GridEnumRadiationOptimizer - производящий поиск минимума простым перебором по сетке параметров с заданным количеством распределенных равномерно логарифмически точек, и GradientDescentRadiationOptimizer - в котором минимум находится методом градиентного спуска. Эти два класса полезно использовать совместно, используя результат работы первого как начальную точку для второго. Схема насследования классов оптимизаторов показана на рисунке 2.1, а список их публичных методов приведен в Таблице 2.1. Реализованные методы оптимизации применимы для всех описанных выше типов источников и видов электромагнитного излучения.

Таблица 2.1: Публичные методы классов оптимизаторов параметров источников

RadiationOptimizer	абстрактный класс для оптимизации параметров источника
evaluateOptimizationFunction(const double* vector, double* energy, double* observedFlux, double* observedError, int Ne, RadiationSource* source)	вычисляет целевую функцию - взвешенную сумму квадратов ошибок во всех наблюдательных точках $f = \sum \frac{(F_i - F_{obs,i})^2}{\sigma_i^2}$, где F_i - расчетная спектральная плотность потока излучения, $F_{obs,i}$ - наблюдаемая спектральная плотность потока излучения, σ_i - её погрешность
optimize(double* vector, bool* optPar, double* energy, double* observedFlux, double* observedError, int Ne, RadiationSource* source)	функция, осуществляющая оптимизацию, принимает на вход массив подбираемых параметров, в который будет записан результат, массив булевских переменных, определяющих оптимизировать соответствующий параметр, или считать его фиксированным, массив энергий, на которых производились наблюдения, соответствующие наблюдаемые потоки в единицах $см^{-2}с^{-1}$, погрешности измерения потоков, количество наблюдательных точек, и источник излучения.
optimize(double* vector, bool* optPar, double* energy, double* observedFlux, int Ne, RadiationSource* source)	функция, осуществляющая оптимизацию, в случае не заданных наблюдательных ошибок. В таком случае ошибки у всех точек считаются равными елинице и веса всех ошибок в целевой функции оказываются равными
GridEnumRadiationOptimizer	класс предназначенный для оптимизации параметров с помощью перебора по сетке
GridEnumRadiationOptimizer(RadiationEvaluator* evaluator, const double* minParameters, const double* maxParameters, int Nparams, const int* Npoints)	конструктор, создает экземпляр класса с указанным вычислителем излучения, минимальными и максимальными значениями оптимизируемых параметров, количеством этих параметров и массивом с количеством перебираемых точек по каждому параметру. При переборе точки будут распределены логарифмически равномерно по оси.

GradientDescentRadiationOptimizer	класс, предназначенный для оптимизации параметров методом градиентного спуска
GradientDescentRadiationOptimizer(RadiationEvaluator* evaluator, const double* minParameters, const double* maxParameters, int Nparams, int Niterations)	конструктор, создает экземпляр класса с указанным вычислителем излучения, минимальными и максимальными значениями оптимизируемых параметров, количеством этих параметров и максимальным количеством итераций градиентного спуска

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

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

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

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

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

SynchrotronEvaluator* synchrotronEvaluator = new
        SynchrotronEvaluator(200, Emin, Emax);
MassiveParticlePowerLawDistribution* electrons = new
        MassiveParticlePowerLawDistribution(massElectron, index,
        Emin, electronConcentration);
SimpleFlatSource* source = new
        SimpleFlatSource(electrons, B, 1.0, R, fraction*R, distance);
```

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

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

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

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

Далее создадим два оптимизатора - действующий перебором и градиентым спуском, и применим их последовательно. Так же укажем количество точек для перебора и то, что оптимизируем все параметры.

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

RadiationOptimizer* enumOptimizer = new GridEnumRadiationOptimizer(
    synchrotronEvaluator,minParameters,maxParameters,Nparams,Npoints);
RadiationOptimizer* gradientOptimizer = new
    GradientDescentRadiationOptimizer(synchrotronEvaluator,
    minParameters, maxParameters, Nparams, Niterations);
```

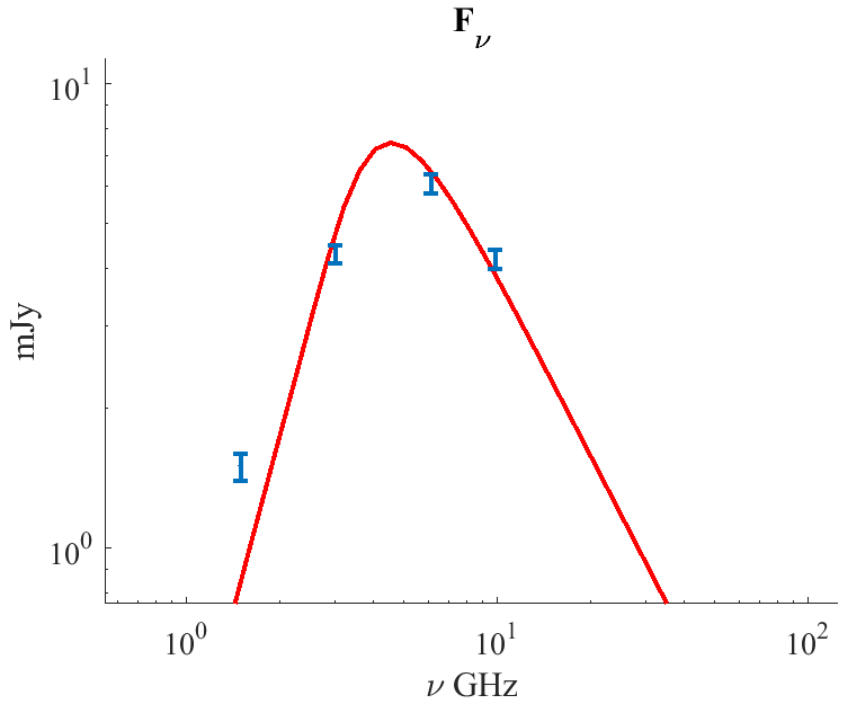

Рисунок 2.2: Наблюдаемый и расчетный спектр радиоизлучения объекта CSS161010 на 98 день после вспышки

Применим функцию optimize у последовательно у обоих оптимизаторов. Сначала перебором найдем начальное приближение, потом уточним результат с помощью градиентного спуска, и изменим параметры источника на оптимальные

```
enumOptimizer->optimize(vector, optPar, energy1, observedFlux,
    observedError, Nenergy1, source);
gradientOptimizer->optimize(vector, optPar, energy1, observedFlux,
    observedError, Nenergy1, source);
source->resetParameters(vector, maxParameters);
```

Полученные в результате оптимизации парметры источника равны: радиус диска $R = 1.8 \times 10^17$ см, магнитное поле $B = 1.6$ Гс, концентрация электронов $n = 2.3$ см$^{-3}$, доля толщины $fraction = 0.54$. Значение целевой функции $f \approx 50$. Модельный спектр излучения с данными параметрами и наблюдательные данные изображены на рисунке 2.2.

2.2 Фитирование источников, зависящих от времени

Для фитирования постоянных во времени кривых блеска изменяющихся во времени предназначен абстрактный класс RadiationTimeOptimizer. В нем определена виртуальныя функция optimize(double* vector, bool* optPar, double** energy, double** observedFLux, double** observedError, int* Ne, int Ntimes, double* times, RadiationTimeDependentSource*

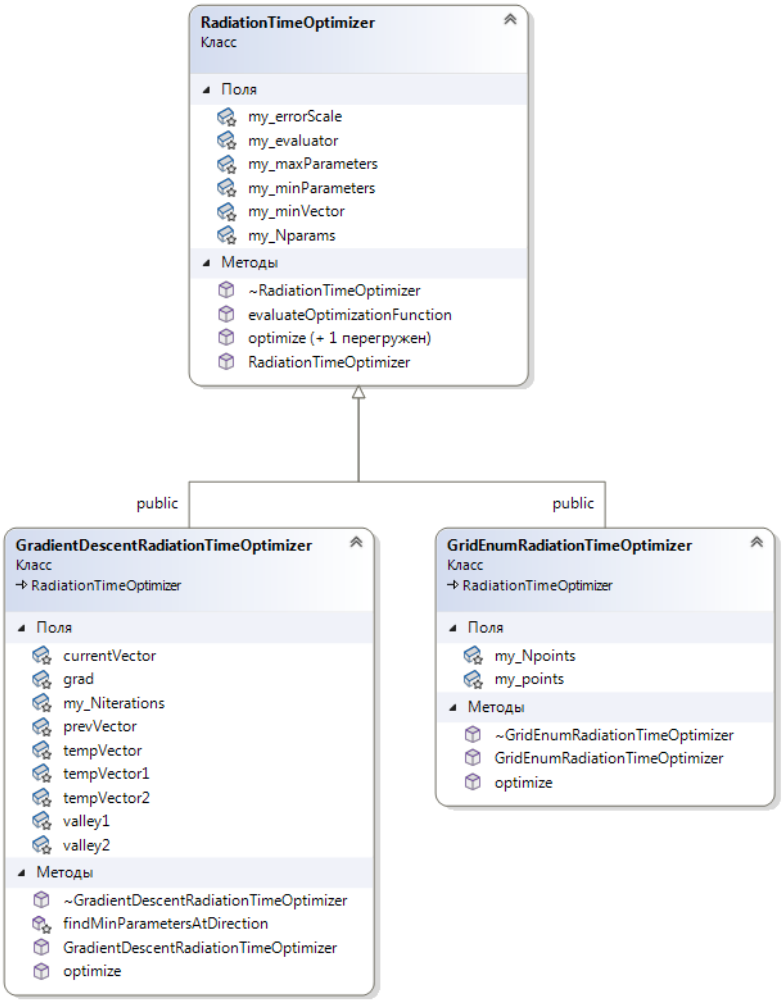

Рисунок 2.3: Схема наследования классов оптимизаторов, учитывающих переменность источников

source), которая и выполняет оптимизацию. Так же как и в случае не зависящей от времени оптимизации она принимает на вход массив параметров, массив булевских переменных, и наблюдательные данные. Но наблюдательные данные теперь представляют собой двумерные массивы (причем количество заданных точек в разные моменты времени так же может быть разным). Так же нужно указать количество серий измерений во времени и соответствующие им времена. Исследуемый источник должен относиться к классу зависящих от времени источников.

Как и ранее, в коде реализованы два наследника класса RadiationTimeOptimazer: GridEnumRadiationTimeOptimizer - для поиска минимума перебором, и GradientDescentRadiationTimeOptimizer - в котором минимум находится методом градиентного спуска. Схема насследования классов оптимизаторов показана на рисунке 2.3, а список их публичных методов приведен в Таблице 2.2.

Таблица 2.2: Публичные методы классов оптимизаторов параметров переменных во времени источников

RadiationTimeOptimizer	абстрактный класс, предназначенный для оптимизации параметров зависящих от времени источников
evaluateOptimizationFunction(const double* vector, double** energy, double** observedFlux, double** observedError, int* Ne, int Ntimes, double* times, RadiationTimeDependentSource* source)	вычисляет целевую функцию - взвешенную сумму квадратов ошибок во всех наблюдательных точках во все моменты времени $f = \sum \frac{(F_i - F_{obs,i})^2}{\sigma_i^2}$, где F_i - расчетная спектральная плотность потока излучения, $F_{obs,i}$ - наблюдаемая спектральная плотность потока излучения, σ_i - её погрешность
optimize (double* vector, bool* optPar, double** energy, double** observedFLux, double** observedError, int* Ne, int Ntimes, double* times, RadiationTimeDependentSource* source)	функция, осуществляющая оптимизацию, принимает на вход массив подбираемых параметров, в который будет записан результат, массив булевских переменных, определяющих оптимизировать соответствующий параметр, или считать его фиксированным, двумерные массивы энергий, на которых производились наблюдения, соответствующих энергиям наблюдаемых потоков в единицах $см^{-2}с^{-1}$, погрешности наблюдаемых потоков, массив количества наблюдательных точек в каждый момент времени, количество моментов времени, когда производились измерения, соответствующие времена и переменный источник излучения.

optimize(double* vector, bool* optPar, double** energy, double** observedFlux, int* Ne, int Ntimes, double* times, RadiationTimeDependentSource* source)	функция, осуществляющая оптимизацию, в случае не заданных наблюдательных ошибок. В таком случае ошибки у всех точек считаются равными елинице и веса всех ошибок в целевой функции оказываются равными
GridEnumRadiationTimeOptimizer	класс, предназначенный для оптимизации параметров переменных источников с помощью перебора по сетке
GridEnumRadiationTimeOptimizer(RadiationEvaluator* evaluator, const double* minParameters, const double* maxParameters, int Nparams, const int* Npoints)	конструктор, создает экземпляр класса с указанным вычислителем излучения, минимальными и максимальными значениями оптимизируемых параметров, количеством этих параметров и массивом с количеством перебираемых точек по каждому параметру. При переборе точки будут распределены логарифмически равномерно.
GradientDescentRadiationTimeOptimizer	класс, предназначенный для оптимизации параметров переменных источников методом градиентного спуска
GradientDescentRadiationTimeOptimizer(RadiationEvaluator* evaluator, const double* minParameters, const double* maxParameters, int Nparams, int Niterations)	конструктор, создает экземпляр класса с указанным вычислителем излучения, минимальными и максимальными значеними оптимизируемых параметров, количеством этих параметров и максимальным количеством итераций градиентного спуска

Пример фитирования параметров источника по наблюдательным данным приведен в функции fitTimeDependentCSS161010() в файле examples.cpp. Подберем параметры Быстрого Оптического Голубого Транзиента CSS161010 на основе наблюдений радиоизлучения, проведенных на 98, 162, 357 день после вспышки. Расчет синхротронного излучения учитывает самопоглощение и раширение источника. Источник будем считать шаровым слоем с однородной плотностью и однороным магнитным полем, направленным перпендикулярно лучу зрения. Функцию распределения излучающих электронов возьмем на основе Particle-in-Cell расчетов для ударной волны со скоростью 0.3c,как сделано в работе [**?**]. Учтена зависимость функции распределения от угла между магнитным полем и направлением распространения ударной волны.

Подберем параметры Быстрого Оптического Голубого Транзиента CSS161010 на основе наблюдений радиоизлучения, проведенных на 98, 162, 357 день после вспышки. Зададим сначала массивы наблюдательных точек, переведя при этом из единиц герцы и милиянские в эрги и $см^{-2}с^{-1}$

```
const double cssx1[4] =
```

```
{1.5*hplank*1E9, 3.0*hplank*1E9, 6.1*hplank*1E9, 9.87*hplank*1E9};
const double cssy1[4] = {1.5/(hplank*1E26),
4.3/(hplank*1E26), 6.1/(hplank*1E26), 4.2/(hplank*1E26)};
const double cssError1[4] = {0.1/(hplank*1E26),
0.2/(hplank*1E26), 0.3/(hplank*1E26), 0.2/(hplank*1E26)};

const double cssx2[4] =
{2.94*hplank*1E9, 6.1*hplank*1E9, 9.74*hplank*1E9, 22.0*hplank*1E9};
const double cssy2[4] = {2.9/(hplank*1E26),
2.3/(hplank*1E26), 1.74/(hplank*1E26), 0.56/(hplank*1E26)};
const double cssError2[4] = {0.2/(hplank*1E26),
0.1/(hplank*1E26), 0.09/(hplank*1E26), 0.03/(hplank*1E26)};

const double cssx3[6] = {0.33*hplank*1E9, 0.61*hplank*1E9,
1.5*hplank*1E9, 3.0*hplank*1E9, 6.05*hplank*1E9, 10.0*hplank*1E9};
const double cssy3[6] = {0.375/(hplank*1E26), 0.79/(hplank*1E26),
0.27/(hplank*1E26),0.17/(hplank*1E26),0.07/(hplank*1E26),0.32/(hplank*1E27)};
const double cssError3[6] = {0.375/(hplank*1E26), 0.09/(hplank*1E26),
0.07/(hplank*1E26),0.03/(hplank*1E26),0.01/(hplank*1E26),0.8/(hplank * 1E28)}
```

Определим моменты времени инаблюдений и соответствующие им количества точек

```
const int Ntimes = 3;
double times[Ntimes] = { 99 * 24 * 3600, 162 * 24 * 3600, 357 * 24 * 3600 };
int Nenergy[Ntimes];
Nenergy[0] = 4;
Nenergy[1] = 4;
Nenergy[2] = 6;
```

Создадим и инициализируем необходимые массивы с наблюдательными данными

```
double** energy = new double* [Ntimes];
double** F = new double* [Ntimes];
double** Error = new double* [Ntimes];
for (int m = 0; m < Ntimes; ++m) {
        energy[m] = new double[Nenergy[m]];
        F[m] = new double[Nenergy[m]];
        Error[m] = new double[Nenergy[m]];
}

for (int i = 0; i < Nenergy[0]; ++i) {
        energy[0][i] = cssx1[i];
```

```
        F[0][i] = cssy1[i];
        Error[0][i] = cssError1[i];
}

for (int i = 0; i < Nenergy[1]; ++i) {
        energy[1][i] = cssx2[i];
        F[1][i] = cssy2[i];
        Error[1][i] = cssError2[i];
}

for (int i = 0; i < Nenergy[2]; ++i) {
        energy[2][i] = cssx3[i];
        F[2][i] = cssy3[i];
        Error[2][i] = cssError3[i];
}
```

Зададим физические параметры источника (или их начальные приближения) - расстояние, размер, концентрацию, магнитное поле, долю толщины шара, занятую излучающим веществом, скорость расширения и магнетизацию.

```
const double distance = 150 * 1E6 * parsec;
double rmax = 1.3E17;
double electronConcentration = 150;
double B = 0.6;
double widthFraction = 0.5;
double v = 0.3 * speed_of_light;
double sigma = B * B / (4 * pi * massProton *
electronConcentration * speed_of_light2);
```

Укажем для оптимизаторов количество параметров, ихи минимальные и максимальные значения и соответствие вектора параметров и физических величин. Оптимизируемыми параметрами являются - размер источника, магнетизация, доля заполнения и скорость расширения в первый момент времени, а так же показатели степени расширения со временем и изменения магнитного поля и концентрации с радиусом, то есть α, β, γ где эти величины определены через уравнения $R(t) = R_0 + \frac{1}{\alpha-1} \cdot V(0) \cdot t_0 \cdot (t/{t_0}^{\alpha-1} - 1)$, $B(R) = B(R_0) \cdot R_0/R^{\beta-1}$, $n(R) = n(R_0) \cdot R_0/R^{\gamma-1}$. Единица добавлена к показателям степени для удобства численных расчетов при близости величин к нулю.

```
const int Nparams = 8;
double minParameters[Nparams] = { 1E16, 0.0001, 0.01, 0.1,
0.01 * speed_of_light, 1.1, 1.0, 1.0 };
double maxParameters[Nparams] = { 2E17, 1, 1000, 1.0, 0.6 *
```

```
speed_of_light, 2.0, 3.5, 3.5 };
double vector[Nparams] = { rmax, sigma, electronConcentration,
widthFraction, v, 2.0, 2.0, 3.0 };
for (int i = 0; i < Nparams; ++i) {
    vector[i] = vector[i] / maxParameters[i];
}
bool optPar[Nparams] = { true, true, true, true, true, true, true, true };
```

Далее создадим источник излучения. Воспользуемся моделью расширяющейся однородной сферической оболочки, с однородным магнитным полем, перпендикулярным лучу зрения и функцией распределения электронов, зависящей от угла между направлением магнитного поля и направлением расширения оболочки. Функции распределения получены с использованием Particle-in-Cell кода Smilei [**?**] и содержатся в директории examplesData. Методика расчетов описана в статье [**?**]. Количетсво распределений, посчитанных для углов от 0 до 90 градусов равно десяти. Их можно считать из соответствующих файлов, используя метод класса MassiveParticleDistributionFactory. Так же будет добавлено продолжение мтепенного хвоста, так как PIC расчеты не пользволяют получать длинные спектры из-за большой вычислительной сложности. Так же необходимо провести масштабирование распределения, так как в PIC расчетах испольовалось уменьшенное отношение масс протонов и электронов $m_p/m_e = 100$. Имея массив распределений создадим источник, учитывающий угловую зависимость, и передим его далее источнику. учитывающему зависимость от времени.

```
const int Ndistributions = 10;

MassiveParticleIsotropicDistribution** angleDependentDistributions =
MassiveParticleDistributionFactory::readTabulatedIsotropicDistributionsAddPow
(massElectron, "./input/Ee", "./input/Fs", ".dat", 10,
DistributionInputType::GAMMA_KIN_FGAMMA, electronConcentration, 200, 20 * me_
for (int i = 0; i < Ndistributions; ++i) {
(dynamic_cast<MassiveParticleTabulatedIsotropicDistribution*>
(angleDependentDistributions[i]))->rescaleDistribution(sqrt(18));
}

AngleDependentElectronsSphericalSource* angleDependentSource = new
AngleDependentElectronsSphericalSource(20, 20, 4, Ndistributions,
angleDependentDistributions,B,1.0,0,electronConcentration,rmax,0.5*rmax,dista

RadiationTimeDependentSource* source = new
ExpandingRemnantSource(rmax, B, electronConcentration, 0.3 * speed_of_light,
```

Рисунок 2.4: Наблюдаемый и расчетный спектр радиоизлучения объекта CSS161010 на 99, 162 и 357 дни после вспышки

```
0.5, angleDependentSource, times[0]);
```

Теперь создадим вычислитель синхротронного излучения и два оптимизатора параметров - первый будет работать перебором параметров по сетке, а второй - градиентным спуском. Укажем количество точек по осям для перебора, количество итераций для градиентного спуска и диапазон энергий электронов, который будет рассматривать вычислитель синхротронного излучения.

```
int Npoints[Nparams] = { 3, 3, 3, 3, 3, 3, 3, 3 };
int Niterations = 5;

double Emin = me_c2;
double Emax = 10000 * me_c2;

SynchrotronEvaluator* synchrotronEvaluator=new SynchrotronEvaluator(200, Emin

RadiationTimeOptimizer* gridEnumOptimizer =
new GridEnumRadiationTimeOptimizer(synchrotronEvaluator, minParameters,
maxParameters, Nparams, Npoints);
RadiationTimeOptimizer* gradientOptimizer =
new GradientDescentRadiationTimeOptimizer(synchrotronEvaluator,minParameters,
maxParameters, Nparams, Niterations);
```

Применим созданые оптимизаторы и изменим параметры источника на найденные, соответствующие минимуму.

```
gridEnumOptimizer->optimize(vector, optPar, energy, F, Error, Nenergy, Ntimes

gradientOptimizer->optimize(vector, optPar, energy, F, Error, Nenergy, Ntimes

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

Полученные в результате оптимизации парметры источника равны: радиус диска в начальный момент времени $R = 1.8 \times 10^17$ см, магнитное поле $B = 1.6$ Гс, концентрация электронов $n = 2.3$ см$^{-3}$, доля толщины $fraction = 0.54$, степени зависимости . Значение целевой функции $f \approx 50$. Модельный спектр излучения с данными параметрами и наблюдательные данные изображены на рисунке 2.4.

Глава 3

Формулы расчета излучения

3.1 Преобразование функции распределения фотонов

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

$$N = n_{ph}(\epsilon, \mu, \phi) d\epsilon d\mu d\phi dV = n'_{ph}(\epsilon', \mu', \phi') d\epsilon' d\mu' d\phi' dV' \tag{3.1}$$

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

$$\begin{pmatrix} \epsilon' \\ \mu'\epsilon' \end{pmatrix} = \begin{pmatrix} \gamma & -\beta\gamma \\ -\beta\gamma & \gamma \end{pmatrix} \times \begin{pmatrix} \epsilon \\ \mu\epsilon \end{pmatrix} \tag{3.2}$$

Из первой строчки матрицы получаем уравнение для допплеровского сдвига энергии

$$\epsilon' = \gamma(1 - \mu\beta)\epsilon \tag{3.3}$$

Вычислим производные новой энергии по старым координатам

$$\frac{d\epsilon'}{d\epsilon} = \gamma(1 - \mu\beta) \tag{3.4}$$

$$\frac{d\epsilon'}{d\mu} = -\gamma\beta\epsilon \tag{3.5}$$

Из второй строчки матрицы получаем $\mu'\epsilon' = -\beta\gamma\epsilon + \gamma\mu\epsilon$. Подставив значение ϵ' из 3.3 и сократив ϵ получим уравнение аберрации света

$$\mu' = \frac{\mu - \beta}{1 - \mu\beta} \tag{3.6}$$

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

$$\frac{d\mu'}{d\mu} = \frac{d}{d\mu}\frac{1}{\beta}\frac{\beta\mu - 1 + 1 - \beta^2}{1 - \mu\beta} = \frac{d}{d\mu}\frac{1}{\beta}\frac{1-\beta^2}{1-\mu\beta} = \frac{1-\beta^2}{(1-\mu\beta)^2} = \frac{1}{\gamma^2(1-\mu\beta)^2} \tag{3.7}$$

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

$$\frac{dV'}{dV} = \frac{1}{\gamma(1 - \mu\beta)} \tag{3.8}$$

Матрица якоби преобразования координат выглядит следующим образом

$$J = \begin{pmatrix} \frac{d\epsilon'}{d\epsilon} & \frac{d\epsilon'}{d\mu} & 0 & 0 \\ 0 & \frac{d\mu'}{d\mu} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \frac{dV'}{d\mu} & 0 & \frac{dV'}{dV} \end{pmatrix} \tag{3.9}$$

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

$$\frac{D(\epsilon', \mu', \phi', V')}{D(\epsilon, \mu, \phi, V)} = \frac{d\epsilon'}{d\epsilon}\frac{d\mu'}{d\mu}\frac{dV'}{dV} = \gamma(1-\mu\beta)\frac{1}{\gamma^2(1-\mu\beta)^2}\frac{1}{\gamma(1-\mu\beta)} = \frac{1}{\gamma^2(1-\mu\beta)^2} \tag{3.10}$$

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

$$n'_{ph}(\epsilon', \mu', \phi') = \frac{n_{ph}(\epsilon, \mu, \phi)}{\frac{D(\epsilon', \mu', \phi', V')}{D(\epsilon, \mu, \phi, V)}} = \gamma^2(1-\mu\beta)^2 n_{ph}(\epsilon, \mu, \phi) \tag{3.11}$$

3.2 Комптоновское рассеяние

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

$$\frac{d\sigma}{d\epsilon'_1 d\Omega'_1} = \frac{{r_e}^2}{2}\left(\frac{\epsilon'_1}{\epsilon'_0}\right)^2 \left(\frac{\epsilon'_1}{\epsilon'_0} + \frac{\epsilon'_0}{\epsilon'_1} - \sin^2\Theta'\right)\delta\left(\epsilon'_1 - \frac{\epsilon'_0}{1 + \frac{\epsilon'_0}{m_e c^2}(1-\cos\Theta')}\right) \tag{3.12}$$

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

$$\epsilon'_1 = \frac{\epsilon'_0}{1 + \frac{\epsilon'_0}{m_e c^2}(1-\cos\Theta')} \tag{3.13}$$

$$\epsilon'_0 = \frac{\epsilon'_1}{1 - \frac{\epsilon'_1}{m_e c^2}(1-\cos\Theta')} \tag{3.14}$$

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

$$\frac{dN'}{dt' d\epsilon'_1 d\Omega'_1} = \int c\frac{d\sigma}{d\epsilon'_1 d\Omega'_1}\frac{dn'}{d\epsilon'_0 d\Omega'_0} d\Omega'_0 d\epsilon'_0 \tag{3.15}$$

Перепишем дельта-функцию через энергию начального фотона с помощью соотношения

$$\delta(f(x)) = \sum \frac{\delta(x - x_k)}{|f'(x_k)|} \tag{3.16}$$

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

$$\frac{d\epsilon_1'}{d\epsilon_0'} = \frac{1}{(1+\frac{\epsilon_0'}{m_e c^2}(1-\cos\Theta'))^2} \tag{3.17}$$

и она сократится с квадратом отношения энергий в формуле для сечения. Функцию распределения начальных фотонов выразим в лабораторной системе с помощью выражения 3.11.

$$\frac{dN'}{dt' d\epsilon_1' d\Omega_1'} = \int \frac{r_e^2 c}{2}\gamma_e^2(1-\mu_0\beta_e)^2(\frac{\epsilon_1'}{\epsilon_0'}+\frac{\epsilon_0'}{\epsilon_1'}-\sin^2\Theta')\frac{dn}{d\epsilon_0 d\Omega_0}\delta(\epsilon_0' - \frac{\epsilon_1'}{1-\frac{\epsilon_1'}{m_e c^2}(1-\cos\Theta')})d\epsilon_0' d\mu_0' d\phi_0' \tag{3.18}$$

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

$$\frac{dN'}{dt' d\epsilon_1' d\Omega_1'} = \int \frac{r_e^2 c}{2}\gamma_e^2(1-\mu_0\beta_e)^2(1+\cos^2\Theta' + (\frac{\epsilon_1'}{m_e c^2})^2\frac{(1-\cos\Theta')^2}{1-\frac{\epsilon_1'}{m_e c^2}(1-\cos\Theta')})\frac{dn}{d\epsilon_0 d\Omega_0}d\mu_0' d\phi_0' \tag{3.19}$$

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

$$\frac{dN}{dt d\epsilon_1 d\Omega_1} = \int \frac{r_e^2 c}{2}\frac{(1-\mu_0\beta_e)^2}{1-\mu_1\beta_e}(1+\cos^2\Theta' + (\frac{\epsilon_1'}{m_e c^2})^2\frac{(1-\cos\Theta')^2}{1-\frac{\epsilon_1'}{m_e c^2}(1-\cos\Theta')})\frac{dn}{d\epsilon_0 d\Omega_0}d\mu_0' d\phi_0' \tag{3.20}$$

При интегрировании нужно выразить углы в лабораторной системе отсчета μ_0, ϕ_0 через переменные интегрирования μ_0', ϕ_0'. Для расчета рассеяния на распределении электронов нужно проинтегрировать формулу 3.20 с функцией распределения электронов, нормированной на количество частиц. При этом надо учесть разные направления движения электронов и произвести повороты углов.

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

$$\frac{dN}{dt d\epsilon_1 d\Omega_1} = \int \frac{r_e^2 c}{2}\frac{1}{\gamma_e^2(1-\mu_1\beta_e)}(1+\cos^2\Theta' + (\frac{\epsilon_1'}{m_e c^2})^2\frac{(1-\cos\Theta')^2}{1-\frac{\epsilon_1'}{m_e c^2}(1-\cos\Theta')})\frac{dn}{d\epsilon_0 d\Omega_0}d\mu_0 d\phi_0 \tag{3.21}$$

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

$$\delta_e = 1 - \beta_e \tag{3.22}$$

$$\text{versin}\ \theta = 1 - \cos\theta \tag{3.23}$$

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

$$1 - \mu\beta_e = \text{versin}\ \theta + \delta_e - \text{versin}\ \theta\ \delta_e \tag{3.24}$$

а выражение для угла между конечным и начальным фотоном как

$$1 - \cos\Theta' = \text{versin } \theta'_0 + \text{versin } \theta'_1 - \text{versin } \theta'_0 \text{versin } \theta'_1 - \sin\theta'_0 \sin\theta'_1 \cos(\phi'_1 - \phi'_0) \qquad (3.25)$$

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

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

$$\frac{dN}{dtd\epsilon_1 d\Omega_1} = \int \frac{2\pi r_e^2 m_e c^3}{\epsilon_0 \gamma_e^2} \frac{dn_{ph}}{d\epsilon_0} \frac{dn_e}{d\epsilon_e} (2q \ \ln(q) + 1 + q - 2q^2 + \frac{q^2(1-q)\Gamma^2}{2(1+q\Gamma)}) d\epsilon_0 d\epsilon_e \qquad (3.26)$$

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

3.3 Синхротронное излучение

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

$$I(\nu) = \int_{E_{min}}^{E_{max}} dE \frac{\sqrt{3} e^3 n F(E) B \sin(\phi)}{m_e c^2} \frac{\nu}{\nu_c} \int_{\frac{\nu}{\nu_c}}^{\infty} K_{5/3}(x) dx, \qquad (3.27)$$

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

$$k(\nu) = \int_{E_{min}}^{E_{max}} dE \frac{\sqrt{3} e^3}{8\pi m_e \nu^2} \frac{nB\sin(\phi)}{E^2} \frac{d}{dE} E^2 F(E) \frac{\nu}{\nu_c} \int_{\frac{\nu}{\nu_c}}^{\infty} K_{5/3}(x) dx. \qquad (3.28)$$

Литература

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Литература

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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