UNIVERSITY OF LIVERPOOL

PHYS 488

Modelling Physical Phenomena

Lecture 4

Phys488: What we learnt in week 3.

Last week we encountered the powerful concept of an ***abstract data type*** (**class Histogram)**. By making ***multiple instances*** of this class with different parameters, e.g.

```
Histogram hist1 = new Histogram("Random",20,0.4,0.9);
Histogram hist2 = new Histogram("Gauss",40,-1.0,1.0);
Histogram hist3 = new Histogram("D ln(r)",40,0.0,4.0);
```

any number of histograms can easily be made this way.

Several useful instance methods could be added to this class.

E.g. with an *instance method* **writeToDisk()** included in **class Histogram**, it was very easy to write each histogram to disk into its own file. This could be done for example by passing the filename to the method in the parameter list as a String:

```
hist1.writeToDisk("hist1.csv");
```

Monte Carlo techniques

In PHYS205 you saw how to calculate the value of π by throwing pairs of random numbers.

Particularly useful to simulate physics processes that involve a convolution of multiple probability distributions.

Last week we used the *gauss* method to throw random numbers following a Gaussian distribution function.

Look up the “Central Limit Theorem” to understand why the *gauss* method gives you a distribution that is (approximately!) Gaussian.

This week we encounter a general method that allows us to throw random numbers following any probability distribution.

Looking ahead:

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

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

then values of **D** which satisfy

$$r = \int_{x_{\min}}^{D} P(x)\,dx \qquad (2)$$

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

Sometimes it is possible to do this integral ***analytically***, and solve it **to find D as a function *r*** (as in assigment 2 below).

Otherwise the integration has to be done ***numerically***. For this it is adequate to approximate the integral as the sum of the areas of small strips (bins) from the lower limit up to **D**. An example of this attached (**`class ThrowDist`**).

PHYS488: Work for Week 4

1. On the Excel chart showing the Gaussian distribution you did last week, draw the predicted curve. Hint: the predicted contents of a bin in the histogram is given by: ***N P(x) dx*** where ***N*** is the total number of entries in the histogram (including underflows and overflows), ***P(x)*** is the normalised Gaussian distribution, ***dx*** is the bin width and ***x*** is the coordinate of the bin centre.

   One can use the **Chi-squared per Degree of Freedom** (CSFD) value to compare a theoretical function (here a predicted curved) **T(x)** with data (here a randomly thrown distribution) $\mathbf{E_i}$. For all the data values at coordinates $\mathbf{x_i}$ one evaluates the χ^2 value.

$$\chi^2 = \sum_i \frac{(T(x_i) - E_i)^2}{\sigma^2}$$

   where σ is the error on $\mathbf{(T(x_i) - E_i)}$ and CSPD is defined as: **CSPD = χ^2 / d.o.f.** with **d.o.f.** (the number of degrees of freedom) = number of data points – number of **free parameters** in the theory curve **T(x)**.

   (free parameters occur when a theory function is fitted to the data)

   The curve T(x) is in good agreement with the data if CSPD ~ 1.

   For your Gaussian distribution calculate the CSPD in EXCEL**.**

[1]

p.t.o.

Work to be done this week

2. You will produce a histogram with random numbers following the probability distribution $\mathbf{P(x) = x\ exp(-x^3).}$

   First find the normalisation factor for $\mathbf{P(x) = x\ exp(-x^3)}$ using the ***Integrate class***. You will have to adjust the range and the number of integration steps taken. By varying these parameters convince yourself (and explain why) that your normalisation factor is correct to at least 4 decimal places.

   Now produce the histogram using the ***ThrowDist class*** and show that the generated distribution follows the expression N P(x) dx by plotting the expected curve together with the thrown distribution and by calculating the CSPD value.

   [1]

p.t.o.

Work to be done this week

3. A distribution of e.g. decay lengths**,** given a mean free path **λ,** is given in the **interval 0 < x <** ∞ by

$$P(x)\,dx = \frac{\exp(-x/\lambda)}{\lambda}\,dx \qquad (3)$$

Show that this is normalised.

In this particular case, the integral (2) **can be done analytically**. Do this, and show that the explicit expression for D in terms of r is:

D(r) = -λ *ln(r)

[tip: if r is a random number between 0 and 1, then so is (1-r)]

Last week you showed that this formula 'throws' an exponential distribution of **D**'s **(**using **λ =15).** Draw the expected curve given by equation (3) on the EXCEL histogram you made last week and show they agree using the CSPD value.

[1]

p.t.o.

Work to be done this week

4. Use *ThrowDist* to produce a histogram of random numbers following a Gaussian distribution:

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(x - mean)^2}{2\sigma^2} \right)$$

Use the same sigma (σ) and mean parameters as for your Gaussian distribution of week 4.

As in week 4's exercise make a graph of the produced histogram, showing also the expected distribution and calculate the CSPD value.

You'd expect the CSPD value to be slightly better since as the previous Gaussian distribution was only approximate. Do you see any improvement?

[1]

p.t.o.

More challenging work to be done this week

5. write a program (using your work for parts 3 and 4) to simulate a distribution of lifetimes measured with a detector with a limited resolution.

   To do this you will need to convolution the probability distributions of parts 3 and 4.

   First throw a random number following the distribution of decay lengths for a particle with a mean free path of 15 cm(see part 3).
   Then add to this number a second random number following a Gaussian distribution with mean 0 and width of 5cm (see part 4).

   Make a histogram of the resulting distribution using 30 bins between -5cm and +10 cm.

[2]

p.t.o.

Work for Week 4: class Integrate

```
// Example of Integrating an arbitrary distribution
import java.io.*;
import java.util.Random; // notice this..needed to load the class Random.
class Integrate
{
     static PrintWriter screen = new PrintWriter( System.out, true);
     private static double func(double x)
    {
          return ( x*x*Math.exp(-x*x));
     }
     public static void main (String [] args ) throws IOException
    {
          double sum = 0;
          int nsteps = 200;// this is a guess.. making it smaller will make program faster
                           // but the result will be less accurate
          double maxOfX = 4;
          double minOfX = 0;
          double x;
          double deltaX = ( maxOfX - minOfX )/(double)nsteps;
          for (int n = 0; n <= nsteps-1; n++) // integrate func(x)
          {
           x = minOfX + n*deltaX + deltaX/2. ;
           sum =sum +  func(x)*deltaX;
          }
        screen.println("Integral is " + sum);
    }
}
```

Work for Week 4: `class ThrowDist`

```
// Example of throwing an arbitrary distribution in a Monte Carlo program.
import java.io.*;
import java.util.Random;
class ThrowDist
{
     static BufferedReader keyboard = new BufferedReader (new InputStreamReader(System.in));
     static PrintWriter screen = new PrintWriter( System.out, true);
     static Random value = new Random();
     //---------class Methods start here ------------------------------------------------
     private static double p(double x)
     {
          // define the normalised function to be 'thrown'.
          // remember, p(x) MUST BE normalised over the range    minOfX <= x <= maxOfX .
          return ( x*x*Math.exp(-x*x)/0.44311);
     }
     //-------------------------------------------------------------------------------
```

p.t.o.

Work for Week 4: `class ThrowDist`

```
private static double  throwAValue()
{
     double sum = 0;
     double nextOne;     // primitive variable to store each random number
     int nsteps = 200;   // this is a guess.. making it smaller will make program
                         // faster but the 'thrown' function will not be as smooth.
     double maxOfX = 4;    // above this x-value, p(x) ~ 0.
     double minOfX = 0;    // below this x-value, p(x) ~ 0.
     double x = 0;
     double deltaX = ( maxOfX - minOfX )/(double)nsteps;
     nextOne = value.nextDouble();// generate a new random in range (0 , 1)
     // integrate p(x) until sum > the random number 'nextOne'.
     for (int n = 0; n <= nsteps-1; n++)
     {
          x = minOfX + n*deltaX + deltaX/2. ;// find x-value at the centre of each
strip.
          sum =sum +  p(x)*deltaX;  // add up the areas of the strips
          if ( sum > nextOne ) break;// note this way to jump out of a loop
     }
     return x ;
}
//------------------------end of class methods-------------------------------------
```

p.t.o.

Work for Week 4: `class ThrowDist`

```
    public static void main (String [] args ) throws IOException
    {
         int trials;    // number of random numbers to generate
         double nextX;  // random number 'thrown' from the distribution
         int numberBins;
         Histogram hist1 = new Histogram(" P(x) = x*x exp(-x*x) ",20,0,4);
         screen.print( "Input the number of random numbers to generate ");screen.flush();
         trials = new Integer(keyboard.readLine()).intValue();
         for ( int goes=1; goes <= trials; goes++)
         {
              // Make histogram and store it in disk to import into EXCEL to make the plot.
              nextX = throwAValue();
              hist1.fillh(nextX);
         }
         //histogram has been filled. Show the contents on the screen.
         numberBins = hist1.getSize();
         for (int bins =0; bins <= numberBins-1; bins++)
         {
              screen.println(  hist1.getContent(bins) + "\t");
         }
         hist1.writeToDisk("histfile.csv");
    }
}
```