Numerical Recipes

2015/16 Dept Physics and Astronomy

Unit 2

Random number generation and numerical integration

Random number generators & Numerical Integration

Aim:

- To use a standard function to return a random number between 0-1
- To generate a random number according to an arbitrary distribution
- To use library random generators
- To understand how functions may be integrated numerically
- To use a random number generator to perform Monte Carlo integration of a function

Random number generators : uniform distribution

All programming environments will give you access to functions which will return to you a random number generated with a uniform distribution in the range 0-1

Actually lots of different random number generators will exist, but we will start with this simple one

In Java use the Random class. Here is a code fragment to show you how to use it.

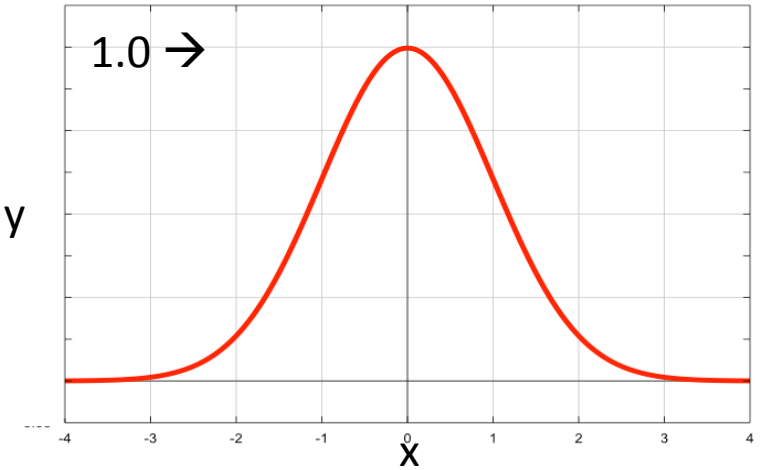
The method nextDouble() returns a random number in the range 0-1

```
import java.util.Random;

...........

Random randomGenerator = new Random();

for (int ix = 1; ix <= 50; ++ix){

    double x = randomGenerator.nextDouble();

    System.out.println("Next number "+x)

}
```

Random number generators : Arbitrary distribution

It is far more likely that you will want to generate a random number drawn from a specified distribution. A simple example is to produce a random number drawn from a Gaussian distribution. Lets be quite clear what this means:

- A Gaussian distribution looks like this:



The formula is

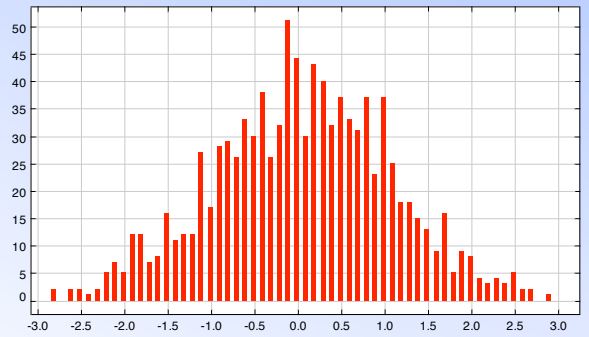
$$y = \exp(- (x-\mu)^2 / 2\,\sigma^2)$$

where

μ = the mean

σ = the width

- We want to generate a series of values of x, such that if you plotted each of them in a histogram (i.e. a binned histogram where you count the number of times a generated value falls into each bin) then the resulting distribution is Gaussian in shape.



Random Generation:

Box method

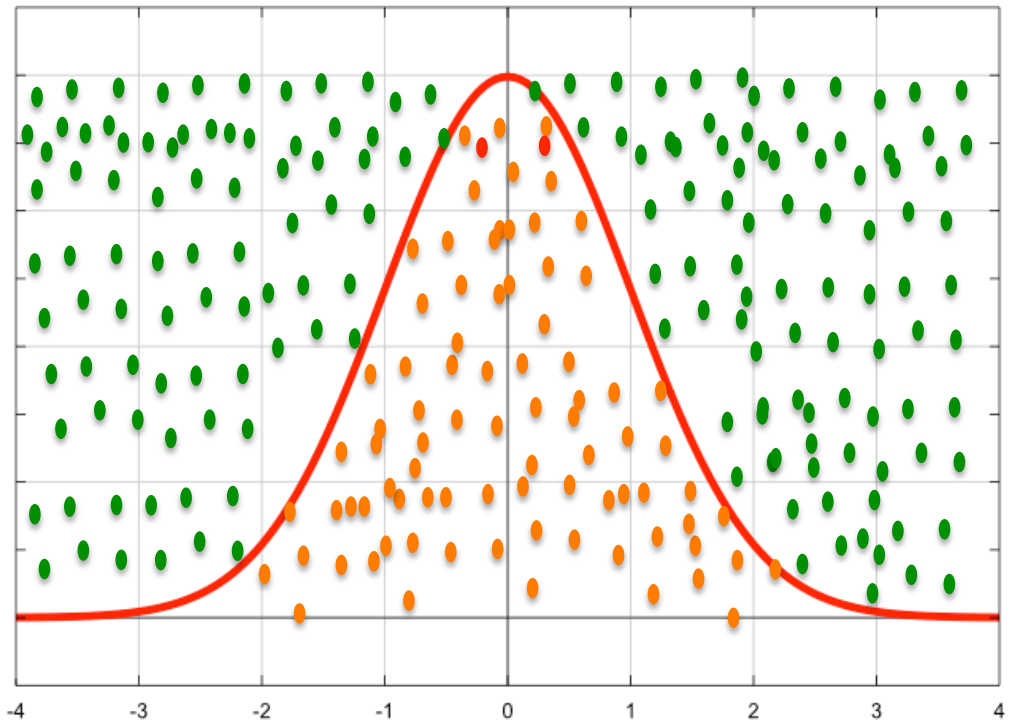
Random number generators : Arbitrary distribution

To do this we use a technique involving two random numbers. For the record the procedure is written out here, but this is best understood by hands on writing the code

1. Let the distribution you wish to generate be f(x)
2. f(x) is defined in the range [a,b], in other words $a < x < b$
3. Find a number fmax which is at least greater than the maximum value of f(x) in this range.
4. Generate a random number x1 in the range 0-1
5. Convert it to a random number in the range [a,b] by the operation $x1 = a + (b-a)*x1$
6. Calculate the value, y1, of the function at x1, in other words $y1= f(x1)$
7. Now generate another random number y2 in the range 0-1
8. Convert this to a random number in the rang [0,fmax]] by the operation $y2 = fmax*y2$
9. Perform the test if($y2 < y1$)
    1. If this is true then return the value of x1 to the user
    2. If this is false then discard the value of x1 and go back to step 4 and keep repeating until the test is true
10. Keep repeating this process until you have as many random values of x1 returned to you as you want.

At the end of this process if you plot all the values of x1 returned to you you will see that they have the shape of f(x)

Picture of the algorithm



This algorithm is equivalent to generating random numbers in a “box”
Then only keeping those which fall under the curve (the orange ones)

Random Generation:

Inverse Cumulative method

Random number generators : Inverse Cumulative method

This uses only 1 random number.
But you have to have the form of the ”cumulative function” and of the “inverse cumulative function” of the function you wish to generate. You may not know these easily!

Let f(x) be defined in the range [a, b], then

Cumulative: $y' = g(x) = \text{Integral}\{ f(x) \}$ from a → x < b
Inverse: $x = g^{-1}(y')$

To generate a random number, x, in the range [a, b] distributed according to f(x) then

1. First generate a random y’ in the range[0, 1]

2. Convert it to a random number in the range [g(a) , g(b)] == [0, g(b)] by the transformation :
$y' = g(b) * y'$

3. Perform the inverse $x = g^{-1}(y')$

4. Repeat this many times

At the end of this process if you plot all the values of x returned to you you will see that they have the shape of f(x)

[Note: if f(x) is normalised to 1 in the range, then g(b) = 1 and so step 2 is not needed.]

Random number generators : Inverse Cumulative method

- This looks like it saves a random number generation, but the cost may be in getting the cumulative and inverse functions.

- Example: for a Gaussian f(x) the cumulative and inverse are hard to calculate, and involves error functions which can use numerical methods -> so you might save nothing

- But you can use clever functions to construct an inverse function mapping in some parameterised way. This may then depend upon the accuracy with which you do this...etc...

- A simple function for which it works easily is the exponential distribution:

$$f(t) = 1/\tau * \exp(- t/\tau)$$

defined in range [0, infinity] and which is normalised to 1 in that range

Then you can easily show that

$$y' = g(t) = [\ 1 - \exp(-t/\tau)\ ]$$

giving:

$$t = g^{-1}(y') = - \tau * \ln (1 - y')$$

Exercises

Random number generators : Arbitrary distribution : Exercise-1

- Implement a class MyGaussianPdf which has (at least) the methods shown below
- This means you will implement one of the prescription(s) on the previous page.
  - For simplicity the box method is easier so do this first.
  - If you are feeling brave then implement both and compare the timing
- Test it in a loop to create 1000 random numbers.
- Plot these in a histogram** and verify that they have a Gaussian distribution by comparing to the the value you get using the evaluate() method which will code up the formula.

```
class MyGaussianPdf {

    // Constructor
    public MyGaussianPdf( double mean, double width ) ;

    // To return a random value of x with Gaussian distribution
    public double next() ;

    // To evaluate the Gaussian at point x
    public double evaluate( double x) ;

    ....
}
```

** see separate course instructions on histogram plotting

Random number generators : Exercise-2

- Do the same exercise as before to generate a Gaussian random distribution using the inbuilt Gaussian random function
  - Gaussian : `Random.nextGaussian() ;`

```
import java.util.Random;

...........

Random randomGenerator = new Random();

    ....

    double x = randomGenerator.nextGaussian();

    ....

}
```

Random number generators : Python

```
import numpy  as np
import pylab  as pl
import numpy.random as rand  # Random number generation module

#Here we generate arrays of size N (shape N x 1) floats distributed
#according to the given PDF with mean=mu and standard deviation=sigma.

#Uniform distribution in the interval [-15.0, 15.0], N=99000
UniformSamples = (15.0 - -15.0)*rand.random_sample(99000) + -15.0

# Gaussian distribution, GaussSamples = sigma*rand.randn(N,1) + mu, e.g
GaussSamples   = 5.0*rand.randn(99000, 1) + 2.0

# Similarly for many other common PDF's see:
# http://docs.scipy.org/doc/numpy/reference/routines.random.html

# e.g. the lognormal distribution, rand.lognormal(mu, sigma, N)
LnSamples = rand.lognormal(1.0, 0.7, 99000)
```

Random number generators : C++

- Only easy random generator is `std::rand()` which returns an integer

```
#include <stdlib.h>

......
// This makes a double random number
// rand() returns an integer between 0 and RAND_MAX

double r = (double)  rand() /  (double) RAND_MAX

.....
```

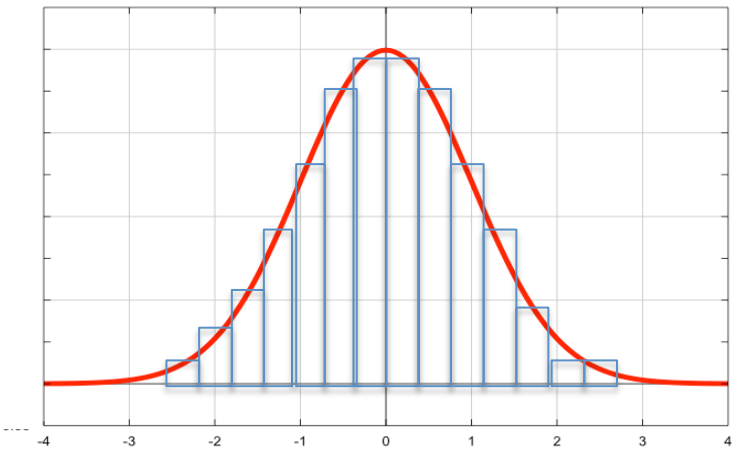
- For other random generators you need to use the GSL library (Gnu Scientific Library)
  - Look in the code examples `/Unit3-Random/cpp/` to see how to do this, and in particular how to compile and link against GSL

Unit 3

Random number generation and **numerical integration**

Numerical integration of a function

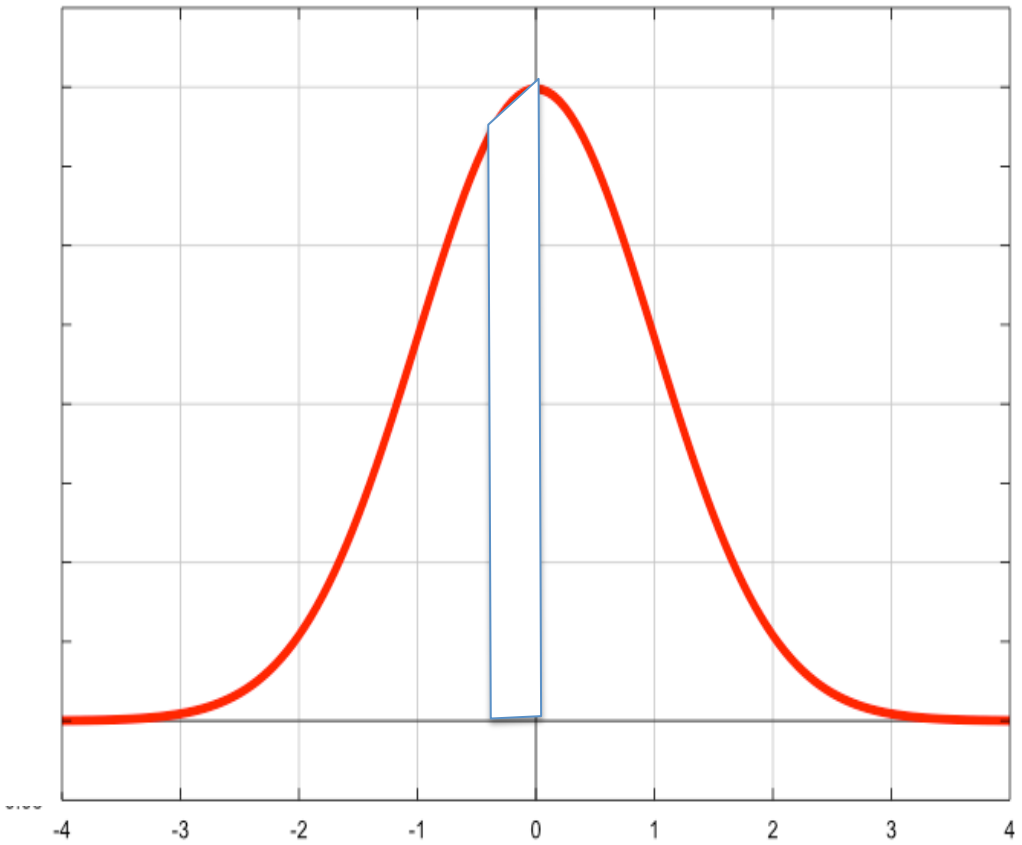
- We have created a MyGaussianPdf class which has an evaluate method which returns the value of the function at some point x
- As we will see in a little while, there are many situations where you need to know the area under a the function within some range. I.e. the integral of the function between x1 -> x2
- Sometimes you can calculate it analytically (you are lucky !) but sometimes this is impossible.
- If it is not possible to do it analytically, then you can perform a numerical integration using various techniques
- The simplest is to fill the shape with rectangles:
- Each rectangle has fixed width and height equal to value of function at the centre
- You know how to calculate the area of each
- You sum over rectangles to obtain the approximate area under the curve.
- You can make the error as small as you like by making thinner rectangles



Numerical integration of a function

- The next most sophisticated thing to do is use a more complex shape to fill the curve

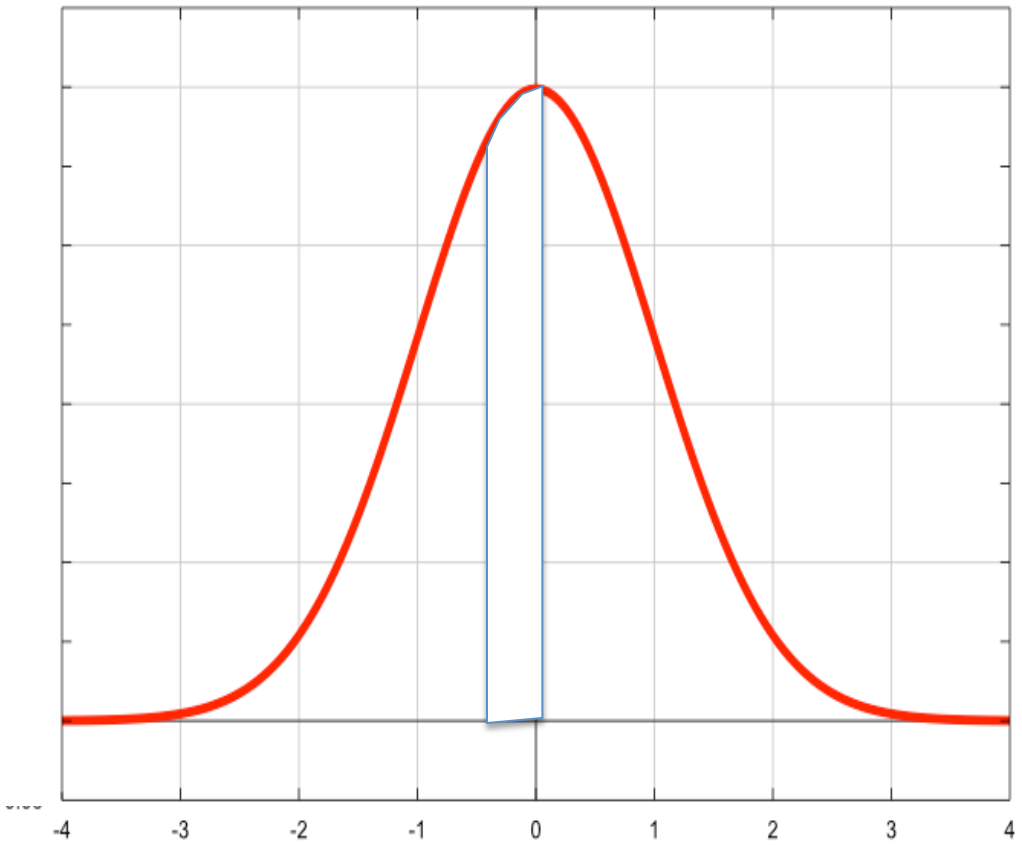
- Rectangle+triangle on top
- In fancy terms you say you are using a “first order polynomial”
- The first case (previous slide) was a zero order polynomial
- It is straightforward to calculate area of each unit
- Some thought is required as to where to set the height on left and right to get most accurate results.



Numerical integration of a function

- Clearly one can become even more accurate by using a higher order polynomial

- This is a second order polynomial
- You can use as much complexity as needed in order to get a result as accurate as you need for your problem

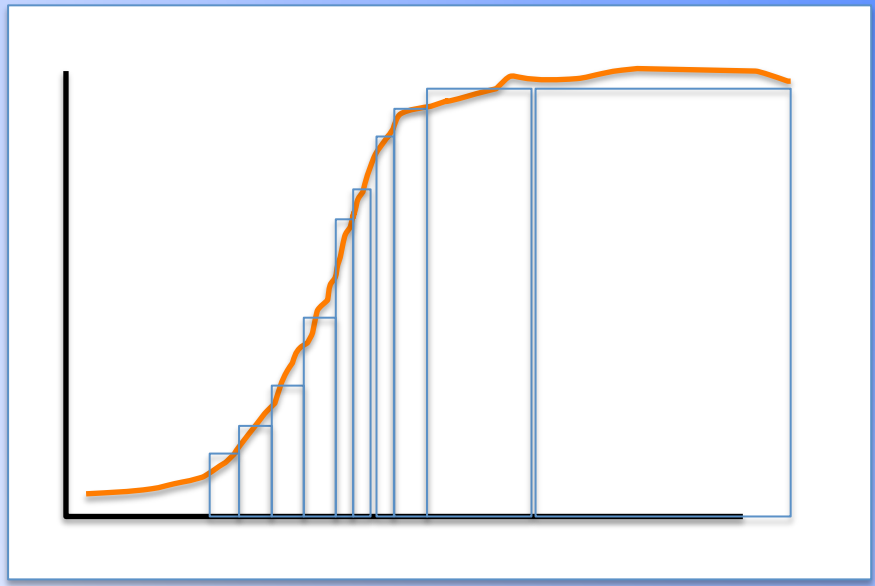


Numerical integration of a function

- Adaptive algorithms:

  - No need to have rectangles at same width.
  - You can adjust width (based on derivative of function)

  - Here is example:



- Errors

  - In general numerical integration functions will estimate the size of the error they incur.

  - You can control this by choosing the method and its parameters.

There are several algorithms for this type of integration in the 'Numerical Recipes' book

Numerical integration of a function : by Monte Carlo

- Sometimes it is impossible to do an integration by filling the curve with boxes.
- This is particularly so in multiple dimensions (we only looked at a 1-D curve)
- In such cases one can use a “Monte Carlo” integration method
- A prescription is (see next slide for picture)

1. You define an area which includes that of the unknown area, and for which the area is easily calculable, call this A. This is often a simple “box”, but can be anything.
2. You then generate random numbers to uniformly sample this area, and count the ratio of those falling within the unknown area to the total, call this ratio F
3. The unknown area if then given by A*F

Numerical integration of a function

Formula:

$y = \exp(- (x-\mu)^2 / 2\,\sigma^2)$

where

$\mu=0$

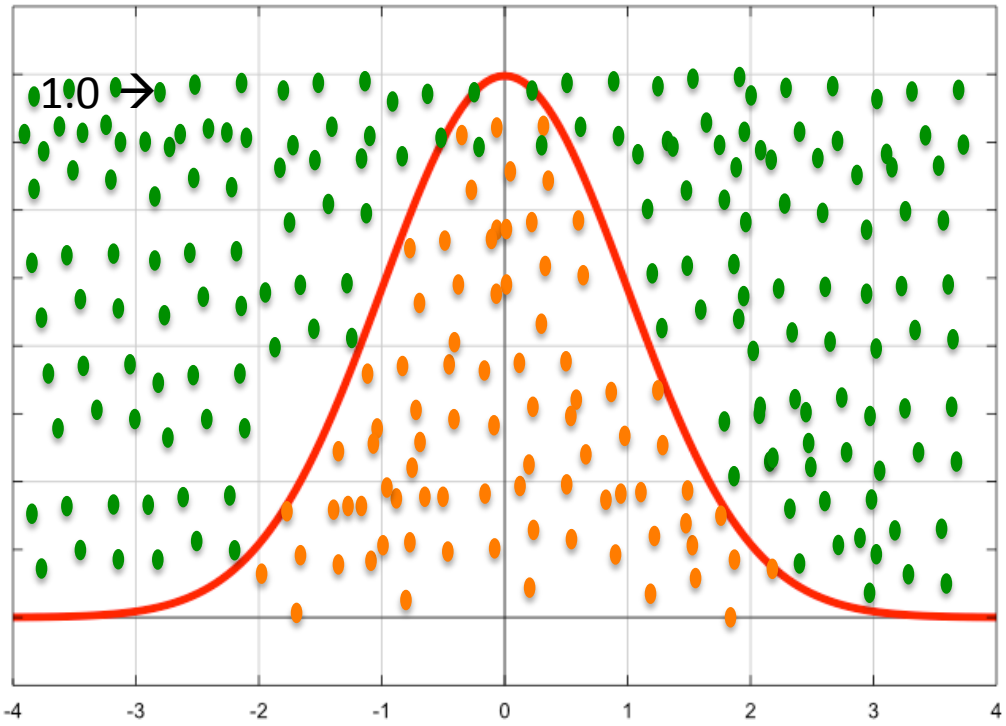
$\sigma=1$

Area of "box" is 8*1 = 8 units

Throw darts at the box

The ones which go inside are red
The ones which go outside are green

F = red / (red+green)



Total area under curve = 8 * F

Numerical integration of a function: issues

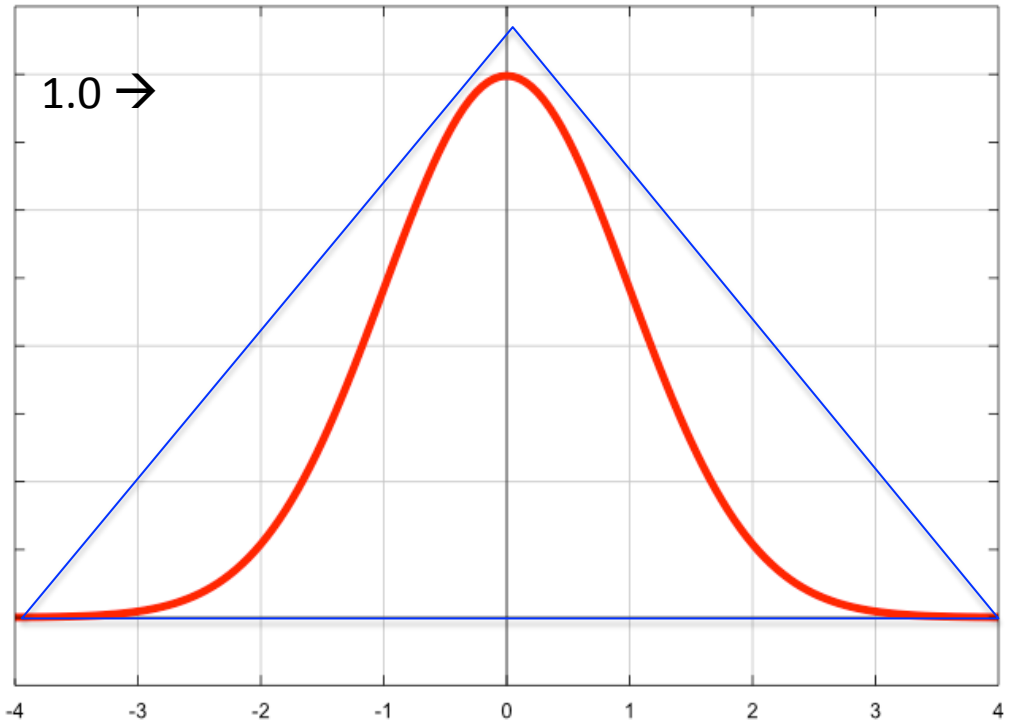
You do this you may find you need to throw a lot of darts to get an accurate result

This can be very inefficient as most darts miss the area in question.

One improvement is not to use a box, but a shape which approximates the curve and which you know the area of.

A triangle would be better here

There are many clever techniques for optimising sampling and you would generally use a numerical integrator from a package which includes all of the appropriate clever technology.



Numerical integration of a function: Another Monte Carlo Method

The method described first is simple to understand.

But it requires 2 random numbers and it requires to know the maximum of the function (which may not be difficult)

Another method is described here. Probably more common.

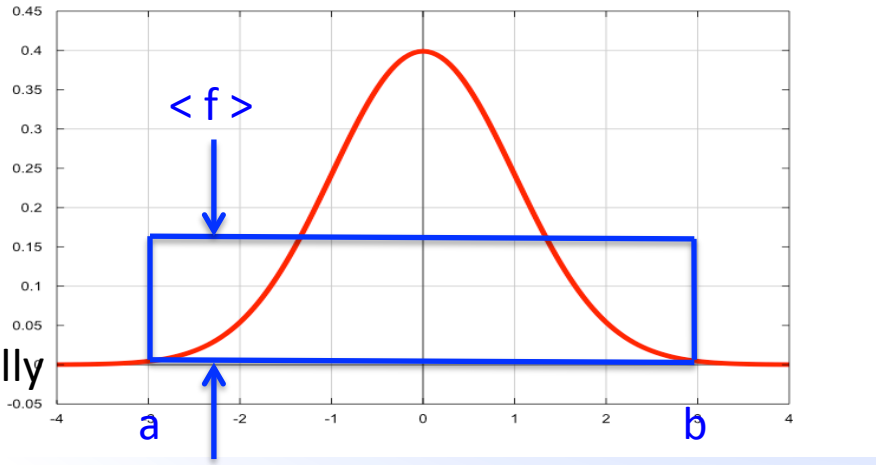
It uses the idea that there is a box which can be drawn (in the integration range) which has the same area as the function.

The height of this box is just the "average value of the function over the region" < f >

Thus

Area = (b-a) * < f >

So it comes down to calculating <f> numerically



- Calculating < f > in the range a→b numerically is straightforward

$$\langle f \rangle = \sum_i f(x_i) \; / \; N$$

where N values of x_i are chosen with a flat random distribution in the range a→b (exactly as we have done several times before).

- Thus the area becomes

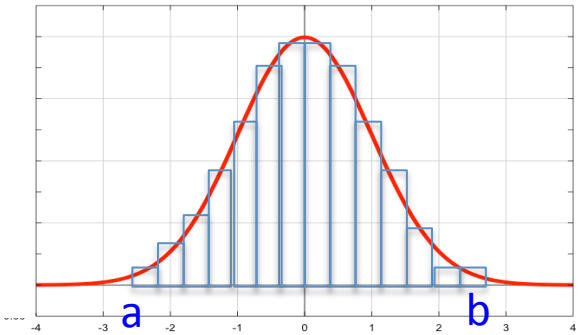
$$\text{Area} = (b - a) \times \sum_i f(x_i) \; / \; N$$

```
//Method to return the numeric integral of the function between
// lo < x < hi

    public double integralNumericII( double lo, double hi ) {

        int npoints = 1000000 ;

        double sumf = 0;

        for( int ii=1; ii<=npoints; ++ii ) {

            //Generate a random number on the x-axis
            double x = lo + randomGenerator.nextDouble()*(hi-lo) ;

            //Calculate and accumulate value of function
            sumf+= evaluate(x) ;
        }

        double Area = sumf *(hi-lo) / npoints ;

        return Area ;
    }
```

...another way to look at this...

Here is another way to arrive at the same formula

Imagine for a moment that you were integrating using the summation of uniform rectangles shown earlier .



PICTURE OF SAMPLING

For N uniform rectangles in the range a → b , the width of each would be

$$(b-a) / N$$

and the area of each rectangle would be height x width:

$$f(x_i)\ (b - a)\ /\ N$$

and so the total area would be the sum over rectangles

$$\text{Area} = \sum_i\ f(x_i)\ (b - a)\ /\ N$$

where the x_i would be regularly spaced $x_i = a + i\,(b-a)/N$

...another way to look at this...

Then it is a minor step to say instead of summing over regular x_i

$$\text{Area} = \Sigma_{\text{regular\_i}} \; f(x_i) \; (b - a) \; / \; N$$

you instead sum over random x_i in the range a→ b

$$\text{Area} = \Sigma_{\text{random\_i}} \; f(x_i) \; (b - a) \; / \; N$$

In effect you are saying you pick random rectangles, with an average area given by (b - a) / N

...why bother...

The astute amongst you will correctly ask why bother to sample randomly if you can do it with regular rectangles...

... and the answer is as always – that you are being shown the principle of the method.

For a 1-dimensional problem you probably would not need to use Monte Carlo integration.

But for a higher dimensional problem you would more likely need to. It might be impractical to define a large number of hyper-rectangles in a multi-dimensional space.

By sampling randomly you avoid the problem - (although may get into other problems of efficiency if the shape has spikes)

Multi dimensional integration

In multi dimensions, i.e. a function of x,y,z

$$f(x, y, z)$$

it generalises as

$$\text{Area} = \Sigma_{\text{random\_i,j,k}} \; f(x_i, y_j, z_k) \;\; \text{range(x) range(y) range(z)} \; / \; N$$

Numerical Integration: Exercise

- Add two methods to your MyGaussianPdf class
  - integralNumeric : which performs a numerical integration using the Monte Carlo method(s)
    - Implement as many as you wish and compare if you have time.
  - integralAnalytic : which calculates same analytically
- Demonstrate these methods and compare the results ?
- How many points do you need to get a result accurate to 1% ?

Note: you will need to consider the integration limits.

- The analytic integral of a Gaussian from –infinity to + infinity is $\sigma \sqrt{(2\pi)}$, but you cannot do the numerical integration between these limits.
- So either you need to make the numerical integration from about -5 sigma → + 5 sigma (which is approximately infinity) and assume the analytic integral is approximately $\sigma \sqrt{(2\pi)}$
- Or you need do it properly and allow the integral between arbitrary limits and then code up the proper analytic integral which involves error functions. This is much harder and you probably wont get it done in the workshop.

Specialist lecture

Dr Steven Booth

will give a specialist lecture this afternoon on the subject of random numbers

Stephen works for EPCC and is involved in many computing projects.

He is a national/international expert