Второй обязательной частью реализации математической модели посредством Geant4 является построение списка используемых физических процессов. Данный аспект модели отвечает на вопрос: «По каким законам осуществляется моделирование?».

В Geant4 представлен огромный набор первичных частиц и процессов взаимодействия между ними и веществом. В основе реализации лежит класс G4VUserPhysicsList, однако сразу стоит отметить его расширение G4VModularPhysicsList. В случае расширения все используемые процессы группируются в модели, а из них, в свою очередь, формируются списки физических процессов.

Вне зависимости от выбранного способа конечный объект класса-списка должен быть передан ядру за счет метода:

```
virtual void SetUserInitialization(G4VUserPhysicsList* userPL);
```

Готовые списки физических процессов

Начать рассмотрение данного раздела было бы уместно с укомплектованных разработчиками Geant4 списков физических процессов. Дело в том, что задачи кафедры «Прикладная ядерная физика» редко выходят за границу областей интересов:

- нейтронов (от тепловых, до быстрых с энергией порядка 20 МэВ),
- а также альфа, бета и гамма источников.

Потребности большинства задач с лихвой покрываются следующими моделями:

QBBC — включает полный набор моделей для альфа-, бета-, гамма-излучений, содержит процессы связанные с упругим/неупругим взаимодействием, тормозным излучением и многое другое. Однако данная модель не применима для нейтронного взаимодействия.

*_HP — любые модели, содержащие в конце своего названия данное сокращение. К примеру, QGSP_BERT_HP или QGSP_BIC_HP. Сокращение HP в данном случае означает, что данные списки содержат высокоточные модели для нейтронов низких энергий. Различия же в таких моделях чаще всего

начинаются в высокоэнергетических диапазонах, и выходят за рамки задач кафедры.

Полное описание содержимого данных моделей во много раз превысит объем данного пособия, поэтому приведено не будет. Однако с ним можно ознакомиться на официальном сайте в разделах D и F (Physics Reference Manual и Physics List Guide) соответственно.

Для использования данных списков необходимо и достаточно передать объекты их классов ядру Geant4:

```
auto runManager = new G4MTRunManager;

runManager->SetUserInitialization(new Geometry());
runManager->SetUserInitialization(new QBBC);          // использование QBBC
runManager->SetUserInitialization(new Action());
```

Так же стоит упомянуть, что в основе нейтронной модели лежат данные по сечениям, базирующиеся на ENDF/B-VII.r1.

Сборка физического списка

Иногда возникают задачи, для решения которых стандартных списков физических процессов оказывается недостаточно. Рассмотрим основные аспекты построения пользовательского списка физических процессов. В качестве базового класса удобнее будет использовать G4VModularPhysicsList, особенно, если имеется необходимость переписывать не весь список, а только его часть.

Данный класс содержит метод

```
void RegisterPhysics(G4VPhysicsConstructor* );
```

в качестве аргумента у которого используется модель физических процессов.

Оригинальный QBBC, основанный на G4VModularPhysicsList выглядит, к примеру, следующим образом:

```
QBBC::QBBC( G4int ver, const G4String&)
{
 G4DataQuestionaire it(photon, neutronxs);
 G4cout << "<<< Reference Physics List QBBC " << G4endl;

 defaultCutValue = 0.7*mm;
 SetVerboseLevel(ver);

 // EM Physics
 RegisterPhysics( new G4EmStandardPhysics(ver) );

 // Synchroton Radiation & GN Physics
 RegisterPhysics( new G4EmExtraPhysics(ver) );

 // Decays
 RegisterPhysics( new G4DecayPhysics(ver) );

 // Hadron Physics
 RegisterPhysics( new G4HadronElasticPhysicsXS(ver) );

 RegisterPhysics( new G4StoppingPhysics(ver) );

 RegisterPhysics( new G4IonPhysics(ver) );

 RegisterPhysics( new G4HadronInelasticQBBC(ver));
```

где G4int - параметр ver отвечает за «уровень» подробности выводимой в консоль информации. Уровень 0 — минимальный, 1 (реже 2) — максимальный.

Перейдем к рассмотрению класса — модели G4VPhysicsConstructor. Данный класс содержит два чисто виртуальных метода:

```
virtual void ConstructParticle()=0;

virtual void ConstructProcess()=0;
```

ConstructParticle() отвечает за первичную инициализацию частиц. Следовательно, абсолютно все частицы, используемые в процессах, объявленных в данной модели, должны быть инициализированы в этом методе. Для инициализации частиц возможно два пути. Для стандартных частиц (гамма, электрон, нейтрон, протон и т. п.) можно воспользоваться статическими методами:

```
G4Gamma::GammaDefinition();
G4Neutron::NeutronDefinition();
G4Electron::ElectronDefinition();
G4Proton::ProtonDefinition();
```

Однако, в случае наличия таких частиц как ионы, барионы и т. п. используются статические методы ConstructParticle(), позволяющие сконструировать все частицы своей группы:

```
G4IonConstructor::ConstructParticle();
G4MesonConstructor::ConstructParticle();
G4ShortLivedConstructor::ConstructParticle();
```

Посредством ConstructProcess() осуществляется подключение всех используемых процессов к модели. В данном случае удобнее будет рассмотреть его реализацию на примере.

Пример использования метода ConstuctProcess()

Реализуем подключение High Precision Models (HP) для нейтронов низких энергий (до 20 МэВ), а именно, подключение процессов упругого и неупругого взаимодействия нейтронов.

За упругое взаимодействие отвечает

```
G4HadronElasticProcess(const G4String& procName = "hadElastic");
```

За неупругое

```
G4NeutronInelasticProcess(const G4String& processName = "neutronInelastic");
```

Сразу стоит отметить, что рассматриваемые выше процессы наследуют класс G4VDiscreteProcess, произошедший в свою очередь от G4VProcess — базового класса всех процессов Geant4.

Первое, что нужно сделать в случае реализации выше рассматриваемых процессов, это указать используемые наборы данных сечений (наследуют G4VCrossSectionDataSet).

Будем использовать два различных набора сечений: один для диапазона 4эВ — 20МэВ, второй для тепловых нейтронов вплоть до энергии 4эВ:

```
auto theNeutronElasticProcess = new G4HadronElasticProcess;

theNeutronElasticProcess->AddDataSet(new G4ParticleHPElasticData());
theNeutronElasticProcess->AddDataSet(new G4ParticleHPThermalScatteringData);
```

Далее следует указать специальные модели для процесса (т. к. оригинальный процесс рассчитан на весь диапазон энергий (т. е. до ТэВ))

Для этого предусмотрен метод:

```
void RegisterMe(G4HadronicInteraction* a);
```

где в качестве аргумента принимается специальная модель (только для адронов)

```
auto theNeutronElasticModel = new G4ParticleHPElastic;
theNeutronElasticModel->SetMinEnergy(4.0*eV);
theNeutronElasticProcess->RegisterMe(theNeutronElasticModel);

theNeutronThermalElasticModel = new G4ParticleHPThermalScattering;
theNeutronThermalElasticModel->SetMaxEnergy(4.0*eV);
theNeutronElasticProcess->RegisterMe(theNeutronThermalElasticModel);
```

в данном случае методы:

```
inline void SetMinEnergy( G4double anEnergy );
inline void SetMaxEnergy( const G4double anEnergy );
```

отвечают за диапазон действия той или иной модели в процессе.

После того как настройки процесса закончены, его нужно передать соответствующему типу частиц. В данном примере это будет выглядеть как:

```
G4ProcessManager* pmanager = G4Neutron::Neutron()->GetProcessManager();
pmanager->AddDiscreteProcess(theNeutronElasticProcess);
```

Объект класса G4ProcessManager является классом - контейнером, содержащим все используемые в программе процессы. У каждого типа частиц существует свой контейнер. Для подключения дискретного процесса в демонстрируемом примере используется метод:

```
G4int G4ProcessManager::AddDiscreteProcess(G4VProcess *aProcess);
```

Что касается неупругого взаимодействия, то ограничимся одной моделью на весь исследуемый диапазон энергии, т. е.:

```
auto theNeutronInelasticProcess = new G4NeutronInelasticProcess();

theNeutronInelasticProcess->AddDataSet(new G4ParticleHPInelasticData());

auto theNeutronInelasticModel = new G4ParticleHPInelastic;
theNeutronInelasticProcess->RegisterMe(theNeutronInelasticModel);

pmanager->AddDiscreteProcess(theNeutronInelasticProcess);
```

Особенности физических списков в Geant4.10+

Geant4.10 и выше позволяет работать в многопоточном режиме. Это особенность несколько изменит код, представленный в примере. Объекты классов-процессов и их компоненты объединяются в специальную структуру, являющуюся статической и одновременно локальной для каждого рабочего потока.

Сама структура будет выглядеть следующим образом:

```
struct ThreadPrivate {
   G4HadronElasticProcess* theNeutronElasticProcess;
   G4ParticleHPElastic* theNeutronElasticModel;
   G4ParticleHPThermalScattering* theNeutronThermalElasticModel;
   G4NeutronInelasticProcess* theNeutronInelasticProcess;
   G4ParticleHPInelastic* theNeutronInelasticModel;
};
```

а статическое поле

```
static G4ThreadLocal ThreadPrivate* tpdata;
```

Сразу стоит отметить, что так как поле статическое, инициализировать его надо аналогично глобальной переменной, т. е. вне пределов функций и классов:

```
G4ThreadLocal Test_HP::ThreadPrivate* Test_HP::tpdata = 0;
```

Что же касается самого класса, то он примет вид

```
class Test_HP : public  G4VPhysicsConstructor{
private:
   struct ThreadPrivate {
      G4HadronElasticProcess* theNeutronElasticProcess;
      G4ParticleHPElastic* theNeutronElasticModel;
      G4ParticleHPThermalScattering* theNeutronThermalElasticModel;
      G4NeutronInelasticProcess* theNeutronInelasticProcess;
      G4ParticleHPInelastic* theNeutronInelasticModel;
   };
   static G4ThreadLocal ThreadPrivate* tpdata;


public:
   void ConstructParticle() override {
      G4Gamma::GammaDefinition();
      G4Neutron::NeutronDefinition();
      G4Electron::ElectronDefinition();
      G4Proton::ProtonDefinition();
      G4IonConstructor::ConstructParticle();
      G4MesonConstructor::ConstructParticle();
      G4ShortLivedConstructor::ConstructParticle();
   }

   void ConstructProcess() override {
      if ( tpdata == 0 ) tpdata = new ThreadPrivate;
      tpdata->theNeutronElasticProcess = new G4HadronElasticProcess;

      tpdata->theNeutronElasticProcess->AddDataSet(new G4ParticleHPElasticData());
      tpdata->theNeutronElasticProcess->AddDataSet(new G4ParticleHPThermalScatteringData);
      tpdata->theNeutronElasticModel = new G4ParticleHPElastic;
      tpdata->theNeutronElasticModel->SetMinEnergy(4.0*eV);
      tpdata->theNeutronElasticProcess->RegisterMe(tpdata->theNeutronElasticModel);

      tpdata->theNeutronThermalElasticModel = new G4ParticleHPThermalScattering;
      tpdata->theNeutronThermalElasticModel->SetMaxEnergy(4.0*eV);
      tpdata->theNeutronElasticProcess->RegisterMe(tpdata->theNeutronThermalElasticModel);

      tpdata->theNeutronInelasticProcess = new G4NeutronInelasticProcess();
      tpdata->theNeutronInelasticProcess->AddDataSet(new G4ParticleHPInelasticData());
      tpdata->theNeutronInelasticModel = new G4ParticleHPInelastic;
      tpdata->theNeutronInelasticProcess->RegisterMe(tpdata->theNeutronInelasticModel);


      G4ProcessManager* pmanager = G4Neutron::Neutron()->GetProcessManager();

      pmanager->AddDiscreteProcess(tpdata->theNeutronElasticProcess);
      pmanager->AddDiscreteProcess(tpdata->theNeutronInelasticProcess);
   }
};
```

Стоит отметить, что методы ConstructParticle() и ConstructProcess() доступны для G4VUserPhysicsList и G4VModularPhysicsList соответственно. Однако в случае этих классов чисто виртуальными они не являются.