UNIVERSITY OF LIVERPOOL

PHYS 488

Modelling Physical Phenomena

Lecture 5

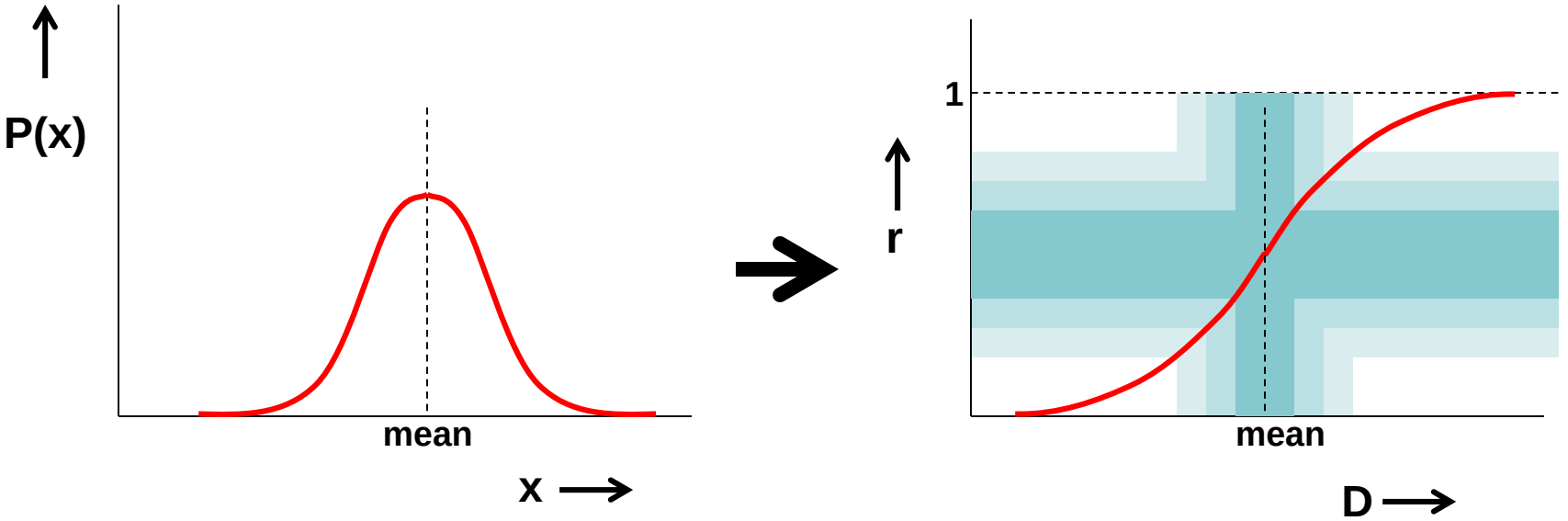
Phys488: What we learnt so far ..

If we have a normalised probability distribution P(x) such that

$$\int_{x_{min}}^{x_{max}} P(x)\,dx = 1$$

then values of D which satisfy $r = \int_{x_{min}}^{D} P(x)\,dx$

(where r is a uniformly distributed random number in the range 0 < r < 1) will follow the distribution P(x).

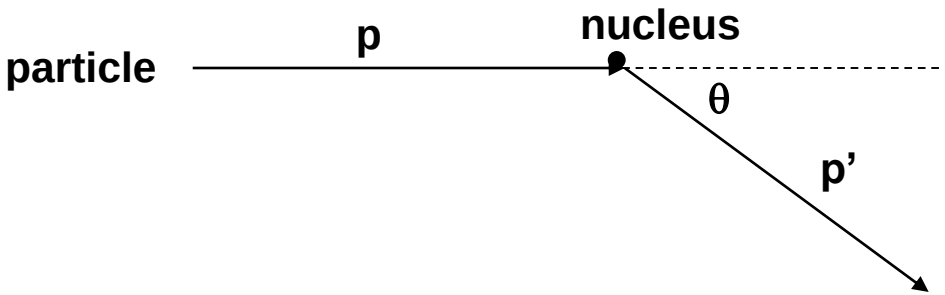


Example of why we need MC calculations/simulations

Many physics problems are difficult to calculate with traditional methods, because the problem involves a *convolution* of (many) different processes for which often we only know the distribution, i.e. what happens on average.

We will look at an example in this weeks exercise and we will use similar techniques in the projects.

An example. Imagine a very simple experiment: a particle scattering off a nucleus



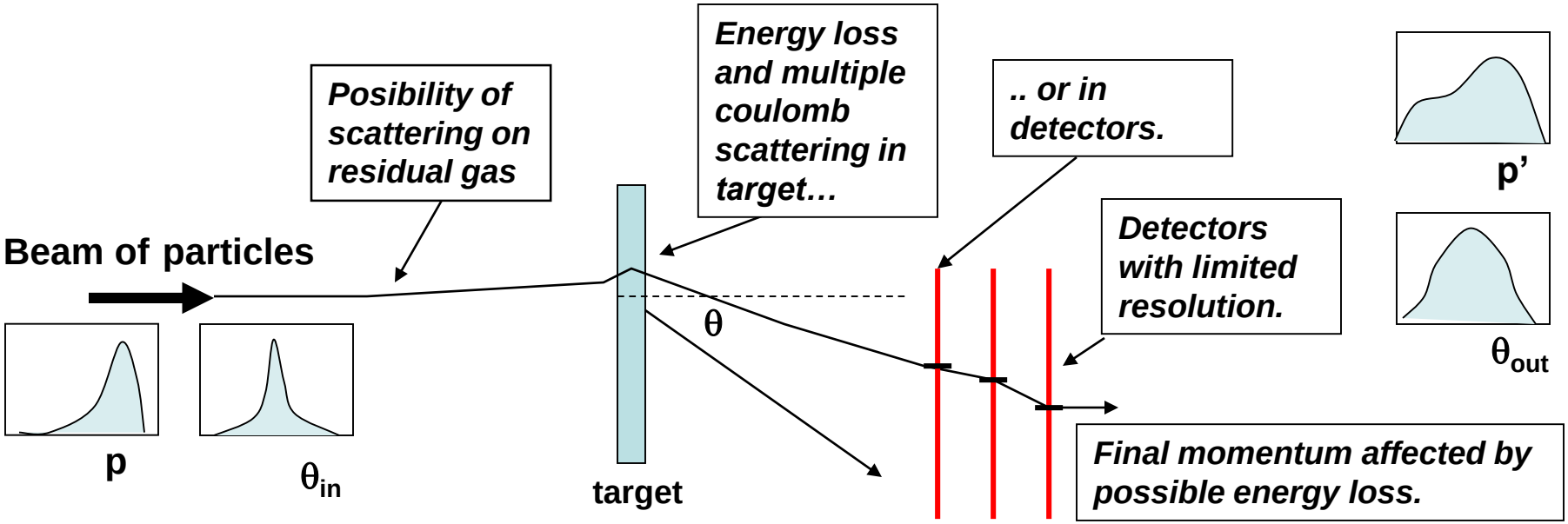
In this simple situation we can work out for example the momentum transfer from the variables p, p' and θ.

What if we tried to do the experiment for real..?

Example of why we need MC calculations/simulations

To do the experiment for real we need:

- **A beam of particles *(spread in direction and momenta particles)***
- **A target *(unpredictable how single particle will scatter)***
- **A detector to measure momenta and angles *(limited resolution)***



Many chance processes need to be taken into account.

On method to understand the results (and hence what physics you can learn from them) is to use *Monte Carlo techniques* to simulate the experiment.

Notes on the meaning of “static” in Java

The keyword static is used to define *class members* (i.e. both *class methods* and *class fields*).

A *field* is a variable that has class scope (as opposed to a locale variable)

Class fields are fields that are considered to belong to the class as a whole and should therefore be defined as static.

1. All constant fields (keyword final) should be static (e.g. the speed of light)

2. A variable field should only be static if it has to be accessible by all **objects** of the class.

Note on the meaning of “static” (cont.)

An example:

```
class Car
{
     private static int totalAge;
     private static int numCarObjects;
     private static double averageAge;
     private int age;
     public Car(int inputAge) // constructor method
     {
          numCarObjects++; // keep track of how many instance were created
          age=inputAge;
          totalAge = totalAge + age;
          averageAge = (double)totalAge/numCarObjects;
     }
     public int getAge() { return age; }
}
```

The **instance variable `age`** refers to one car (one instance of class **`Car`**) and is therefore **not static** (only one value **`age`** for every instance of **`Car`**).

The **class variable `averageAge`** refers to all cars (all instances of the class **`Car`**) and is thus defined as static. (only one value of **`averageAge`**, for the entire class, independent of number instances of **`Car`**)

Static variables are stored once in memory for the class (still accessible by all instances of a class), while non-static variables are stored once for every instance.

Note on the meaning of “static” (cont.)

Methods should (can) only be **static** if they make no use of *instance fields* (they may use *class* (i.e. **static**) *fields*). Such methods are known as *class methods* and are called, from outside the class, with the class name. (e.g. Math.sqrt(x))

The main method is a special kind of *class method*. There should, for obvious reasons, only ever be a single instance of the main method in any program, therefore main is declared **static**.

Consider the MyClass example on the next page which has:

1. a **non-static** (*instance*) field and a **static** (*class*) field
2. a constructor
3. a **non-static** (*instance*) method
4. a **static** (*class*) method

Note on the meaning of “static” (cont.)

```
class MyClass
{    // declaration of fields
     private  int number;               // example instance field
     private static int total = 0;      // example class field
     // Note: the constructor below allows multiple instance of this class to be created.
     public MyClass(int n)
     {
          number = n;
          total = total + n;
     }
     public int getNumber() { return number; }        // example instance method
     public static int getTotal() { return total; }   // example class method
}
```

```
import java.io.*;
class MyProg
{    // Since main is declared to be static only one instance of it will exist!
     public static void main(String [] args)
     {
          MyClass instance1 = new MyCLass(1);
          MyClass instance2 = new MyClass(2);
          // next: two calls to the instance method getNumber()
          System.out.println("instance1 number: " +instance1.getNumber() );
          System.out.println("instance2 number: " +instance2.getNumber() );
          // next: one call to the class method getTotal()
          System.out.println("total: " + MyClass.getTotal() );
     }
}
```

More on good practice

```
protected double p, beta, gamma, mumass...
public double beta(double p)
{
     beta= p/Math.sqrt(p*p+mumass*mumass);
     return beta;
}
public double gamma()
{
     gamma = 1/Math.sqrt(1-beta*beta);
     return gamma;
}
```

This is not good practice, since the method gamma can only be used, if beta has been called previously.

Instead:

```
static double mumass=106;
public double beta(double p)
{
     return p/Math.sqrt(p*p+mumass*mumass);
}

public double gamma(double p)
{
     double b=beta(p);
     return 1/Math.sqrt(1-b*b);
}
```

PHYS488: Work for Week 5

1) Today you have been given two MATHCAD examples of computing the energy loss

$$dE/dx$$

and the multiple coulomb scattering (MCS) parameter

$$\theta_t = \sqrt{2} \times \theta_0$$

for a MUON with momentum ***p MeV/c*** moving through matter.

The task today is to produce Java code for these.

Write instance classes EnergyLoss and MCS where separate instances represent separate materials. The instantiation of instances of these classes would look something like this:

EnergyLoss ironEloss = new EnergyLoss(parameters for iron);

EnergyLoss copperEloss = new EnergyLoss(parameters for copper);

MCS ironMS = new MCS(parameters for iron);

MCS copperMS = new MCS(parameters for copper);

And so on…*carefully* study the structure of class Histogram and try to adapt it to use the same sort of structure in theclasses EnergyLoss and MCS.

p.t.o.

PHYS488: Work for Week 5

Once you have written the instance classes EnergyLoss and MCS, write a program that uses these instance classes to do the calculations for a muon traveling through iron. Find the energy loss or MCS angle by using instance methods like:

```
double dEbydxIron = ironEloss.getEnergyLoss(momentum);
double thetaT = ironMS.findSigma(momentum);
```

You MUST(!) show output that proves that your program is producing the correct results (check that you get the same numbers for the muon dE/dx and MCS parameter as on the MATHCAD handouts).

[2]

p.t.o.

PHYS488 Week 5: Simulation muon in iron
The sample tracking Class 'Trackmuon.java'

On **VITAL** you find is a simple MC Class that simulates a muon travelling through an iron sheet. When it exits the back of the iron it is detected by two counters situated 10 and 20 cm beyond the back. The muon generates 'hits' in the detectors that are *smeared by the expected detector resolution*. In a real experiment hits like these would form part of the experimental data.

The muon is assumed to start just inside the iron at (x, y) = (0, 0) and has a fixed energy at the start, inputEnergy, which the user types in. The particle is then taken through the iron in small steps, at each stage calculating the energy loss and an angle generated by MCS. As the particle is followed, the (x, y) co-ordinates of each point reached by the muon are stored in a 2D array called 'trackOfMuon'. The muon is followed until:

1. It leaves the back of the iron sheet,
2. or, the muon has lost so much energy it stops in the iron.
3. or, too many steps have been taken (this is a safety trap to stop infinite loops).

When a muon comes out of the back of the sheet, useful information is passed to the class Method *lookAtThisMuon* ready to be written to disk for off-line analysis.

The program is incomplete in 3 respects:

1. the correct code to find the energy loss is not yet included.
2. the correct code for calculating the MCS angle θ_0 is not coded.
3. the code for writing data to disk is not there.

Work for Week 5:

2) Work on the following tasks:-

Read the program and try to understand how it works. Ask for help over any points you are not sure about. Set up the code so it makes use of the instance classes you wrote in part 1, for **calculating the energy loss and MCS angle**.

a) Write 10 sample muon tracks (i.e. the (x,y) pairs) into a file on disk and draw them using an x-y scatter plot chart in EXCEL. Discuss what you observe.

[1]

b) Make histograms (based on a sufficiently large number of muons) of the energy and y-position of the muons as they exit from the back of the iron. Try a range of muon energies (low, medium and high!). Discuss what you observe. Why is the energy distribution very narrow (discuss possible limitations in the way it is simulated).

[1]

c) Add in an extra detector 30 cm beyond the iron and generate hits in that. Make a histogram of the hit positions in y in detectors 1, 2 and 3. Discuss what you observe.

[1]

d) If the iron were 1 m thick, ***estimate*** **(using the program) the minimum muon energy** for which at least 50% of the muons pass through the iron without being stopped. Discuss the method you could use to estimate this value. Include an estimate/discussion on the uncertainty(ies) on your result.

[1]

The group report on this weeks work should show the code that has been produced and briefly discuss the role that each group member played in performing these tasks. After this report is submitted the group project will be assigned.

TrackMuon: Muon tracking through Iron (one step)