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 main.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

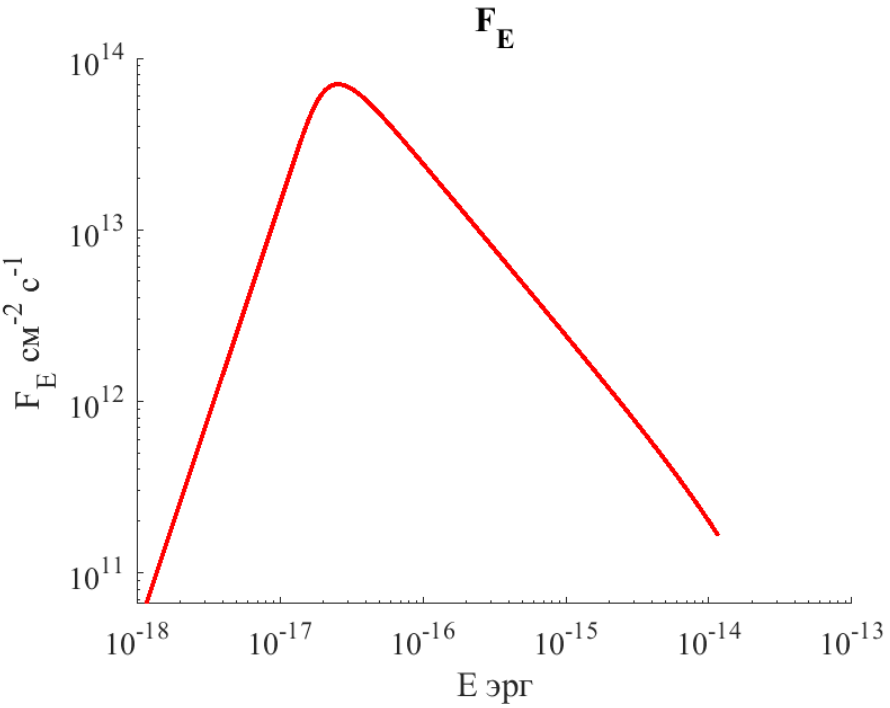

Рисунок 1: Энергетическая плотность потока синхротронного излучения от тестового источника

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 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

Таблица 1.1: Публичные методы класса ParticleDistribution

ParticleDistribution	
distribution(const double& energy, const double& mu, const double& phi)	возвращает функцию распределения от энергии, косинуса полярного угла и азимутального угла, нормированную на единицу
distributionNormalized(const double& energy, const double& mu, const double& phi)	возвращает функцию распределения от энергии, косинуса полярного угла и азимутального угла, нормированную на концентрацию
getConcentration()	возвращает концентрацию частиц

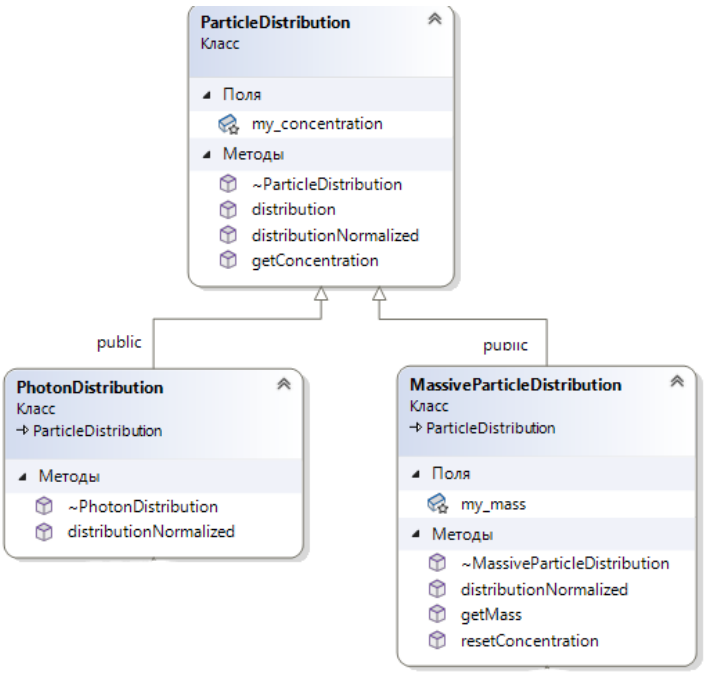

Рисунок 1.1: Схема наследования распределения фотонов и массивных частиц

Таблица 1.2: Публичные методы класса MassiveParticleDistribution

MassiveParticleDistribution	
getMass()	возвращает массу частиц
resetConcentration(const double& n)	позволяет изменить полную концентрацию частиц в распределении

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 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 функ-

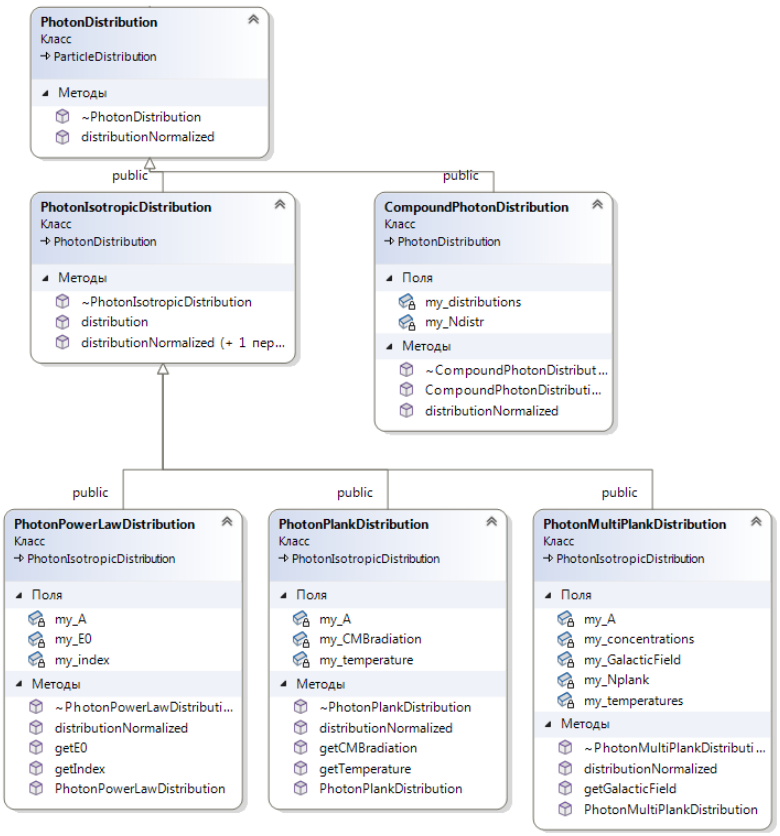

Рисунок 1.2: Схема наследования классов распределений фотонов

ции распределения по энергии нужно домножить значение, возвращенное данным методом на 4π.

У класса PhotonIsotropicDistribution есть три наследника, которые уже не абстрактные классы, а непосредственно предназначены для создания распределений. Это PhotonPowerLawDistribution для представления степенных распределений, PhotonPlankDistribution, для планковских распределений и PhotonMultiPlankDistribution, для суммы планковских распределений. Метода класса PhotonIsotropicDistribution и его наследников перечислены в таблице 1.3

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

Встроенных анизотропных распределений фотонов в коде на данный момент нет, но пользователь может реализовать их самостоятельно, создав класс, наследующий от PhotonDistribution и определив необходимый виртуальный метод distributionNormalized(const double& energy, const double& mu, const double& phi). Аналогично можно, конечно, создать и другие виды изотропных распределений.

Таблица 1.3: Публичные методы классов изотропных распределений фотонов

PhotonIsotropicDistribution	
distribution(const double& energy)	возвращает функцию распределения с отброшенными угловыми аргументами, то есть нормированную на концентрацию, деленную на 4π
distributionNormalized(const double& energy)	возвращает функцию распределения с отброшенными угловыми аргументами, нормированную на $1/4\pi$
PhotonPowerLawDistribution	
PhotonPowerLawDistribution(const double& index, const double& E0, const double& concentration)	конструктор, создающий экземпляр с заданными показателем наклона, начальной энергией и полной концентрацией
getIndex()	возвращает показатель наклона спектра
getE0()	возвращает минимальную энергию степенного распределения
PhotonPlankDistribution	
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 getCMBRadiation()	статический метод, возвращающий экземпляр, соответствующий реликтовому излучению (температура $2.725K$, амплитуда 1)
getTemperature()	возвращает температуру распределения
PhotonMultiPlankDistribution	
PhotonMultiPlankDistribution(int Nplank, const double* const temperatures, const double* const amplitudes)	конструктор, количество планковских распределений, участвующих в смеси, массив их температур и массив амплитуд
static 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

Таблица 1.4: Публичные методы класса CompoundPhotonDistribution

CompoundPhotonDistribution	
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

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. Так же добавлен метод записи функции распределения в файл.

Таблица 1.5: Публичные методы класса MassiveParticleIsotropicDistribution

MassiveParticleIsotropicDistribution	
distribution(const double& energy)	возвращает функцию распределения с отброшенными угловыми аргументами, то есть нормированную на концентрацию, деленную на 4π
distributionNormalized(const double& energy)	возвращает функцию распределения с отброшенными угловыми аргументами, нормированную на $1/4\pi$
writeDistribution(const char* fileName, int Ne, const double& Emin, const double& Emax)	записывает распределение в файл с данным именем, в диапазоне межджу 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 шесть 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 и MassiveParticleTabulatedIsotropicDistribution - для таблично заданных распределений.

Так же имеется четыре реализации 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, 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 1.6. 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 создать любым удобным ему способом.

Таблица 1.6: Публичные методы классов распределений массивных частиц

MassiveParticlePowerLawDistribution	
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

getIndex()	возвращает степенной индекс распределения
getE0()	возвращает начальную энергию распределения
MassiveParticleBrokenPowerLawDistribution	
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
getIndex1()	возвращает степенной индекс распределения на низких энергиях
getIndex2()	возвращает степенной индекс распределения на высоких энергиях
getE0()	возвращает начальную энергию распределения
getEtran()	возвращает энергию излома
MassiveParticlePowerLawCutoffDistribution	
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. $F(E) \propto (E/E_0)^{-index} \cdot \exp(-(E/E_{cut})^{\beta})$
getIndex()	возвращает степенной индекс распределения
getBeta()	возвращает параметр завала распределения
getE0()	возвращает начальную энергию распределения
getEcutoff()	возвращает энергию экспоненциального завала
MassiveParticleMaxwellDistribution	
MassiveParticleMaxwellDistribution(const double& mass, const double& temperature, const double& concentration)	конструктор, создает экземпляр распределния Максвелла частиц с заданными массой, температурой и полной концентрацией
getTemperature()	возвращает температуру распределения
MassiveParticleMaxwellJuttnerDistribution	
MassiveParticleMaxwellJuttnerDistribution(const double& mass, const double& temperature, const double& concentration)	конструктор, создает экземпляр распределния Максвелла-Юттнера частиц с заданными массой, температурой и полной концентрацией

getTemperature()	возвращает температуру распределения
MassiveParticleTabulatedIsotropicDistribution	
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.
getN()	возвращает количество ячеек в таблице задающей функцию
getEmin()	возвращает минимальную энергию распределения
getEmax()	возвращает максимальную энергию распределения
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

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
MassiveParticleTabulatedPolarDistribution	
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.
getNe()	возвращает количество ячеек по энергии в таблице задающей функцию распределения
getEmin()	возвращает минимальную энергию распределения
getEmax()	возвращает максимальную энергию распределения
getNmu()	возвращает количество ячеек по полярному углу в таблице задающей функцию распределения
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	
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.
getNe()	возвращает количество ячеек по энергии в таблице задающей функцию распределения
getEmin()	возвращает минимальную энергию распределения
getEmax()	возвращает максимальную энергию распределения
getNmu()	возвращает количество ячеек по полярному углу в таблице задающей функцию распределения
getNphi()	возвращает количество ячеек по азимутальному углу в таблице задающей функцию распределения
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

CompoundMassiveParticleDistribution	
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	
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

Глава 2

Оптимизация параметров

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 $\frac{d\mu'}{d\mu}$

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

Азиметальный 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. Сечение Клейна-Нишины в системе покоя электрона равно

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

Где r_e - классический радиус электрона, ϵ'_0 и ϵ'_1 - энергии начального и конечного фотона, соответственно, Θ' - угол между начальным и конечным фотоном. Штрихованные индексы относятся к системе отсчета электрона. Число фотонов,

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 [2]. Спектральная плотность мощности излучения единицы объема вещества определеяется формулой

$$I(\nu) = \int_{E_{min}}^{E_{max}} dE \frac{\sqrt{3}e^3 nF(E)B\sin(\phi)}{m_e c^2}\frac{\nu}{\nu_c}\int_{\frac{\nu}{\nu_c}}^{\infty} K_{5/3}(x)dx, \tag{3.13}$$

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

$$k(\nu) = \int_{E_{min}}^{E_{max}} dE \frac{\sqrt{3}e^3}{8\pi m_e \nu^2}\frac{nB\sin(\phi)}{E^2}\frac{d}{dE}E^2 F(E)\frac{\nu}{\nu_c}\int_{\frac{\nu}{\nu_c}}^{\infty} K_{5/3}(x)dx. \tag{3.14}$$

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