Lecture 10:

Monte Carlo Methods

- Origin
- Sampling Distributions
- Markov Chains
- MC Integration
- Smart Sampling
- Weighted Sampling
- Three Mantras

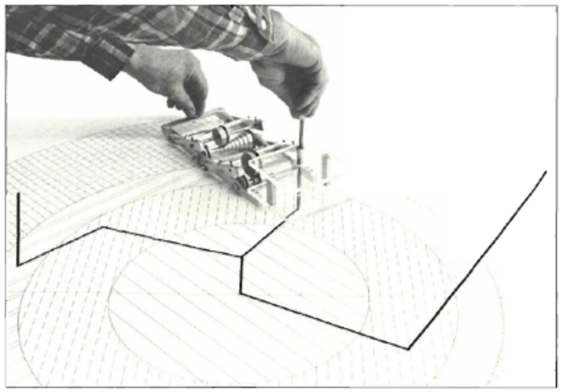
Monte Carlo Method

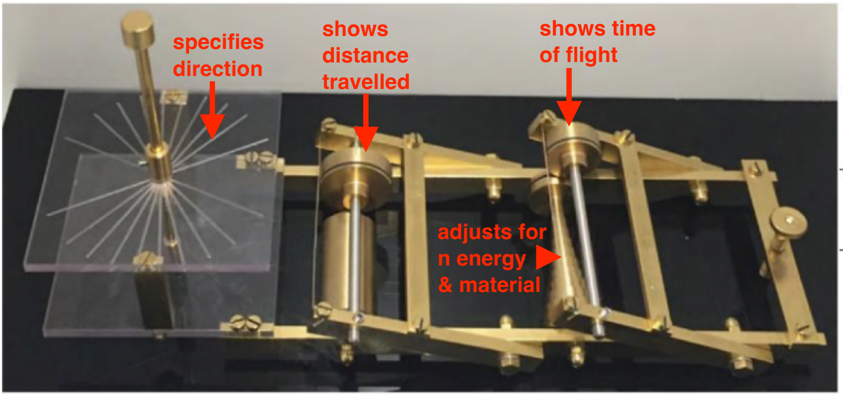
The “Monte Carlo Method” is an approach to numerical integration for complicated functions (or convolutions of functions) using random numbers to explore the allowed phase space.

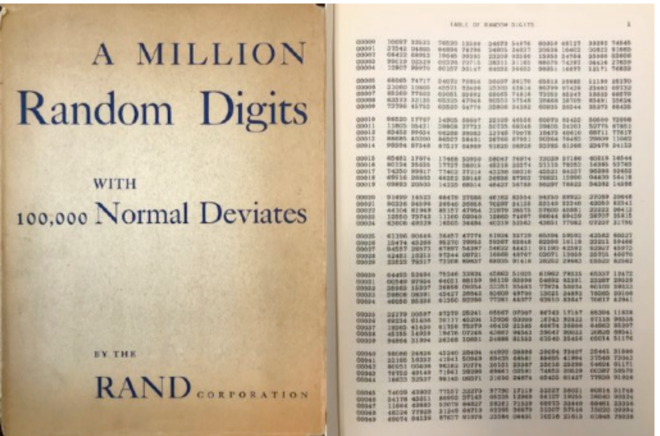
The modern Markov Chain version (for a linked sequence of random steps) was originally developed by Stanislaw Ulam and John von Neumann for the Manhattan Project in 1946 to study neutron diffusion in the core of a nuclear weapon.

Just prior to successful implementation on ENIAC, Fermi came up with a novel analog method…

“Fermiac” (~1947)







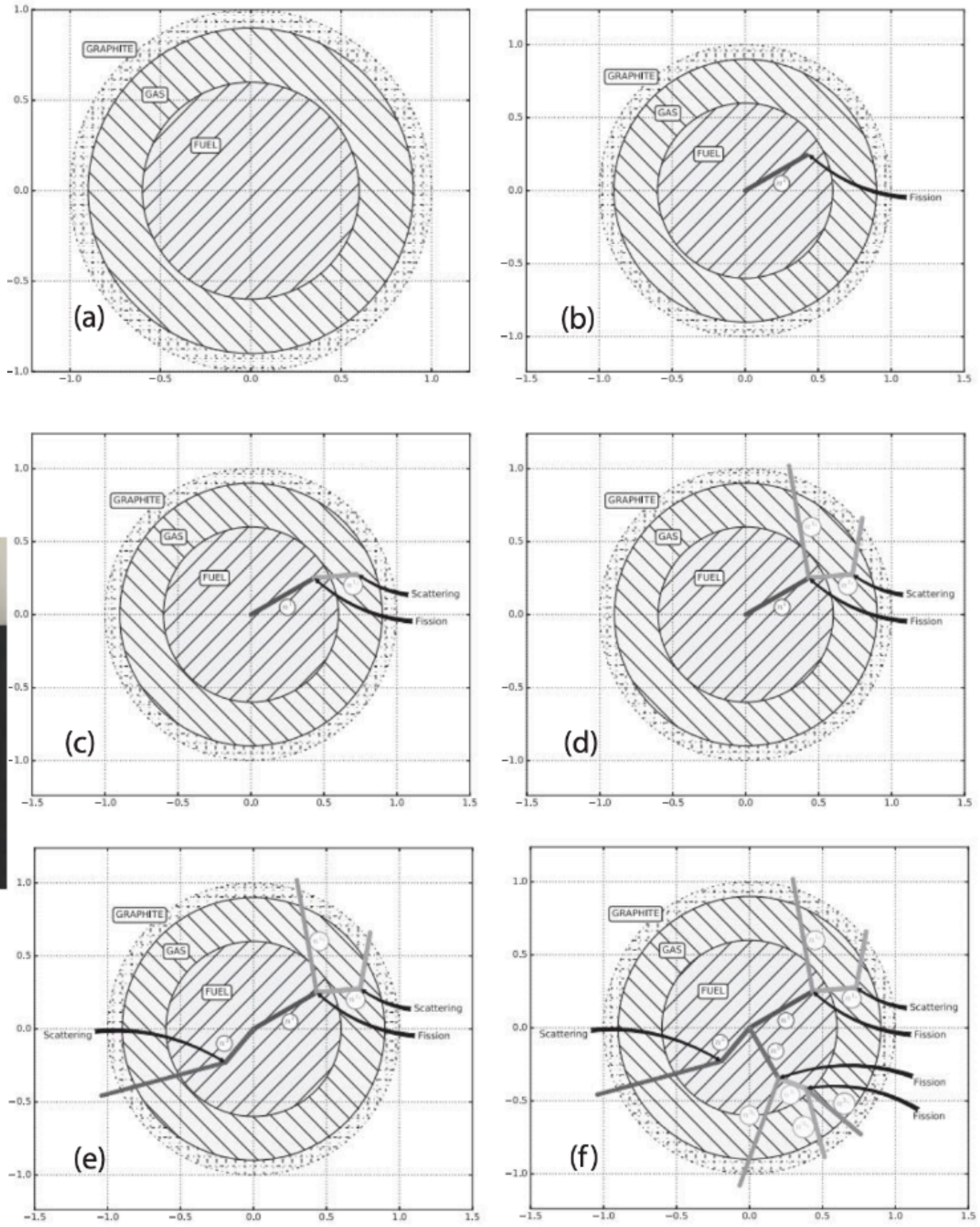


Fig. 6. – An example of the Fermiac used to follow the fate of the genealogy of source neutrons n^1, n^2, n^3 in a cell of a nuclear reactor.

You want to start with a good pseudo-random number generator. A common one is the “Mersenne Twister” and variants thereof. The random number should be between 0 and 1 exclusively (worth checking as some generators sometimes land on 0 or 1 exactly, in which case you should throw again!)

Note: Use a generator in which you explicitly set the initial pseudo-random number “seed” yourself! This allows you to repeat exactly the same sequence again to investigate any anomalies, as well as providing explicit control in generating an independent sequence.

The value of the random number will be used to choose where in a given distribution you should sample your random variable: closer to 0 will select from one end of the distribution, closer to 1 will select from the other. The random number thus acts as an integral probability and, therefore, you use this to choose a random variable (“deviate”) from the integral or “cumulative” probability distribution (CDF) for the parameter of interest.

There are a number of methods for sampling distributions…

1. Algebraic Transformation of CDF

Example 1: Take ***p*** to be a sampled random number. Say we want to sample the time ***t*** of a radioactive decay of some lifetime ***τ***. The differential decay probability is ***(1/τ)exp(-t/τ)*** and the integral probability is then ***1 - exp(-t/τ)***. We can therefore equate:

$$p = exp(-t/\tau) \quad \text{or} \quad \boxed{t = -\tau \ln p} \quad \text{easy!}$$

(obviously, it doesn't matter if we equate this to ***p*** or ***1-p***)

Example 2: Say we want to sample a position uniformly from within a spherical volume. The differential volume element for integration up to some set of spherical coordinate values can be written as:

$$dV = r^2\, dr\, d(\cos\theta)\, d\phi$$

The parameters are orthogonal, so throw 3 random numbers and sample based on the separately integrated elements:

r from a cubic distribution: $r = R_{max} p_1^{1/3}$

cosθ uniformly between ***-1*** and ***1***: $\cos\theta = 1 - 2p_2$

φ uniformly between ***0*** and ***2π***: $\phi = 2\pi p_3$

this part can be used to sample an isotropic direction

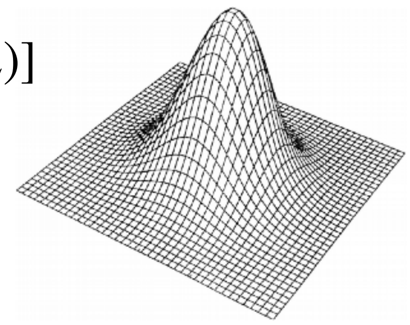
Example 3: Say we want to sample from a 1-D Gaussian centred on zero with unit variance:

$$G = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2)$$

The integral is an error function, which does not have a nice closed form that is easy to invert.

However, consider the integral of a 2-D Gaussian in polar coordinates:

$$\frac{1}{2\pi}\int_0^{\theta}\int_0^{r} \exp(-r'^2/2)\; r' dr' d\theta' \quad = \frac{\theta}{2\pi}[1 - \exp(-r^2/2)]$$



So, throw 2 random numbers and sample according to:

$$p_1 = \exp(-r^2/2) \longrightarrow \quad r = \sqrt{-2 ln p_1}$$

$$p_2 = \frac{\theta}{2\pi} \longrightarrow \quad \theta = 2\pi p_2$$

(Box-Muller Transform)

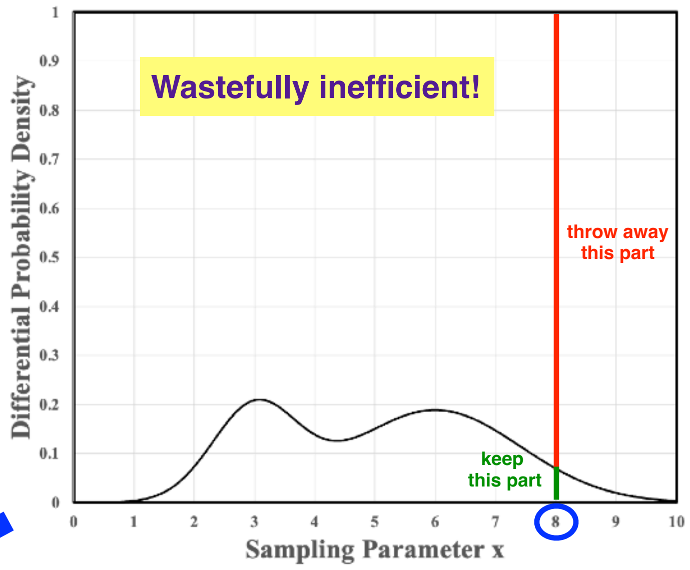
Now convert to orthogonal Cartesian coordinates for two independent Gaussian deviates:

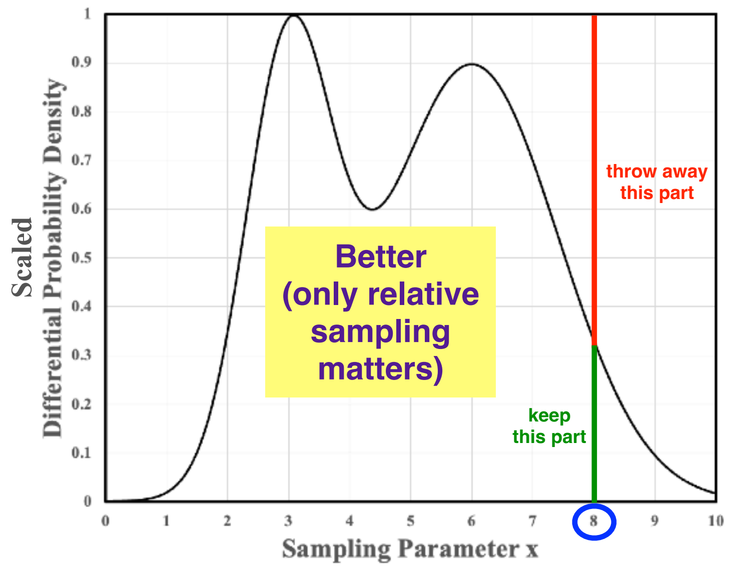
$$x = r\cos\theta \qquad y = r\sin\theta$$

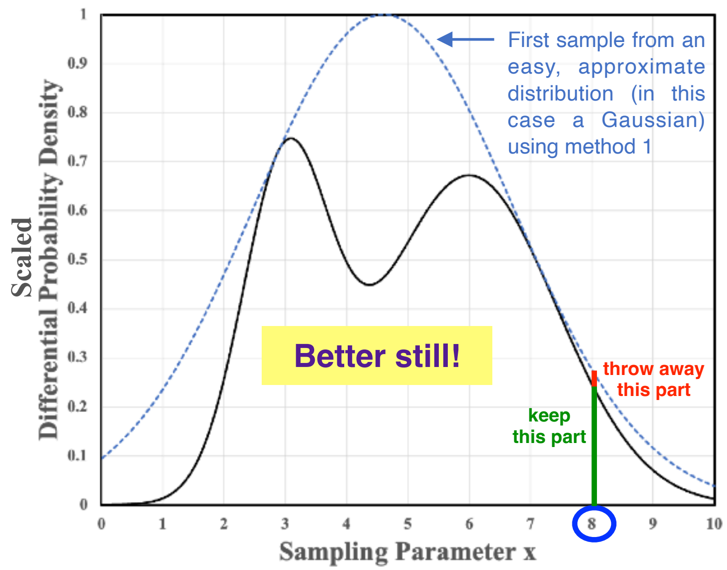
Use one now, save the other for next time! To sample from a distribution with a different variance and position, multiply the deviates by $\boldsymbol{\sigma}$ and add an offset

2. Sample and Reject

When the function does not have a form for the CDF that can be easily inverted, you can randomly choose a value for the sample variable of interest, and then throw another random number to decide whether to accept or reject this based on the differential probability. If rejected, sample again:





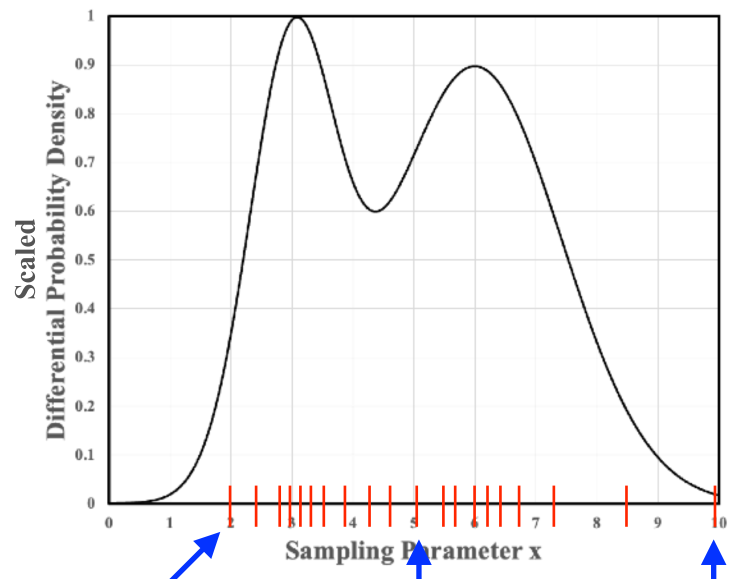


(Poisson samplers generally work this way)

3. Precomputed Percentiles

In many cases, you can pre-compute sampling parameter CDF values (either analytically or numerically) in, say, 0.1% percentile steps and store these in an ordered array of 1000 values. Then simply throw a random number between 1 and 1000, pick the nearest bin and retrieve the parameter value. For finer granularity, you can also linearly interpolate between bins, as well as using finer percentile steps. This is a good trick for making things very fast, even when you can also use methods 1 or 2 !

Simplified case of 5-percentile steps:



Choose x_i by randomly sampling i from 1-20.

$P(<x_1) = 0.05$ $P(<x_{10}) = 0.5$ $P(<x_{20}) = 1$

Markov Chain Monte Carlos used a linked sequence of random deviates to sample trajectories in the overall phase space so as to preserve correlations. Each individual sampling in a given "chain" is only dependent on the previous step to produce a correlated sequence.

Sampling for Competing Processes

1. Sample for the combination of processes, then throw another random number to decide which process occurred
2. Throw random numbers for each competing process, and then take the one that "happens first"

Example: assume we want to sample the next interaction point, in which a particle can either be absorbed or scattered, and then take the appropriate action:

$$P(a+s) = e^{-\frac{x}{\lambda_a}} e^{-\frac{x}{\lambda_s}} \quad = e^{-x\left(\frac{1}{\lambda_a}+\frac{1}{\lambda_s}\right)} \equiv e^{-\frac{x}{\Lambda}}$$

$$x = -\Lambda \ln p_1$$

$$P(a\,|\,x) = \exp\left[-x\left(\frac{1}{\lambda_a} - \frac{1}{\Lambda}\right)\right]$$

Throw another random number p_2 and choose absorption if it's greater than this value, otherwise choose scattering

OR

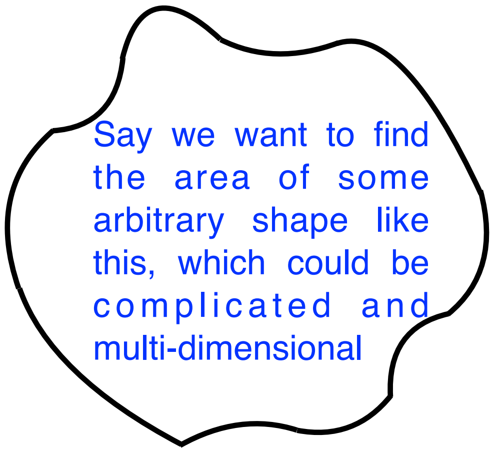
$$x_a = -\lambda_a \ln p_1$$

$$x_s = -\lambda_s \ln p_2$$

Then choose whichever x value is shortest, along with the associated interaction type

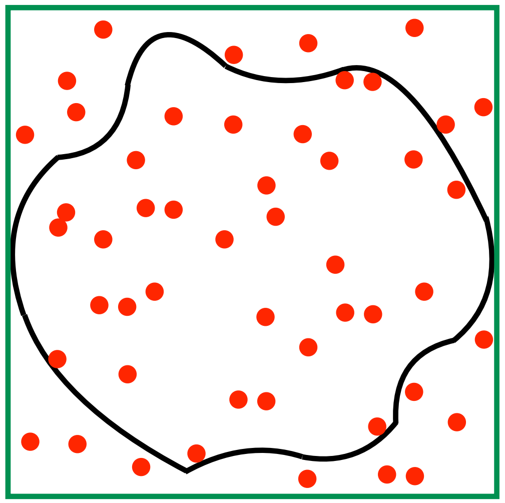
A Simple Example of MC Integration

(non-Markov)



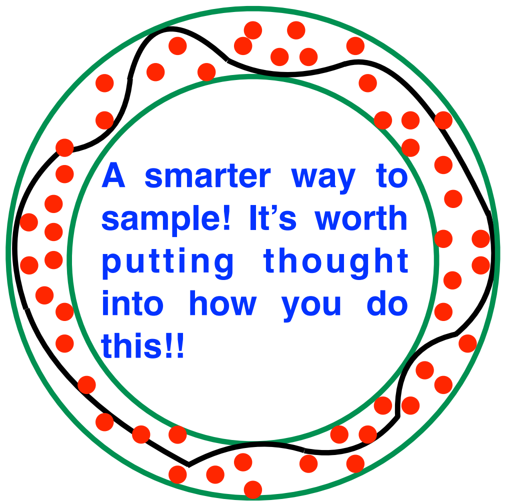
Say we want to find the area of some arbitrary shape like this, which could be complicated and multi-dimensional

You could generate random points in the area/volume and simply count the fraction that land within the boundaries of the function



In this simple example, the accuracy of the integration is ~18% for 50 random samples

Accuracy of the integration goes as ~1/√n, so you may need to throw a lot of random numbers!



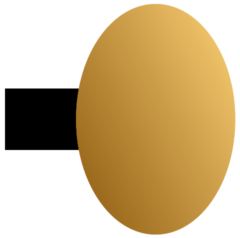
A smarter way to sample! It's worth putting thought into how you do this!!

Accuracy ~5.5% for the same number of samples!

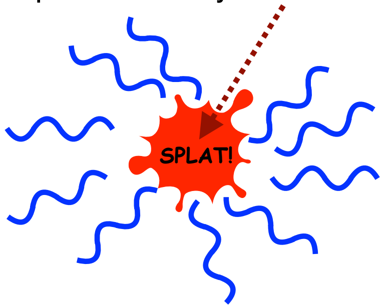
Another Example: MC integration to find mean # of photons observed for a particular interaction

Photodetector with **20%** efficiency

$$P(n) = \frac{\mu^n e^{-\mu}}{n!}$$



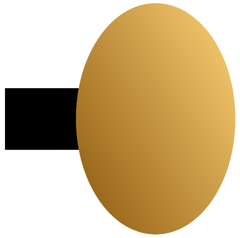
Generate photons as a Poisson fluctuation from an average of **μ** and repeat for many interactions



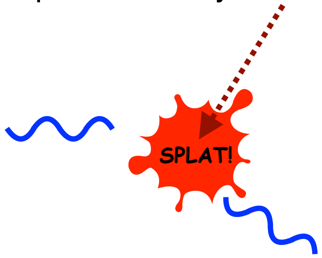
IS EXACTLY EQUIVALENT TO

Photodetector with **100%** efficiency

$$P(n) = \frac{\mu^n e^{-\mu}}{n!}$$



Generate photons as a Poisson fluctuation from an average of **μ/5** and repeat for many interactions



5 times fewer photons to track!!

What about the event-by-event fluctuations in the observation?

Exactly the same! Poisson counting statistics only cares about the mean observed number of photons, which is identical!

For functions with n dimensions, the required number of random samples to achieve a given accuracy goes as the power of n, which can create headaches!

$$\Big\downarrow$$

Factorise whenever possible, using symmetries and dependencies to choose parameters that are independent enough that the problem can be reduced to fewer dimensions.

Example:

Say you're using MC integration to derive a PDF for some class of interactions as a function of energy, scattering angle and reconstructed position in the detector:

$$P(E, \theta_{scatt}, x, y, z)$$

Ok, perhaps the detector is approximately spherical, so we can reduce this to:

$$P(E, \theta_{scatt}, r)$$

Assume that the position reconstruction is very weakly dependent on energy for the events of interest and has no correlation with the scattering angle. Then this might be factorised as:

$$P(E, \theta_{scatt}) \times P(r)$$

Let's say kinematics tells us that the mean scattering angle is inversely proportional the energy, in which case we may be able to further factorise this as:

$$P(E) \times P\left(\frac{\theta_{scatt}}{E}\right) \times P(r)$$

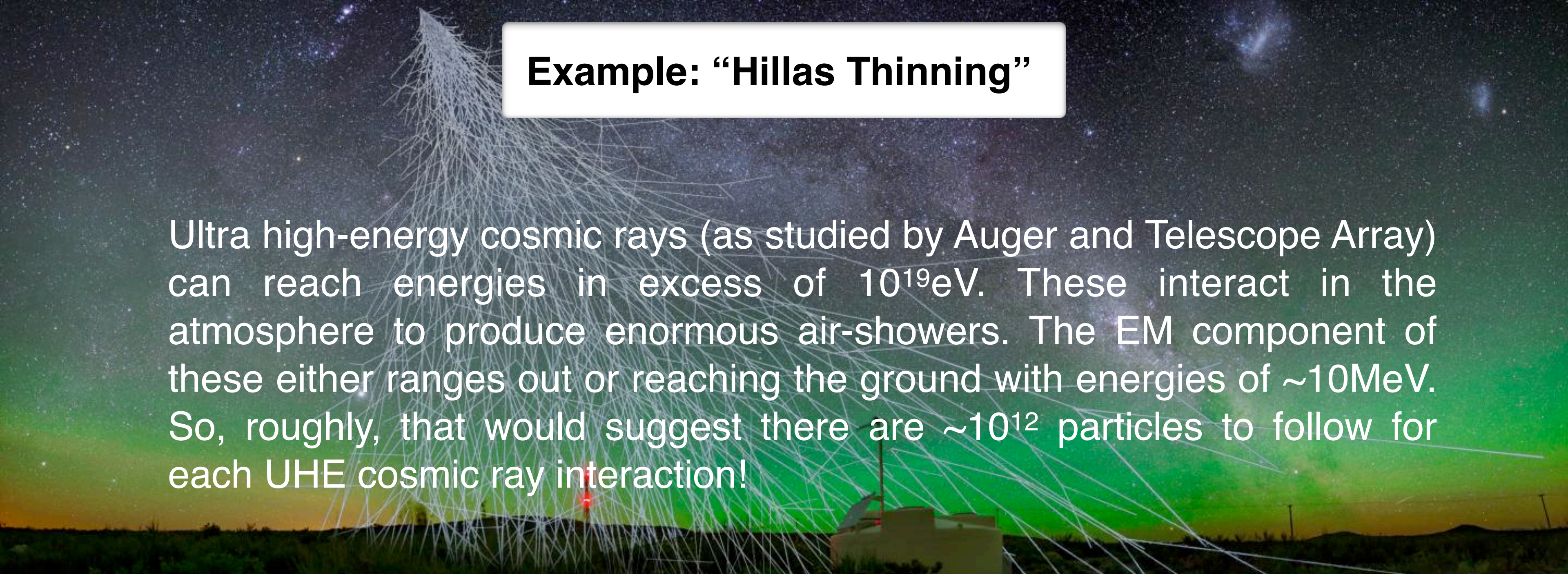
Note: even if not factorable, this is probably a better choice, since reducing parameter dependencies allows you to focus more on the relevant new information content and pick an appropriate PDF binning to optimise this

Weighted Sampling

There are cases where the scale of a problem is such that following every detail in a set of Markov chains is computationally challenging. However, it is not always necessary to do so in order to derive representative distributions, so long as a sufficiently representative (*i.e.* unbiased) sampling has taken place.

One could do this by simply using fewer random samplings and launching fewer Markov chains from the start, but then rare or “defining” processes of interest are unlikely to be sufficiently sampled. One would ideally like to reduce sampling for processes that are well represented, and have higher sampling for more rare process of interest, while still maintaining the correct probabilistic context for these. This is the idea behind weighted sampling.

Weighted sampling trades off between the variances: increasing these for processes that are already well sampled (so that their mean distributions are still well defined), while reducing variances for more sparsely sampled processes to better define their mean behaviours. “Event-by-event” variances are therefore not preserved, but unbiased representative distributions can still be produced.



Example: “Hillas Thinning”

Ultra high-energy cosmic rays (as studied by Auger and Telescope Array) can reach energies in excess of 10^{19}eV. These interact in the atmosphere to produce enormous air-showers. The EM component of these either ranges out or reaching the ground with energies of ~10MeV. So, roughly, that would suggest there are ~10^{12} particles to follow for each UHE cosmic ray interaction!

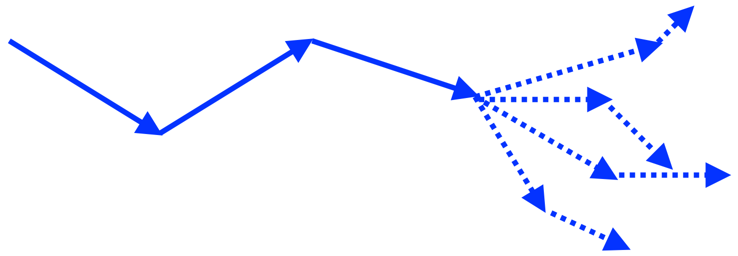
Michael Hillas (Leeds) instead introduced the following scheme:

1. **Select a “demarcation energy,” D, perhaps $D=10^{-4}E_{primary}$. Follow all particle with energy E>D.**
2. **Particles with E<D are subject to a selection test when they are produced, such that they are only retained with a probability p=E/D.**
3. **Each particle retained is given a weight w = 1/p, thus allowing them to also represent those particle not followed. So, if only 5% of the particles are followed, each one is given a weight of 20 when subsequent distributions are tabulated etc.**
4. **When a particle with weight w>1 itself interacts, its secondaries are retained with a probability $p'=E/E_{collision}$, and each one retained is give the weight w’ = w/p’**

In this way, the processes that most define the individual shower characteristics are sampled in detail, while weighted averages are used for the very numerous particles in the lower energy component

“Splitting”

The flip side of thinning is to increase sampling further for regions of parameter space where you would really like to resolve more detail with higher accuracy. In this case, for Markov chains that approach this region of parameter space (*e.g.* maybe pass through some particular geometric region of your detector), you can increase the sampling by duplicating particle tracks that are each given a weight <1 and followed as new, individual sub-chains.



So, for example, you could reduce your computed variance in this region by increasing the effective sampling by a factor of 10, with each ‘split’ particle given a weight of 1/10 to preserve the correct probabilistic context in subsequently tabulated distributions.

This obviously increases the computational burden, but this can be compensated for by thinning elsewhere. This is sometimes referred to as “Splitting vs Russian Roulette.” By adjusting the balance, the focus of MC integration can be appropriately tuned.

Three Mantras…

Simple MC calculations for well-defined mathematical processes are incredibly useful, particularly for understanding the probability calculations and the performance of data analysis schemes.

Large-scale Markov chain “simulations” can be extremely complicated. If done right, they are good tools for interpolation and limited extrapolation. They often appear to mimic data, but it’s important to remember that they are not data!! They always make approximations and (almost certainly) contain bugs of one sort or another. A good mantra is:

MC simulations are ALWAYS wrong, it’s just a question of how much!

Consequently, **don’t “think” with Monte Carlos!** Use them to refine details of what you’ve already worked out from simpler arguments. If the results throw up something unexpected or something for which you do not have an intuitive understanding, use it as a tool to gain that understanding… or to find a bug in the code!

This leads me to my final parting mantra:

TRUST NOBODY (including yourself)!!