# More on Fixing Random

In this lesson, we sum up some of the content we have covered in this course and provide more food for thought regarding fixing random.

WE'LL COVER THE FOLLOWING

## Summing Up #

Let's sum up the last few lessons:

Suppose we have a distribution of `doubles`, `p`, and a function `f` from `double` to `double`. We often want to answer the question "what is the average value of `f` when it is given samples from `p`?" This quantity is called the expected value.

The obvious (or "naive") way to do it is: take a bunch of samples, evaluate the function on those samples, take the average. Easy! However, this can lead to problems if there are "black swans": values that are rarely sampled, but massively affect the value of the average when running through `f`. We want to get a good estimate without having to increase the number of samples in our average massively.

## Techniques to Estimate Expected Value: #

We developed two techniques to estimate the expected value:

### Abandon Sampling and Perform Numerical Integral Calculus

## Abandon Sampling and Perform Numerical Integral Calculus

- Use quadrature to compute two areas: the area under `f(x) * p.Weight(x)` and the area under `p.Weight(x)` (which is the normalization constant of `p`).
- Their quotient is an extremely accurate estimate of the expected value.
- But we have to know what region to do quadrature over.

## Use Importance Sampling

- Find a helper distribution `q` whose weight is large where `f(x) * p.Weight(x)` bounds a lot of areas.
- Use the naïve algorithm to estimate the expected value of `x => f(x)*p.Weight(x)/q`. `Weight(x)` from samples of `q`.
- That is *proportional* to the expected value of `f` with respect to `p.`
- We gave a technique for estimating the proportionality constant by sampling from `q` also.

### Problems with Importance Sampling

The problem with importance sampling then is finding a good `q`. We discussed some techniques:

- If you know the range, use a uniform distribution over that range.
- Stretch and shift `p` so that the transformed PDF doesn’t have a “black swan”, but the normalization constant is the same.
- Use the **Metropolis Algorithm** to generate a helper PDF from `Abs(f * p)`, though in our experiments this worked poorly.
- If we know the range of interest, we can use the VEGAS algorithm. It makes cheap, naïve estimates of the area of subranges, and then uses that information to gradually refine a piecewise-uniform helper PDF that targets spikes and avoid flat areas of `f * p`.
- However, the VEGAS algorithm is complex, and we did not attempt to implement it for this course.

## Food for Thought

The question you may have been asking yourself these past few lessons is:

**If quadrature is an accurate and cheap way to estimate the expected**

**value of `f` over samples from `p` then why are we even considering doing sampling at all? Surely we typically know at least approximately the range over which `f * p` has some area. What's the point of all this?**

A simple quadrature algorithm just splits up the range into some number — say, a thousand — equally-sized pieces, evaluates `f * p` on each of them, and takes the average. That sure seems cheaper and easier than all this mucking around with sampling. Have we just been wasting your time these past few lessons? And why has there been so much research and effort put into finding techniques for estimating expected value?

This course's main purpose is "Fixing Random" because the built-in base class library tools we have in C# for representing probabilities are weak. We've approached everything in this course from the perspective of "*We want to have an object that represents probabilities in our business domain, and we want to use that object to solve our business problems*".

"*What is the expected value of this function given this distribution?*" is a very natural question to ask when solving business problems that involve probabilities, and as we've seen, you can answer that question by simulating integral calculus through quadrature.

But, as we keep on saying: **things equal to the same are equal to each other**. Flip the script. Suppose our business domain involves *solving integral calculus problems*. And suppose there is an integral calculus problem that we *cannot* efficiently solve with quadrature. What do we do?

- We can solve expected value problems with integral calculus techniques such as quadrature.
- We can solve expected value problems with sampling techniques.
- Things equal to the same are equal to each other.
- Therefore **we can solve integral calculus problems with sampling techniques**.

That is why there has been so much research into expected computing values: the expected value is the area under the function `f(x) * p.Weight(x)` so if we can compute the expected value by sampling, then we can compute that area and solve the integral calculus problem without doing quadrature!

We said above "*if quadrature is accurate and cheap*", but there are many scenarios in which quadrature is not a cheap way to compute an area. (There are more complex quadrature algorithms that can be more efficient, but we will not discuss them in this course.)

What's an example? Well, let's generalize. So far in this course, we've assumed that `f` is a `Func<double, double>`. What if `f` is a `Func<double, double, double>` — a function from pairs of doubles to double. That is `f` is not a line in two dimensions; it is a surface in three.

Let's suppose we have `f` being such a function, and we would like to solve a calculus problem: what is the volume under `f` on the range $(0, 0)$ to $(1, 1)$?

We could do it by quadrature, but remember, in our example, we split up the range 0-to-1 into a thousand points. If we do quadrature in two dimensions with the same granularity of $0.001$, that's a million points we have to evaluate and sum. If we only have computational resources to make a thousand points, then we have to have a granularity of around $0.03$.

What if the function is zero at most of those points? We could then have a really bad estimate of the total area because our granularity is so low.

We now reason as follows: take a two-dimensional probability distribution. Let's say we have the standard continuous uniform implementation of `IWeightedDistribution<(double, double)>`.

All the techniques we have explored in this course work equally well in two dimensions as one! So we can use those techniques. Let's do so:

- What is the estimated value of `f` when applied to samples from this distribution?
- It is equal to the volume under `f(x,y) * p.Weight((x,y))`.
- But `p.Weight((x,y))` is always $1.0$ on the region we care about; it's the standard continuous uniform distribution, after all.
- Therefore **the estimated expected value of `f` when evaluated on samples from `p` is an estimate of the volume we care about**.

How does that help?

It doesn't.

If we're taking a thousand points by quadrature or a thousand points by sampling from a uniform distribution over the same range, it doesn't matter. We're still computing the value at a thousand points and taking an average.

But now here's the trick.

Suppose we can find a helper distribution `q` that is large where `f(x,y)` has a lot of volume and very small where it has a little volume.

We can then use importance sampling to compute a more accurate estimate of the desired expected value, and therefore, the desired volume because most of the points we sample from `q` are in high-volume regions. Our thousand points from `q` will give us a better estimate!

Now, up the dimensionality further. Suppose we've got a function that takes three doubles and goes to double, and we wish to know its hypervolume over $(0, 0, 0)$ to $(1, 1, 1)$.

With quadrature, we're either doing a billion computations at a granularity of $0.001$ or, if we can only afford to do a thousand evaluations, that's the granularity of $0.1$.

**Every time we add a dimension, either the cost of our quadrature goes up by a factor of a thousand, or the cost stays the same, but the granularity is enormously coarsened, or we must abandon our simple quadrature algorithm in favor of a smarter one that concentrates on the high-area regions.**

Oh, but it gets worse.

When you are evaluating the hyper-volume of a 3-d surface embedded in 4 dimensions, there are a lot more points where the function can be zero! There is just so much *room* in high dimensions for stuff to be. The higher the dimensionality gets, the more important it is that you find the spikes and avoid the flats.

> **Exercise:** Consider an n-dimensional cube of side $1$. That thing always has a hyper-volume of $1$, no matter what $n$ is.

Now consider a concentric n-dimensional cube inside it where the sides are $0.9$ long.

For a 1-dimensional cube — a line — the inner line is $90\%$ of the length of the outer line, so we'll say that $10\%$ of the length of the outer line is "close to the surface". For a 2-dimensional cube — a square — the inner square has $81\%$ of the area of the outer square, so $19\%$ of the area of the outer square is "close to the surface". At what dimensionality is more than $50\%$ of the hyper-volume of the outer hypercube "close to the surface"?

> **Exercise:** Now consider an n-dimensional cube of side $1$ again, and the concentric n-dimensional sphere. That is a circle that exactly fits inside a square, a sphere that exactly fits inside a cube, and so on. The radius is $\frac{1}{2}$
> .

- The area of the circle is $\frac{\pi}{4} = 79\%$ of the area of the square.
- The volume of the sphere is $\frac{\pi}{6} = 52\%$ of the volume of the cube.
- ... and so on.

At what value for $n$ does the volume of the hypersphere become $1\%$ of the volume of the hypercube?

In high dimensions, any shape that is *anywhere* on the interior of a hypercube is tiny when compared to the massive hyper-volume near the cube's *surface*!

That means: if you're trying to determine the hyper-volume bounded by a function that has large values somewhere inside a hypercube, the samples must frequently hit that important region where the values are big. If you spend time "near the edges" where the values are small, you'll spend $> 90\%$ of your time sampling irrelevant values.

That's why importance sampling is so useful, and why we spend so much effort studying how to find distributions that compute expected values. **Importance sampling allows us to solve multidimensional integral calculus problems with reasonable compute resources numerically.**

> Now you know why we said earlier that we misled you when we said that the VEGAS algorithm was designed to find helpful distributions for importance sampling. The VEGAS algorithm does that, but that's not what it was designed to do; it was designed to solve multidimensional integral calculus problems. Finding good helper distributions is how it does its job.

> **Exercise:** Perhaps you can see how we would extend the algorithms we've implemented on distributions of doubles to distributions of tuples of doubles; we are not going to do that in this course; give it a shot and see how it goes!

---

There is so much more to say on this topic; people spend their careers studying this stuff. But we are going to wrap it up in the next lessons by giving some final thoughts, a summary of the work we've done, a list of some related topics not covered in this course, and a partial bibliography of the papers and other resources that we read when doing this course.