Numerical Physics with Probabilities: The Monte Carlo Method and Bayesian Statistics Assignment 1

Department of Physics, University of Surrey module: Energy, Entropy and Numerical Physics (PHY2063)

1 Numerical Physics part of Energy, Entropy and Numerical Physics

This numerical physics course is part of the second-year Energy, Entropy and Numerical Physics module. It is online at the EENP module on SurreyLearn. See there for assignments, deadlines etc. The course is about numerically solving ODEs (ordinary differential equations) and PDEs (partial differential equations), and introducing the (large) part of numerical physics where probabilities are used.

This assignment is on numerical physics of probabilities, and looks at the Monte Carlo (MC) method, and at the Bayesian statistics approach to data analysis. It covers MC and Bayesian statistics, in that order.

MC is a widely used numerical technique, it is used, amongst other things, for modelling many random processes. MC is used in fields from statistical physics, to nuclear and particle physics.

Bayesian statistics is a powerful data analysis method, and is used everywhere from particle physics to spam-email filters. Data analysis is fundamental to science. For example, analysis of the data from the Large Hadron Collider was required to extract a most probable value for the mass of the Higgs boson, together with an estimate of the region of masses where the scientists think the mass is. This region is typically expressed as a range of mass values where the they think the true mass lies with high (e.g., 95%) probability. Assignment 1 is on this sort of problem.

Many of you will be analysing data (physics data, commercial data, etc) for your PTY or RY, or future careers[1] . And some will using MC or related modelling techniques.

2 Very brief background note on probabilities

You have come across probability before, for example, in quantum physics, and in introductory data analysis. If you want a reminder/description of what is meant by probability/randomness then the Wikipedia Probability page is worth a look.

Note on math notation: We will working with probabilities, these are dimensionless things between 0 and 1, where a probability of 0 means something never-happens/is-impossible, and a probability of 1

[1]Davenport and Patil, in the Harvard Business Review, called the job of data scientist "the sexiest job of the 21st century" `https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century`

means something always-happens/is-certain-to-happen. We will also be working with probability density functions, which in general have the dimension of one over the variable they depend on, and vary from 0 to ∞. For example, a probability density function for a variable x which has units of metres, $p(x)$, is defined so that the probability that the variable x lies between x and $x + \mathrm{d}x$, is $p(x)\mathrm{d}x$. As $p(x)\mathrm{d}x$ is a probability, it is dimensionless and so $p(x)$ has dimensions of m^{-1} ($\mathrm{d}x$ has dimensions of m).

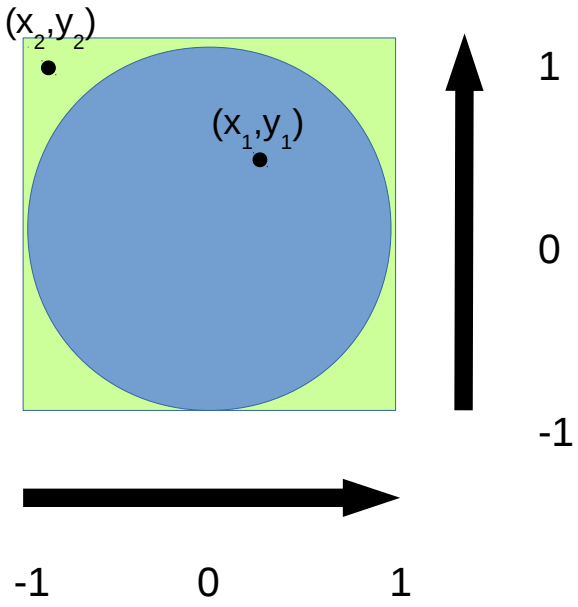

Figure 1: A square of side length 2, centred at the origin, with a circle of radius 1 inside it. Both are in the xy plane. Shown are two points inside the square in the xy plane. Each point is indicated by the vector position (x_i, y_i). One is inside the circle, and one is outside.

3 Monte Carlo Methods

The Monte Carlo (MC) method gets its name from the part of Monaco famous for its casinos. It is named after casinos because it involves a lot of random numbers. In practice it is not really one method, but a set of related methods. These methods are used all over physics, from particle and nuclear physics, to statistical physics. They are often used to model random processes.

For example, when ionising radiation is going through matter, the particles (α and β particles, and photons) scatter off, and are absorbed, by atoms. Both scattering and adsorption are random processes and so are often modelled using MC techniques. Particle, nuclear and medical physicists all use MC codes to calculate the interaction of radiation with matter.

Nuclear and particle physicists need to design detectors that absorb and so detect as many as possible of the particles produced (e.g., the particles produced by proton-proton collisions used at the LHC in CERN to detect the Higgs boson) while medical physicists want to design a radiotherapy protocol that delivers as high a dose as possible to the cancer tumour, and as little as possible to healthy tissue. There are MC codes available to do these calculations, called FLUKA, GEANT and SRIM. These are big complex codes, but Task 2 should give you an idea of the principles they use.

3.1 Monte Carlo Estimation of π

So often MC is used to model random processes but it can do a lot of other things as well. We will start with a simple classic example. We will use a simple MC method to produce an estimate for π. Note that this is just to show how MC methods work, there are *much* more efficient ways to calculate π. Of course, there is nothing random about π but we can use MC to calculate a probability that depends on π, and so estimate π.

To see how this works, look at the schematic in Fig. 1. This shows a square of side 2, and hence area 4, which has a circle of radius 1 in side it. The circle has an area of π. Both the square and the circle are centred at the origin, and both are in the xy plane.

To obtain an estimate for π, we proceed as follows. First, we generate a random number, x_1, uniformly between -1 and $+1$; we take this as our random position along the x axis, and inside the square. We then generate a second random number, y_1, between -1 and $+1$ and take this to be to our random position along the y axis. This pair of numbers, (x_1, y_1) is then a position vector for a point that is distributed at random within the square in Fig. 1.

We then put generating this position vector inside a do loop and generate a set of N random points inside the square: (x_1, y_1), (x_1, y_2), (x_3, y_3), ..., (x_N, y_N). See the *Random numbers* box for how we calculate random numbers on a computer.

Now the MC method for estimating π works as follows. The probability that a randomly selected point inside the square falls inside the circle, call it p_{circ}, is simply the ratio of the area of the circle to the area of the square

$$p_{circ} = \frac{\text{area of circle}}{\text{area of square}} = \frac{\pi}{4}$$

but for large N, p_{circ} is well approximated by the fraction of the N random points that fall inside the circle. If we write the number of points that fall inside rthe circle as N_{circ}, then

$$p_{circ} \simeq \frac{N_{circ}}{N}$$

Combining these two equations we get

$$\pi_{est} \simeq \frac{4N_{circ}}{N} \tag{1}$$

To calculate the right-hand side all we do is generate say $N = 1000$ points inside the square, using a do loop. Then for each point we work out if it is inside the circle, and so calculate N_{circ}. With values for N and N_{circ}, we can use the above equation and get π_{est}, which is our estimate for the value of π. Simple.

3.1.1 Coding the MC estimation of π

To generate the pairs (x_i, y_i) we need random numbers in the range -1 to $+1$. The random number generator `rand(iseed,first)` (see the *Random numbers* box), generates numbers between 0 and 1. We can use to this to generate a random number, x, uniformly distributed between -1 and $+1$, by using

$$\texttt{x} = 2.0 * (\texttt{rand}(\texttt{iseed}, \texttt{first}) - 0.5)$$

i.e., we subtract off -0.5 to shift the random number to the range -0.5 to $+0.5$ and multiply by two to expand the range to -1 to $+1$.

As the condition that a point (x, y) is inside a circle of radius 1 is that $x^2 + y^2 < 1$, we can use a simple `if` statement to calculate N_{circ}

$$\texttt{if}(\texttt{x} * *2 + \texttt{y} * *2 < 1.0)\texttt{n\_circ} = \texttt{n\_circ} + 1$$

i.e., if the point falls within the circle we increase the variable N_{circ} by one, and at the end of the do loop N_{circ} is the number of points inside the circle. Note that before we start the do loop we need to zero the variable N_{circ}, i.e., need the line `n_circ = 0`. Also, the $+1$ is there not $+1.0$ as are treating `n_circ` as an integer variable. We will need to divide it by `n` (another integer variable) to get a real number (π) and so need to use real not integer arithmetic. To do this real arithmetic we need `real(n_circ)/real(n)`. With values for N and N_{circ}, we can use the above equation and calculate π_{est}.

Random numbers

To implement MC algorithms we need long sequences of random numbers. In practice numerical physics does not use sequences of true random numbers, but sequences of what are called pseudo-random numbers. A sequence of pseudo-random numbers is actually deterministic not random in the sense that if you run the program twice you get the same sequence of numbers. However, the sequence depends on the value of a variable called 'seed', so you can change that variable and get a different sequence.

A pseudo-random-number sequence looks random in the sense that if you look at it you cannot distinguish it from a genuinely random sequence of numbers. For techniques like MC methods pseudo-random numbers are just as good as random numbers.

The simplest pseudo-random sequence is a sequence of N pseudo-random numbers uniformly distributed between 0 and 1. This is defined by

1. The numbers in the sequence of pseudo-random numbers must be uniform in the sense that the numbers are equally likely to fall in any small range of values between $x = 0$ and $x = 1$, i.e., for any values of x and δ, the average number of the numbers in the sequence in the range x to $x + \delta$, is the same.

2. They must be no correlations between numbers in the sequence, i.e., the value of one number in the sequence should tell you nothing about the value of the next number. For example, if one number in the sequence is close to 0, this should not mean that the next number is also close to 0.

Any sequence that satisfies these requirements can be used in MC programs. You can use the built in random number generators, as you did in the first year, but in practice in research, people almost always use routines to generate pseudo-random generators. Built in random generators can give results that vary from one compiler to another, and in some cases they are not perfectly random. Both these things are highly undesirable in research that uses MC codes.

Use of a real function random number generator You can download a function (within a small self-contained program) to generate pseudo-random numbers at:

http://personal.ph.surrey.ac.uk/∼phs1rs/teaching/ran_park.f95

there is a link to this on the SurreyLearn module.

The random number generator is a real function and has two arguments: `ran=rand(iseed,first)`. This is because a pseudo-random generator function can generate not just one sequence of pseudo-random numbers but many many sequences. Which sequence you get is controlled by the value of `iseed`, and `first` is how you start a sequence. To start a sequence set `first=0` to tell the function that you want to start a sequence, and then pick an integer and set `iseed` to that number, e.g., `iseed=917171`, and then so long as you don't try and change `first` and `iseed` again, the function will generate the sequence of numbers corresponding to `iseed=917171`. If then you want a different sequence, simply set `first=0` and then set `iseed` to a *different* integer. Note both `first` and `iseed` must be integer (not real) variables.

Random number generator for exponentially distributed random numbers This is for a uniformly distributed pseudo-random sequence, sometimes we want a sequence of numbers distributed according to a different probability density function, $p(x)$, e.g., $p(x) = \exp(-x/\lambda)/\lambda$. Here λ is the

mean value of the random number x. Note that this probability density function depends on one parameter: λ. Here, there again must be no correlations but now the sequence must generate numbers such that the probability of a number falling between x and $x + \delta$ is $\exp(-x/\lambda)\delta/\lambda$ and so is exponential not uniform.

You can download a function (within a small self-contained program) to generate pseudo-random numbers from an exponential distribution with mean `lambda` at:

http://personal.ph.surrey.ac.uk/∼phs1rs/teaching/ran_exp_park.f95

there is a link to this on the SurreyLearn module.

Task 1

Write a FORTRAN program to estimate the value of π. This should generate N points inside a square of side 2, as shown in Fig. 1. Then π is estimated by calculating the fraction of these N points that lie within a circle inside this square and of radius 1, and then using equation (1).

The program should write out the estimate for π, π_{est}, the error, i.e, $\pi_{est} - \pi$ and the number of points used, N. This number, N, should be a user input into the program. If you increase N you should see that the error decreases as in Figure 2.

To generate the random numbers you can use the random number generating function which you can download using a link on the SurreyLearn module for EENP. This function can be cut and pasted from the program you download into your program. The top part of the program you download shows you how to use the function.

3.2 MC estimation errors decrease slowly, as $N^{-1/2}$

As with essentially all numerical methods, MC methods do not calculate exact answers, they calculate approximate estimates. In fact MC methods give answers that are typically not that accurate, often there are uncertainties of at least a few %. This is because the error bars on quantities estimated using MC methods usually only decrease in size as $1/N^{1/2}$, which is a slow decrease: we need to multiply N by 100 to reduce the error by a factor of 10. This is a direct consequence of what is called the Central Limit Theorem (CLT) of statistics. See any statistics textbook, the Wikipedia page, etc, for definitions of the CLT. The CLT is part of the second year Energy and Entropy lectures as it is very important in statistical physics.

For the MC estimation of π, the error as a function of N is shown in Fig. 2. Note that we need N of order 1 million to get π with an accuracy of one part in a thousand. The error decreases with increasing N, just as the error in integrating an ODE decreases as the step size h decreases. However, the error in MC methods is statistical, i.e., is random. If you change `iseed` in your program then the error will change, it may get bigger or smaller. Although on *average* the error is approximately $1/N^{1/2}$, as it is statistical sometimes by chance it is a bit bigger and sometimes it is a bit smaller.

4 MC calculation of dynamics in a simple model of particles moving through matter

In this section we will look a simple model of particles moving through matter, and scattering off the atoms in the matter. We will write a small program that, in a very simple way, illustrates how programs

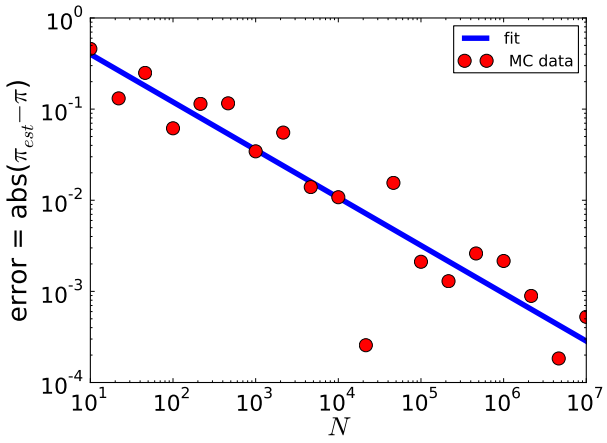

Figure 2: The absolute error in our estimate of π, i.e., $|\pi_{est} - \pi|$, plotted as a function of the number of random points inside the square we used to calculate the estimate, N. Note that the plot is a log-log plot because the data is a power law. The data is shown as circles, and a power law fit is shown as the line. The fit is $|\pi_{est} - \pi| = 1.34N^{-0.525}$. Note that the fit gives a value for the power of N near the CLT prediction of -0.5.

like FLUKA, GEANT etc model how radiation interacts with matter (the matter could be part of a particle detector, or a cancerous tumour).

Scattering of a particle, e.g., a photon, from an atom is a random process. This is illustrated schematicall in Figure 3. If the particle is uncharged then it may interact very weakly between scattering events, and then the distance travelled by the particle between scattering events is approximately exponentially distributed, i.e., it can be modelled by a random number drawn from an exponential distribution. The Fortran function `ran_exp_park.f95` (link on SurreyLearn module to download this function) will generate random numbers from an exponential distribution of width specified by `lambda`.

Thus, after a particle has scattered, for example, 10 times, then the distance travelled is the sum of 10 random numbers, each drawn from an exponential distribution of some width `lambda` (a real number), which is the mean distance travelled by the particle between scattering events. Note that for simplicity we are treating scattering in one dimension, whereas in reality it occurs in three dimension.

So to estimate the distance travelled after n_{scatt} scattering events, we can start off with a distance variable equal to zero, and then go round a do loop n_{scatt} times, each time incrementing the distance by a random amount drawn from an exponential distribution, using the line

```
distance = distance + ran_exp(iseed, first, lambda)
```

The do loop going over the n_{scatt} scattering events can then be nested inside an averaging do loop which calculates the distances travelled in n_{scatt} scattering events, N times, i.e.,

```
do i = 1, n     ! averaging do loop
  do i = 1, n_scatt     ! do loop over scattering events for 1 particle
  enddo
enddo
```

The average distance travelled is the sum of these N distances, divided by N. So inside the outer loop but outside the inner do loop, you need a line like

```
sum_distance = sum_distance + distance
```

and then finally the average distance travelled is (from its definition) the sum divided by N

$$\texttt{ave\_distance} = \texttt{sum\_distance}/\texttt{real}(\texttt{n})$$

Note that you should not mix reals (like `distance`) and integers (like `n`), in an arithmetic expression such as division. Therefore, you should use the Fortran command `real` which converts the integer `n` to a real number, which is then used in the division.

To calculate the standard deviation of the total distance travelled you need to work out the average of the square of the distance — which you can do in the same way as you do for the average distance. Then the standard deviation can be obtained from its definition as the square root of the mean of the square minus the square of the mean.

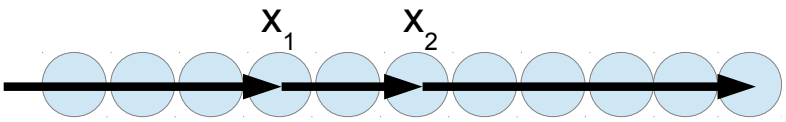

Figure 3: Schematic of a 1D model solid, illustrated by a 1D lattice of atoms (shown as circles), plus particle that is being scattered by the atoms. In its way through the particle scatters twice inside the slab, x_1 and x_2. The particle trajectories between scattering events are indicated by arrows. Note that this schematic is in 1D whereas real solids are three dimensional of course.

Task 2

Write a program to estimate the average distance travelled, for a particle that scatters elastically n_{scatt} times. Also calculate the standard deviation of the distance. As it is elastic scattering, no energy is lost and we only need to consider how far the particle travels. The distance travelled between scattering each event is a random variable drawn from an exponential distribution of width λ. To calculate these averages you should use a user-specified number of particles, N, and n_{scatt} should also be user specified.

The value of λ does not matter here, as it just scales the total distance travelled. You can just set it to 100 nm — in reality λ depends very sensitively on what particle it is (e.g., neutron), the energy of the particle, and the material it is passing through.

You should find that the average value is approximately $n_{scatt}\lambda$ with an MC error of approximately $n_{scatt}\lambda/(Nn_{scatt})^{1/2}$. You should write both out to the screen, to allow you to check that your MC estimate is within $n_{scatt}\lambda/(Nn_{scatt})^{1/2}$ of $n_{scatt}\lambda$.

Finally, determine how the ratio of the standard deviation of the distance travelled, to the average distance travelled, varies with the number of scattering events, n_{scatt}.

5 Bayesian statistics

In a physics experiment you are usually trying to measure a physical quantity as accurately as possible. This might be, for example, the mean lifetime τ of a radioactive isotope. It could also be one of millions of other measurable quantities: the mass of a star, the mass of a fundamental particle etc etc. Typically, at the start of the experiment you have a rough idea of the lifetime τ, and you want to use the results of your experiments to improve upon this initial rough guess. You want to not only obtain an accurate estimate of the value of τ, but also to obtain an error estimate. This error estimate should give you as much information about the uncertainty in your estimate for τ as possible.

Bayes' Theorem is one of the best ways of incorporating new knowledge into an existing estimate of the value of a variable so that our uncertainty in this value is reduced by the new data. This idea is essentially something we use informally all the time but without the maths. For example, if I tell you that X is a footballer then the probability that they are a forward is about 2/11 (assuming a 4-4-2 formation) but if I tell you that they scored 20 goals last season then presumably the footballer is either a forward or an attacking midfielder and so now you would update your estimate and say that the probability that they are a forward is about 1/2. Bayes' theorem takes this basic idea and converts it into an equation.

We are going to learn how to use Bayes' Theorem via a simple example.

5.1 Example problem: Estimating the lifetime of an unstable isotope

If a radioisotope has a lifetime[2] of τ s, the probability that after t s, the isotope has *not* decayed is

$$P(t) = \exp\left(-t/\tau\right)$$

and the probability that the isotope survives to a time t s, and then decays between t and $t+\mathrm{d}t$ s is $p(t)\mathrm{d}t$. This defines $p(t)$, which is essentially the instantaneous probability density of the isotope decaying at time t. $p(t)$ is just the negative of the derivative of $P(t)$, i.e.,

$$p(t) = -\frac{\mathrm{d}P(t)}{\mathrm{d}t} = \tau^{-1}\exp(-t/\tau)$$

As $p(t)$ is a probability density function, it is normalised, i.e., its integral over all times equals 1

$$\int_0^\infty p(t)\mathrm{d}t = 1$$

this is just because the decay happens with probability 1, between time equals zero and infinity.

5.1.1 Conditional probabilities

We now need to introduce the idea of a conditional probability or conditional probability density. Conditional probabilities are the probability of one thing, given that another thing is true, e.g., probability of A given B is written $p(A|B)$ – note the | between the A and B, and the order of A and B. Here, we note that the probability density of a decay occurring at time t is a conditional probability as it depends on τ, so we note that and write

$$p(t|\tau) = \tau^{-1}\exp(-t/\tau)$$

which is the probability density for the decay of a radioisotope at a time t, given that the lifetime of the isotope is τ.

[2]The isotope's half life, $t_{1/2}$, is related to its lifetime τ, by $t_{1/2} = (\ln 2)\tau$

The point here is that $p(t|\tau)$ is the wrong way round for us. It tells us how probable it is that we will observe a decay at a time t, *providing* we know the lifetime. But in experiment we don't measure τ, we measure t and then want to use those measurements to estimate the value of τ. What we want is not $p(t|\tau)$ but $p(\tau|t)$. Bayes' theorem will allow us to estimate this.

Now that we know what conditional probabilities are, we need to look at how we can make our initial rough estimate for the value for τ into an equation.

5.2 Starting estimate: the Prior probability density p_0

At the start of the experiment, before making any measurements, we must have some idea of what the true value, τ_{TRUE}, of the lifetime is. If we had no idea what the value of τ_{TRUE} is we could not design an experiment to estimate its value as we would not know if our detector needed a time resolution of nanoseconds, seconds, days, millenia, etc.

Our rough estimate of the value of τ, the one we have at the start of the experiment is expressed by what is called a prior probability density function, $p_0(\tau)$. This is our best guess for the probability density function at the start. p_0 is often just called the prior.

For example, we may think that τ is somewhere in the range τ_{MIN} to τ_{MAX}, i.e., greater than τ_{MIN} but less than τ_{MAX}. Then if we know nothing more than that we can start with a prior probability density function of

$$p_0(\tau) = \begin{cases} 0 & \tau < \tau_{MIN} \text{ s} \\ \frac{1}{\tau_{MAX}-\tau_{MIN}} \text{ s}^{-1} & \tau_{MIN} \leq \tau \leq \tau_{MAX} \text{ s} \\ 0 & \tau_{MAX} \text{ s} < \tau \end{cases},$$

This is uniform from τ_{MIN} to τ_{MAX} s and zero outside that range. In this range $p = 1/(\tau_{MAX} - \tau_{MIN})$ s^{-1} because the integral over p_0 must equal one as it is a probability density function. We have taken p_0 to be uniform between τ_{MIN} and τ_{MAX}, because in the absence of any more information, we have no reason to make the probability density of τ higher in one part of the range, than in another.

5.3 Bayes' Theorem

If our first measurement of a radioisotope decay is at a value $t = t_1 = 8.954$ s, then what we want is the probability $p(\tau|t_1)$: the probability that the isotope lifetime equals τ, given that we have observed a decay at a time t_1. This is given by Bayes' Theorem, which is

$$p(\tau|t_1) \propto p(t_1|\tau)p_0(\tau) \qquad \textbf{Bayes' Theorem}$$

In words: after a decay at a time $t = t_1$, the probability that the lifetime has the value τ is proportional to the probability that it had the value τ before we made the measurement (the prior $p_0(\tau)$), times the probability that a source with lifetime τ would give a decay at $t = t_1$. For the background to this equation, you can look up the Wikipedia page on Bayesian Inference, or one of many statistics textbooks. Many textbooks discuss Bayesian statistics.

The prior probability p_0, and $p(\tau|t_1)$ are shown in Fig. 4. Note that the prior is just a horizontal line as it is constant, while $p(\tau|t_1)$ has a peak at $\tau \approx 10$. So, after the measurement, we know that the lifetime is more likely to be near 10, than near 100, but there is still considerable uncertainty in the true value of τ. To reduce this uncertainity, we do further measurements.

If we make successive measurements of decay times at $t_1, t_2, t_3, \ldots, t_n$, i.e., we have a set of measurements $\{t_i\}$, then Bayes' Theorem is

$$p\left(\tau|\{t_i\}\right) \propto p_0\left(\tau\right) \Pi_{i=1,n} p(t_i|\tau) \qquad \textbf{Bayes' Theorem – multiple measurements}$$

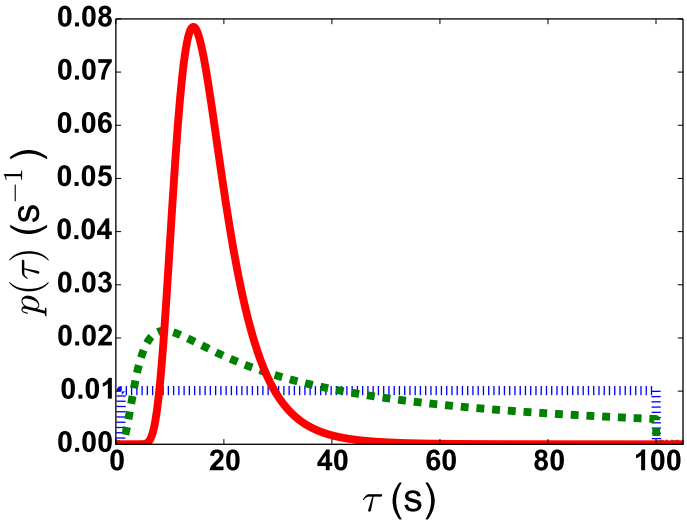

Figure 4: Plots of the probability distribution function for the estimated lifetime τ of a radioactive isotope. The dotted curve is $p_0(\tau)$, the prior probability distribution function, that before any measurements are taken. The dashed curve is after one measurement has been taken and this observed a decay at a time $t_1 = 8.954$ s. The solid curve is after ten measurements have been taken.

Note that $\Pi_{i=1,n}$ indicates a product, i.e., we multiply all the $p(t_i|\tau)$ together. Introducing a proportionality constant c we have that

$$p\left(\tau|\{t_i\}\right) = cp_0\left(\tau\right)\Pi_{i=1,n}p(t_i|\tau)$$

and c is obtained by noting that the integral over $p\left(\tau|\{t_i\}\right)$ must equal one. This is the basic equation we need to convert into Fortran to allow us to do Bayesian statistics on a computer. The result $p\left(\tau|\{t_i\}\right)$ for a set of 10 measurements of decay times is plotted in Fig. 4. Note that the new probability density function for τ is much more sharply peaked than after 1 measurement: the more measurements we make the more information we have and the more accurate is our estimate for the lifetime. But there is always some uncertainty in the true value, i.e., there will always be error bars.

5.3.1 Normalising a probability density function

Often we are in the situation that we have computed a function, $p'(x)$, that is almost a probability density function but is not normalised, i.e., the true probability density function $p(x) = cp'(x)$, where c is an unknown constant. Normalising $p'(x)$ to get $p(x)$ takes just two steps, first integrate p' to obtain the integral I. Then just divide p' by I to get p, i.e., $p(x) = p'(x)/I$.

Integration is easy to to do on a computer. If in Fortran we write the integral I, as `norm_const`, and if the function $p'(x)$ is in the array `p`, then

$$\texttt{norm\_const} = (\texttt{sum(p)} - 0.5 * (\texttt{p(0)} + \texttt{p(m)})) * \texttt{dx}$$

where the array `p` has elements `i=0` to `m`. Also `dx` is the spacing along the x axis between successive elements in the array `p`. So this is the first step. The built-in Fortran `sum` takes an array and sums up all its elements.

The second step is just

$$\texttt{p} = \texttt{p/norm\_const}$$

and then p is a normalised probability density function (in an array). Note that we coud have defined two arrays p and pdash, and have pdash as the unnormalised one and p as the final normalised one. That would also work.

5.4 Coding Bayes' Theorem

Here we will see how to write a program to take an array decay_meas_t with 10 measured decay times, and use it to obtain a probability density for the decay time, $p(\tau)$. These 10 measure decay times are in a file you can download from SurreyLearn onto your computer. Your program should then read these 10 numbers in from the file.

We will need to discretise the τ axis in order to represent the functions $p_0(\tau)$ and $p(\tau|\{t_i\})$ by one-dimensional Fortran arrays prior and p. We will also need a corresponding array of τ values: tau_arr. In our case the prior will be zero outside the range from τ_{MIN} =1 to τ_{MAX} = 100 s, so a suitable step to discretise the τ axis is dtau=0.1. Then if we define arrays of size 0 to m=1050

$$\texttt{integer}, \texttt{parameter} :: \texttt{m} = 1050$$
$$\texttt{real} :: \texttt{prior}(0:\texttt{m}), \texttt{p}(0:\texttt{m}), \texttt{tau\_arr}(0:\texttt{m})$$

these will span the range $\tau = 0$ to 105 s – the range from τ_{MIN} to τ_{MAX} plus some extra at either end. With dtau=0.1, the array tau_arr has elements tau_arr(i)=real(i)*dtau.

The prior array should be set, which means you put the values you need to, into the array. You can then input the prior straight into the array p with

$$\texttt{p} = \texttt{prior}$$

Now you are ready to improve the array p using Bayes' Theorem. To update the array element i in array p for the point at tau_arr(i), we need

$$\texttt{p(i)} = \texttt{p(i)} * \texttt{exp}(-\texttt{decay\_meas\_t(j\_meas)}/\texttt{tau\_arr(i)})/\texttt{tau\_arr(i)}$$

Now looping over i updates the whole array p for one measured decay time, decay_meas_t(j_meas). Looping over j_meas then updates the array p for all 10 measurements. In other words you need 2 nested do loops.

Finally, all you then need to do is normalise the array p. This array is the probability distribution function for the decay time τ, that incorporates the information provided by the 10 measured decay times.

Assignment 1

Write a program to estimate the probability density function for the lifetime τ, of an isotope, $p(\tau|\{t_i\})$, after a set of 10 decay times have been measured. The 10 decay time measurements are in the file bayes_data.txt on the EENP SurreyLearn module. You will need to read the times into an array of length 10, decay_meas_t.

The probability density function should be obtained as a one-dimensional array with steps along the τ axis of size 0.1. It should cover the range of the τ axis from $\tau = 0$ to $\tau = 105$ s. Therefore, you will need three one-dimensional arrays: one for τ, one for the prior p_0 and one for the probability p. You will also need a fourth array, of length 10, for the 10 measured decay times.

The prior is

$$p_0(\tau) = \begin{cases} 0 & \tau < \tau_{MIN} \\ \frac{1}{\tau_{MAX} - \tau_{MIN}} \text{ s}^{-1} & \tau_{MIN} \leq \tau \leq \tau_{MAX} \text{ s} \\ 0 & \tau_{MAX} \text{ s} < \tau \end{cases},$$

with $\tau_{MIN} = 1$ s and $\tau_{MAX} = 100$ s.

Once the program has determined $p(\tau|\{t_i\})$, find the most probable value of τ. This value should be output to the screen. The function $p(\tau|\{t_i\})$ should be written to a file as two columns, ready for plotting. The two columns should be the values of τ and $p(\tau|\{t_i\})$. Note that output real numbers should be formatted. Write the name of the file containing these numbers, to the screen.

You should check your program by plotting $p(\tau|\{t_i\})$, using gnuplot or other software, and it should look like the solid curve in Fig. 4.

N.B. When compiled with nagfor, on running you may get a warning: "Warning: Floating underflow occurred". You do not need to worry about this, and you will not lose marks for this. The calculation of $p(\tau)$ gives you values at some values of τ that are extremely small, so small that the program sets them to zero. This is not a problem.