Multi-dimensional integration and Monte-Carlo integration

Why multi-dimensional integration is hard

Direct integration

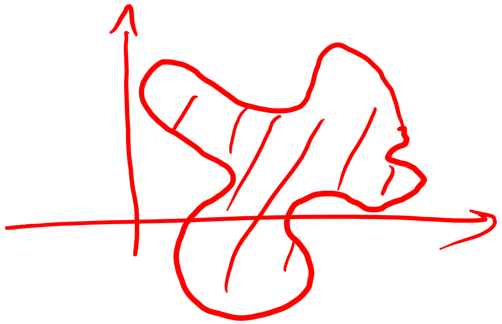
Monte-Carlo integration

Sampling: the big problem with Monte-Carlo integration

Solutions to this problem

- quasi-random numbers
- stratified sampling
- importance sampling

Why multi-dimensional integration is hard:



1. It can scale really badly:

   If for a 1D integral you need 100 points in a quadrature evaluation,
   For a 2D integral you need 100^2 points, for a 3D integral you need 100^3 points, and so on

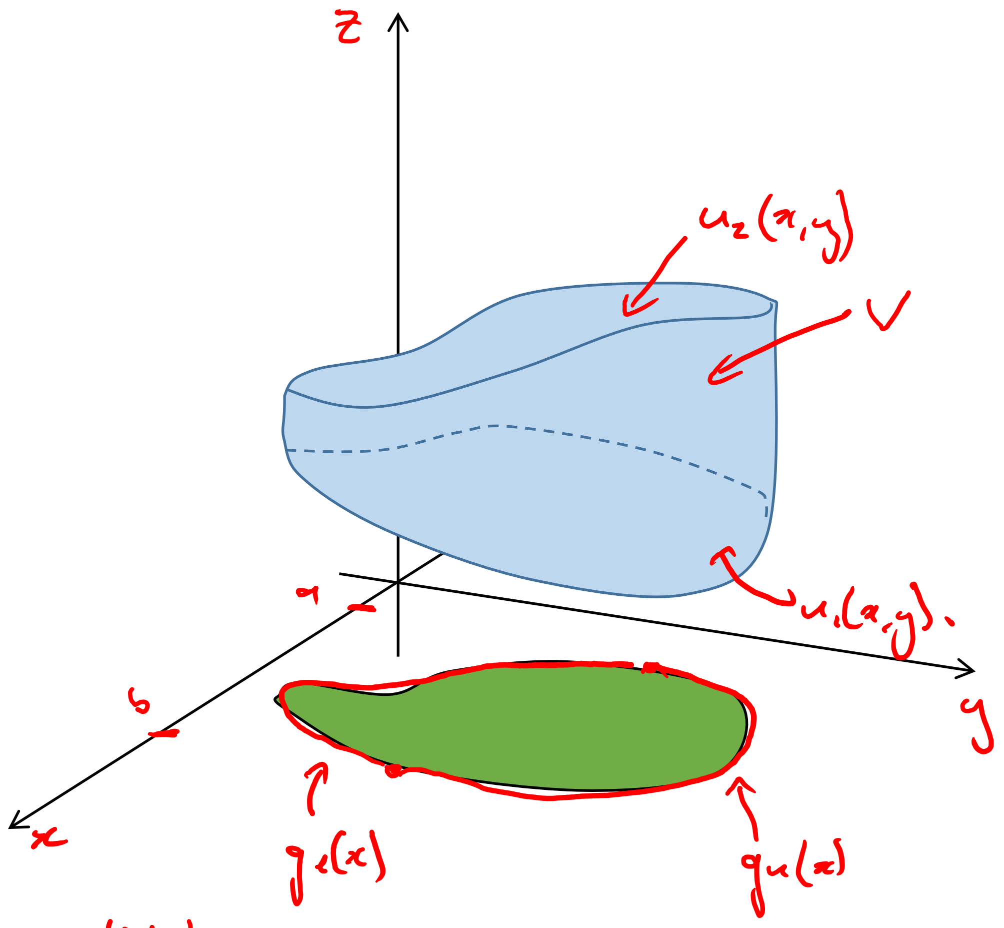
2. *Boundaries* become really complicated.

3. It is (often) difficult to know, in ND space, *where the maximum contributions to the integral come from*

Nested Integration

This is the only reliable option if you need to evaluate an N-D integral to numerical precision.

The idea is that you integrate over one dimension at a time, Using your 1D quadrature.



A simple domain in (say) 3D can be written

$$
\begin{aligned}
D = \{(x,y,z) | a \le x \le b \quad &, \\
g_\ell(x) \le y \le g_u(x) \quad &, \\
u_1(x,y) \le z \le u_2(x,y) \quad &\}
\end{aligned}
$$

curve

The integral in 3D over this domain is then

first

third

$$\iiint_V f(x,y,z)\mathrm{d}x\mathrm{d}y\mathrm{d}z = \int_a^b \left[\int_{g_\ell(x)}^{g_u(x)} \left[\int_{u_1(x,y)}^{u_2(x,y)} f(x,y,z)\mathrm{d}z\right]\mathrm{d}y\right]\mathrm{d}x$$

function of x

$f(x)$

function of (x,y)

second

Code structure for nested integration can be a bit subtle, because you need to return a function as the answer to the inner integrals.

Nested integration over x,y

Function yint(f,x,c,d)

    For the given value of x, integrate f(x,y) over y between c and d using 1D quadrature, call this yint

    return yint

Define limits for x as [a,b] ← beginning
Define limits for y as [c,d]

Result = integration of yint(f,x,c,d) over x between a and b using 1D quadrature

Note:

For most programming languages,
You'll have to define "dummy" functions

f1(y) = f(x,y) with a fixed x

single variable

to do this step, and another "dummy function"

f2(x) = yint(f,x,c,d) with a fixed f, c and d

To do this step.

Nested quadrature is:

- Reliable ✓
- Accurate ✓
- Complicated ✗
- Slow ✗

Is there a method that scales better for higher dimensions, even if it means sacrificing some accuracy?

Monte Carlo Integration

To find the integral

$$\int_a^b f(x)\mathrm{d}x$$

We previously *approximated the function* with something that is easier to integrate.

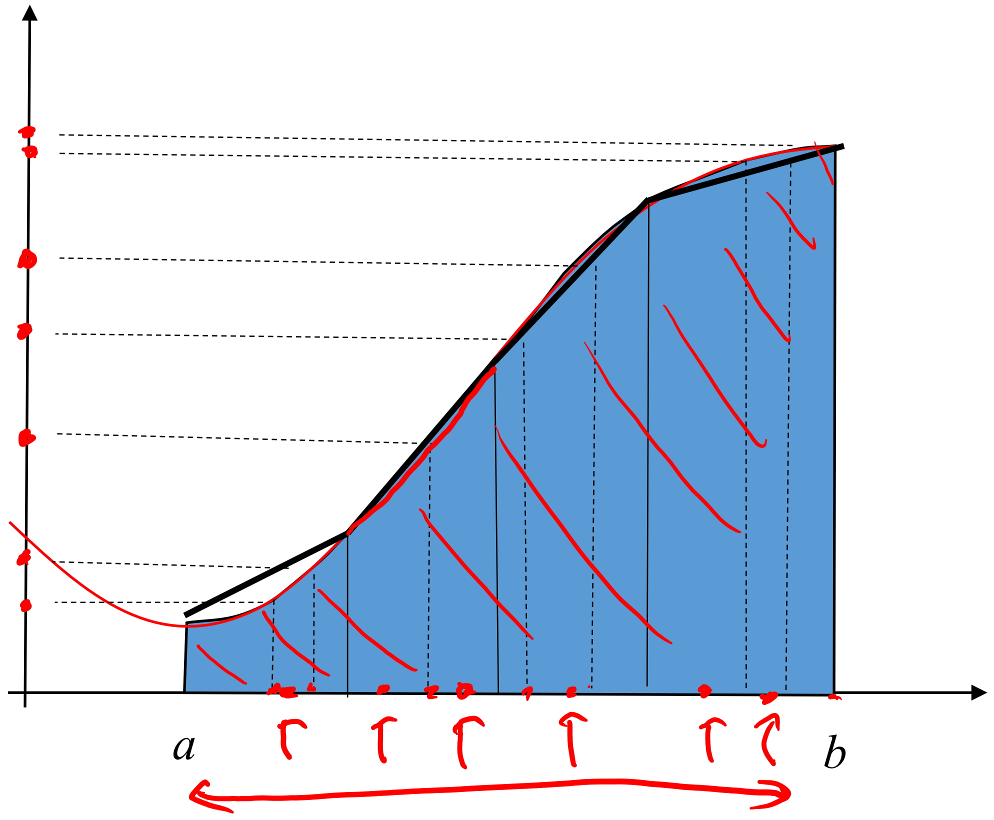
However we could also view the integral as the *expected value of f if we sampled it over a large number of points.*

$$E(f) = \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

As the number of sampling points increases,

$$E(f) \to \frac{1}{b-a}\int_a^b f(x)\mathrm{d}x$$

In accordance with *the Law of Large numbers.*



$$\frac{1}{N}\sum_{i=1}^{N} f(x_i) = \frac{1}{b-a}\int_a^b f(x)\,dx \quad \text{as } N \to \infty.$$

For large N we can approximate

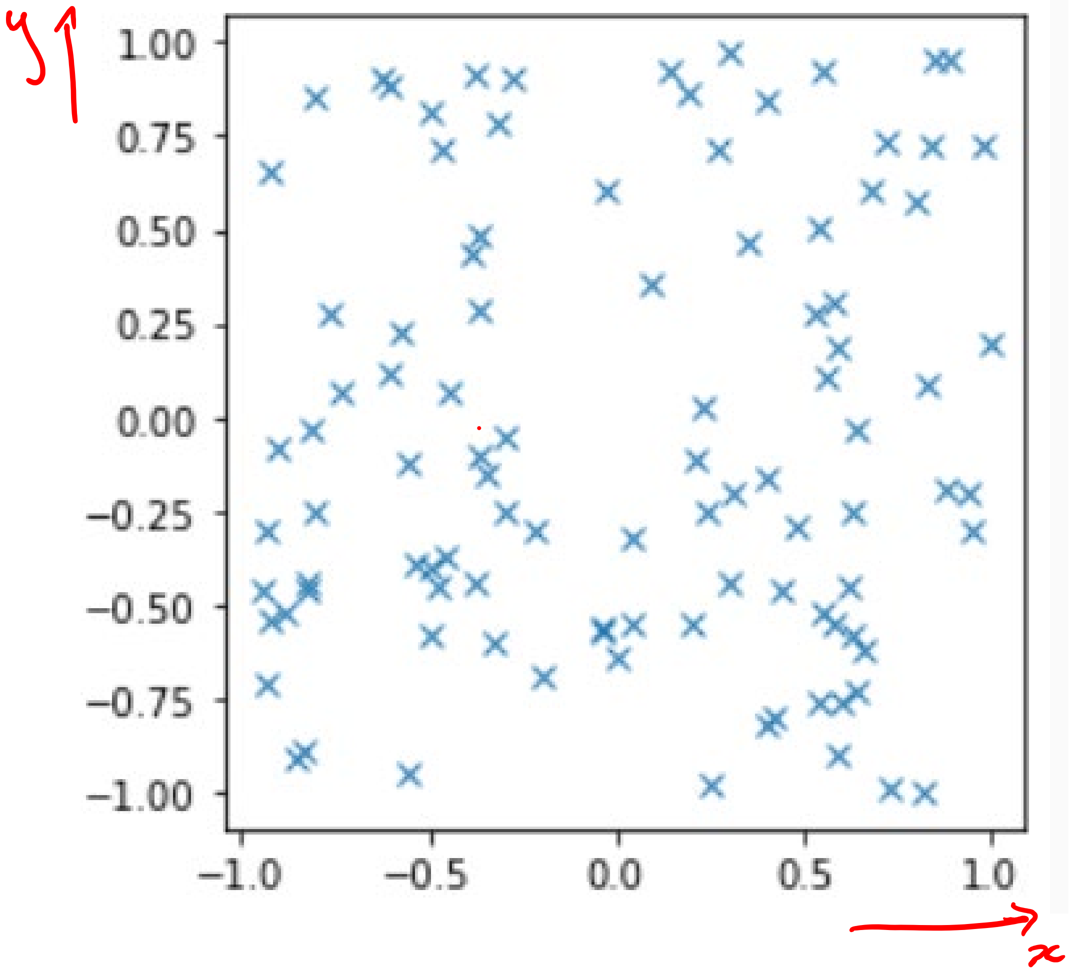
$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{N}\sum_{i=1}^{N} f(x_i)$$

This also works in two dimensions:

"Volume" of the domain

$$\int_a^b \int_c^d f(x, y)\mathrm{d}x\mathrm{d}y \approx \underline{\frac{V}{N} \sum_{i=1}^{N} f(x_i, y_i)}$$

Where V is the *size of the region being sampled.*

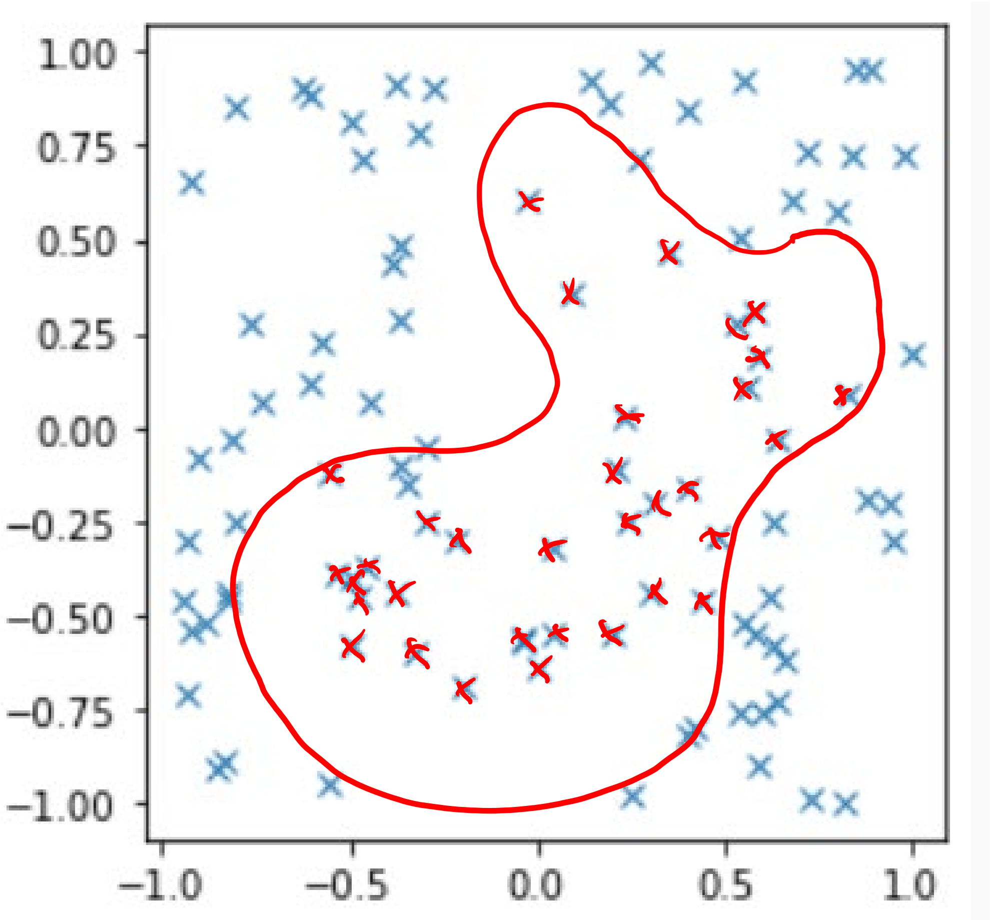


This also works in two dimensions:

$$\boxed{\int_a^b \int_c^d f(x,y)\mathrm{d}x\mathrm{d}y \approx \frac{V}{N}\sum_{i=1}^{N} f(x_i, y_i)}$$

Where V is the *size of the region being sampled.*

This also works for complicated regions:
Just define the function to be zero *outside the region that you're interested in.*



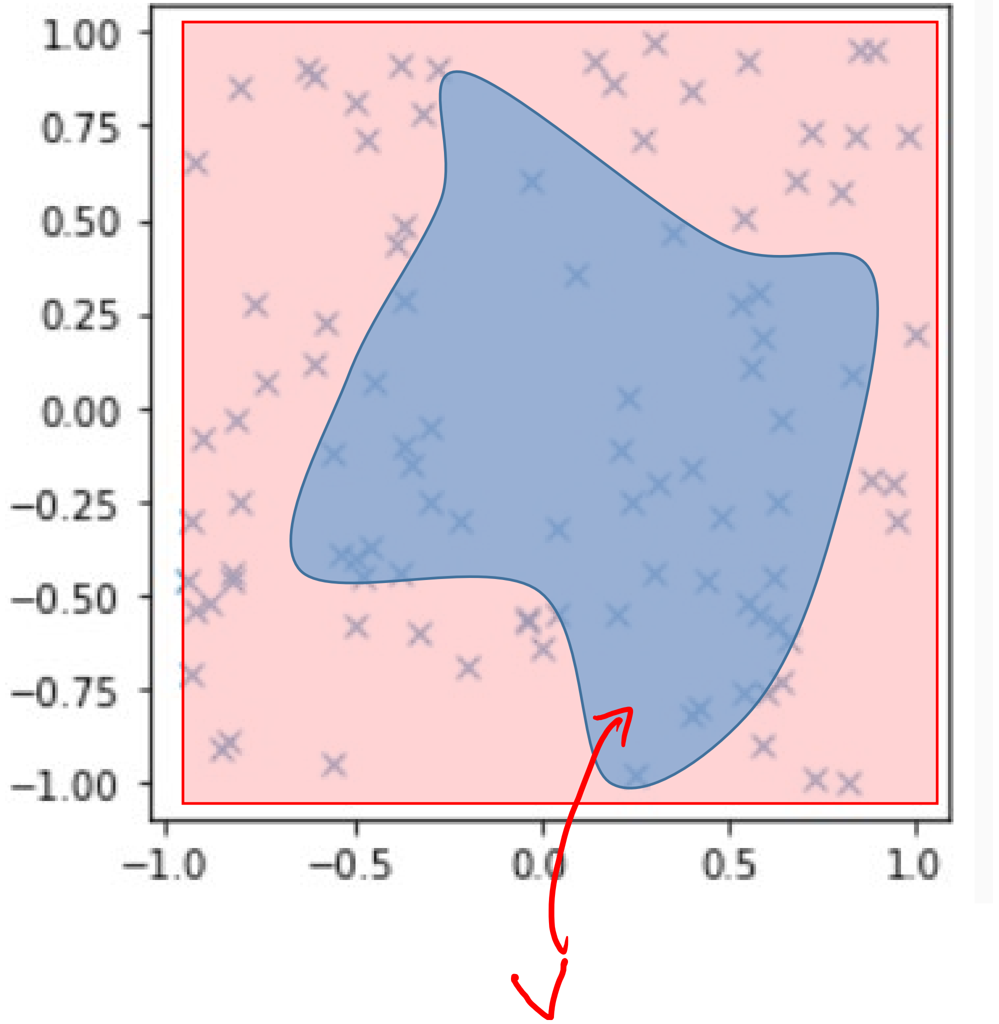
Monte Carlo integration

1. Pick a random sample of points for each of your dimensions p = (xp,yp,zp, …), which *covers your region of integration.*

2. The integral is then the sum

$$I \approx \frac{V}{N} \sum_{i=1}^{N} f(x_i, y_i)$$

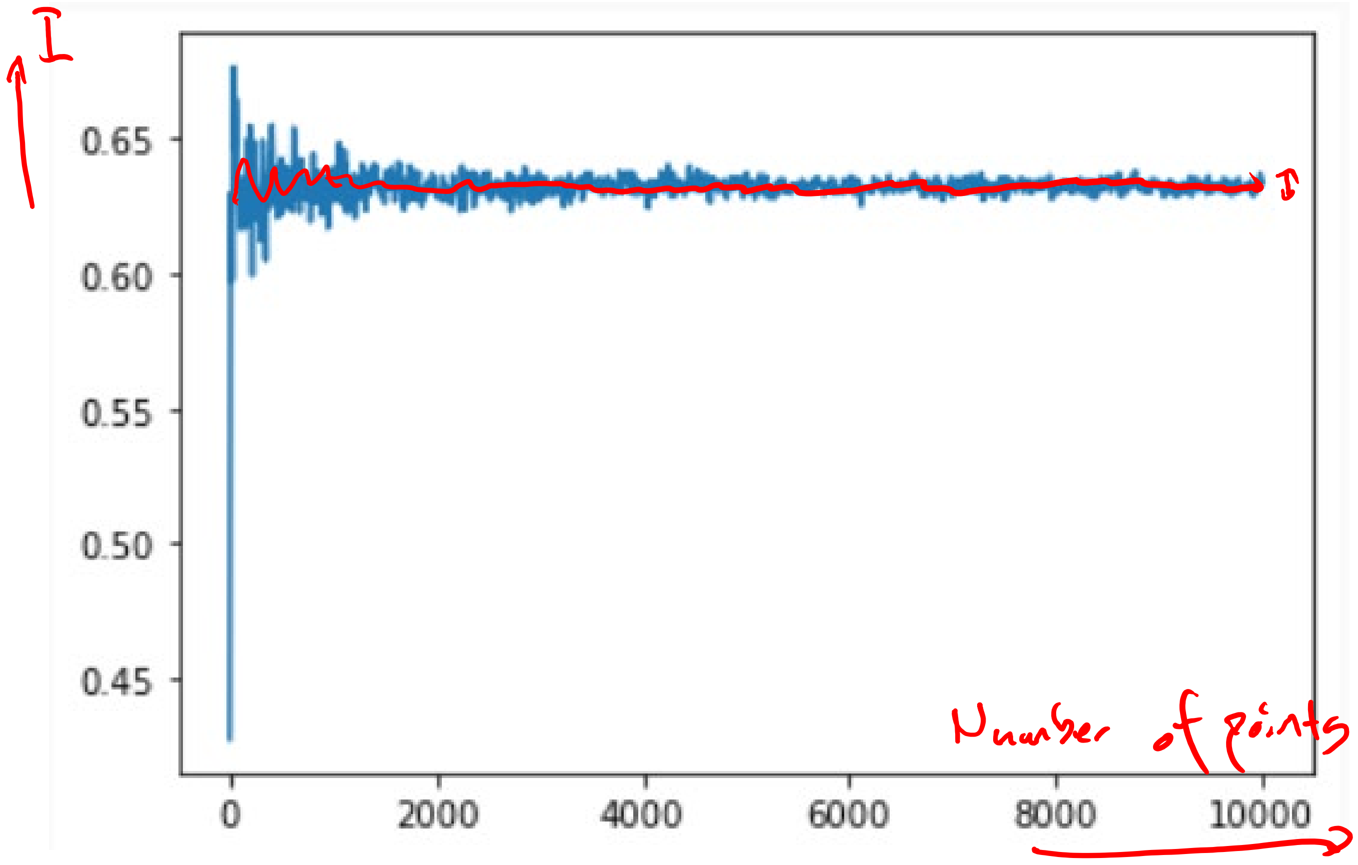
Where you only include the points lying in the integration region. Here V is the volume of the sampling domain.



Advantages:

1. Scales with the number of points rather than as the power of the number of dimensions !!

2. Simple to implement



Disadvantages:

Sample variance!

Variance of Monte Carlo:

Let the approximate integral be

$$I_N = \frac{V}{N}\sum_{i=1}^{N} f(x_i)$$

$$\mathrm{Var}[I_N] = \mathrm{Var}\left[\frac{V}{N}\sum_{i=1}^{N} f(x_i)\right]$$

$$= \frac{V^2}{N^2}\sum_{i=1}^{N}\mathrm{Var}[f(x_i)]$$

$$= \frac{V^2}{N^2}\sum_{i=1}^{N}\sigma_N^2 = \frac{V^2}{N^2}\sigma_N^2 N$$

$$= \frac{V^2}{N}\sigma_N^2 \quad \sim \frac{1}{N}$$

Recall:

$$\mathrm{Var}[f(x)] = \sigma_N^2 = \frac{1}{N-1}\sum_{i=1}^{N}[f(x_i) - E(f)]^2$$

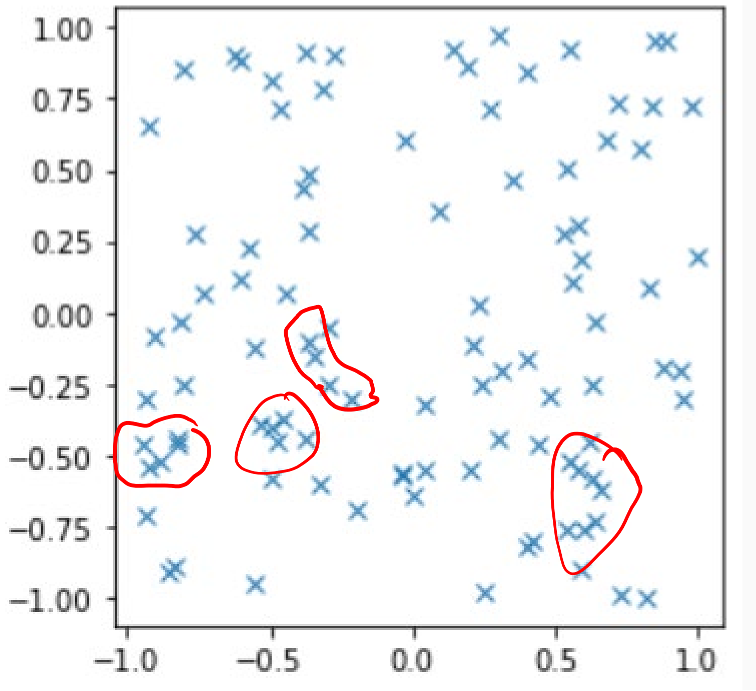
So that:

$$\mathrm{Var}[af(x)] = a^2\mathrm{Var}[f]$$

The error in the estimate of the integral decreases as $\sim 1/\sqrt{N}$

Strategies for reducing the variance (and hence the error) revolve around *reducing the "clumping"* of the random points.

Random numbers clump together in a way not ideally suited For Monte-Carlo integration.



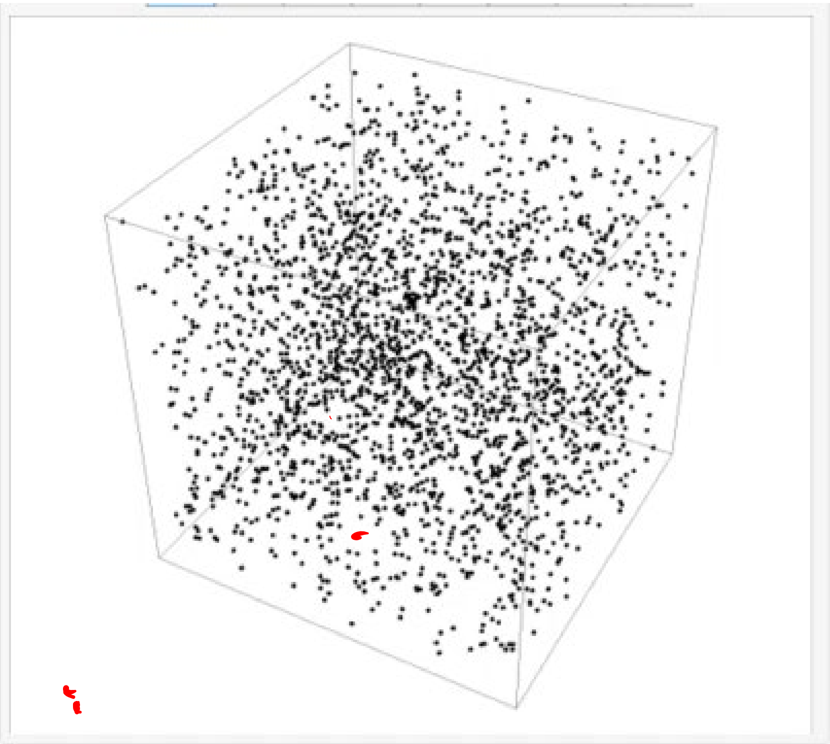
Is there a type of random number that is Random, but not too random?

The answer is "yes": these are known as *quasi-random numbers.*

Quasi-random numbers vs Pseudo-random numbers

Quasi-random numbers, which aim at a certain level of randomness, should Not be confused with Pseudo-random numbers, which try to be as random as possible.

Sequences of genuine random numbers are (surprisingly!) difficult to simulate using computers. Often a sequence that seems random Ends up having hidden order in it.



The attempt to generate pseudo-randomness is a whole subject in itself! See Chapter 7 of Numerical Recipes for more info.

Quasi-random number sequences

These sequences are also known as "sub-random sequences"; they
Are random numbers that "avoid each other" to a defined extent.

Important types:

Halton sequences:
(easy to code)

To get the jth number in a Halton sequence:

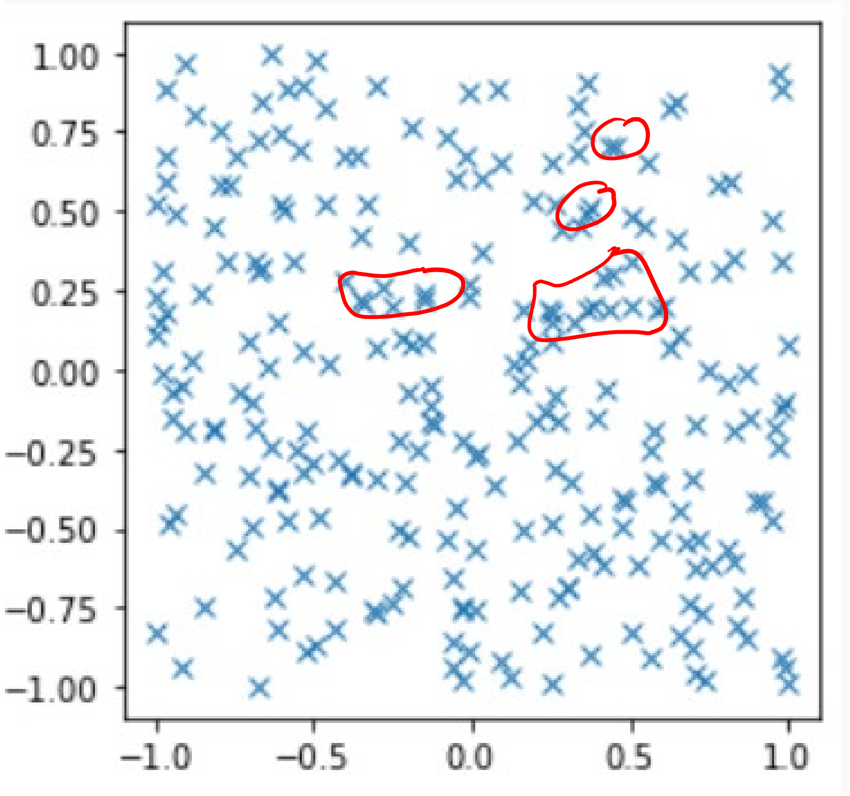
1. Write j as a number in base b, where b is prime

2. Reverse the digits, and put a decimal point in front of the sequence. This is the jth number H_j

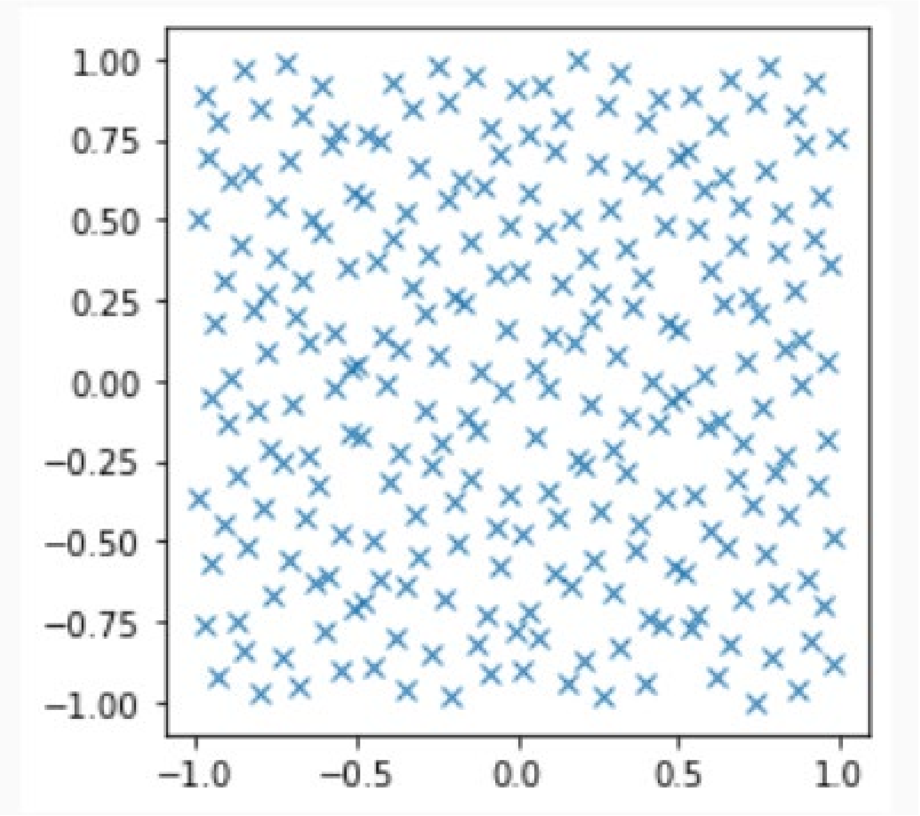
Sobol sequences:
(harder to code, but generally better)

To get the jth number of the Sobol sequence,
We need to compute the *Gray Code* of j G(j), then write it as a binary fraction. It then gets pretty complicated, but let me know If you're interested!

https://en.wikipedia.org/wiki/Sobol_sequence
https://en.wikipedia.org/wiki/Gray_code



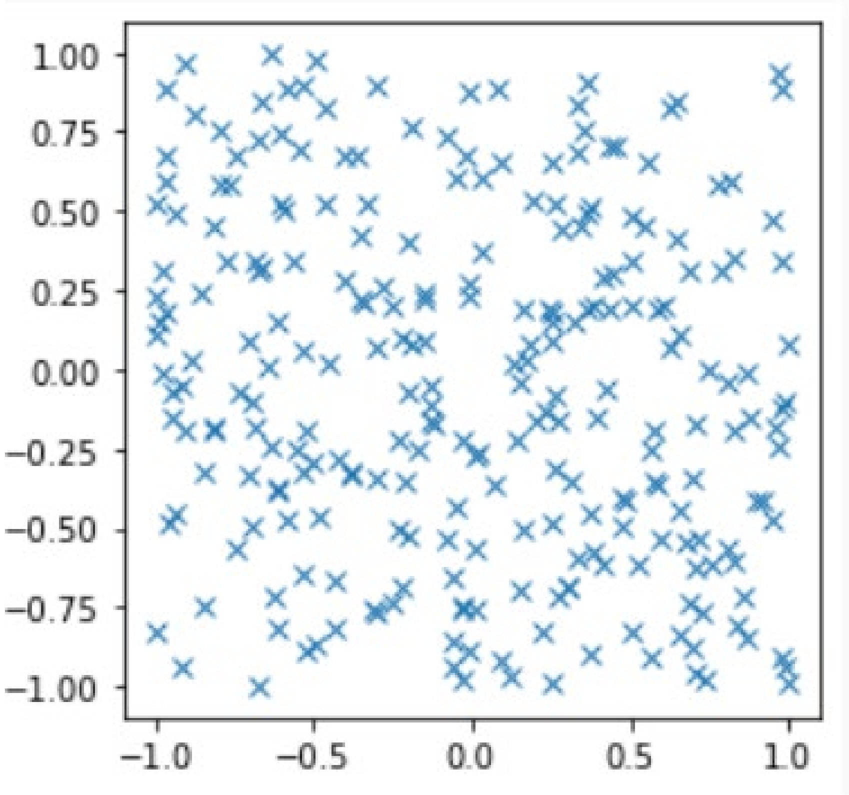
pseudo-random

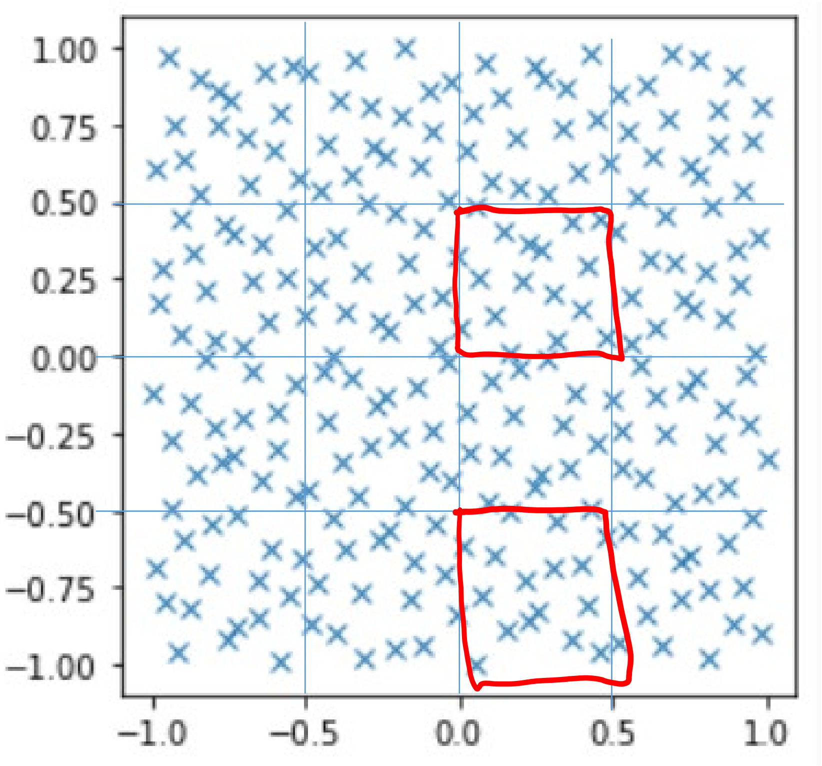


Sobol sequence

Stratification sampling:

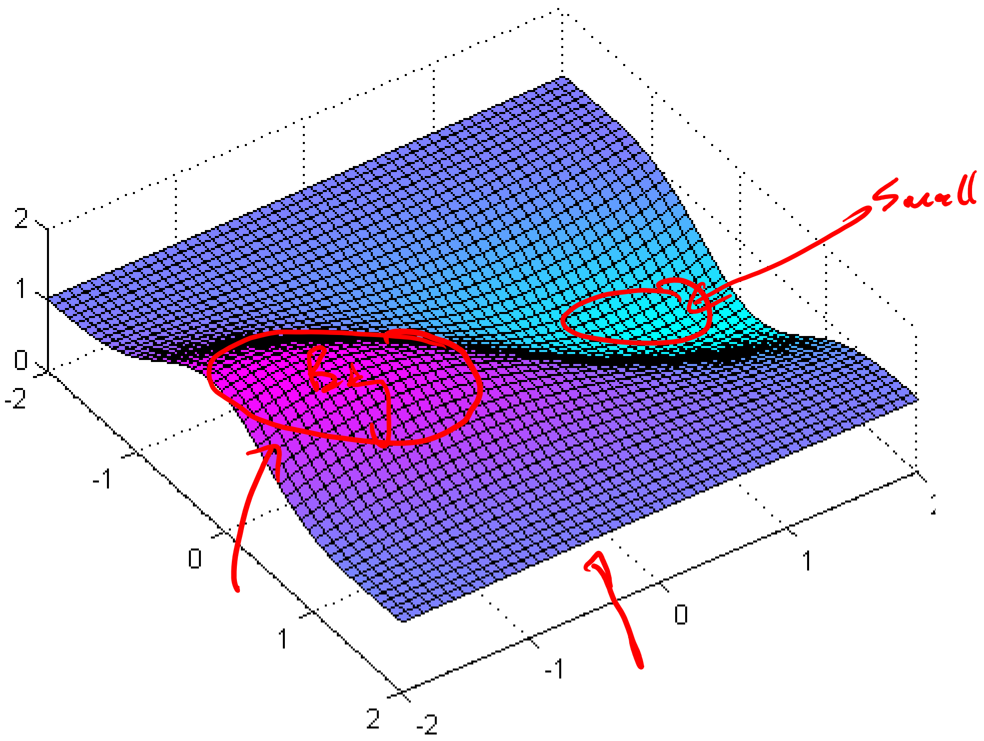
This strategy reduces the sampling variance by (recursively)
Dividing the volume into sub-domains, and sampling only on those domains.





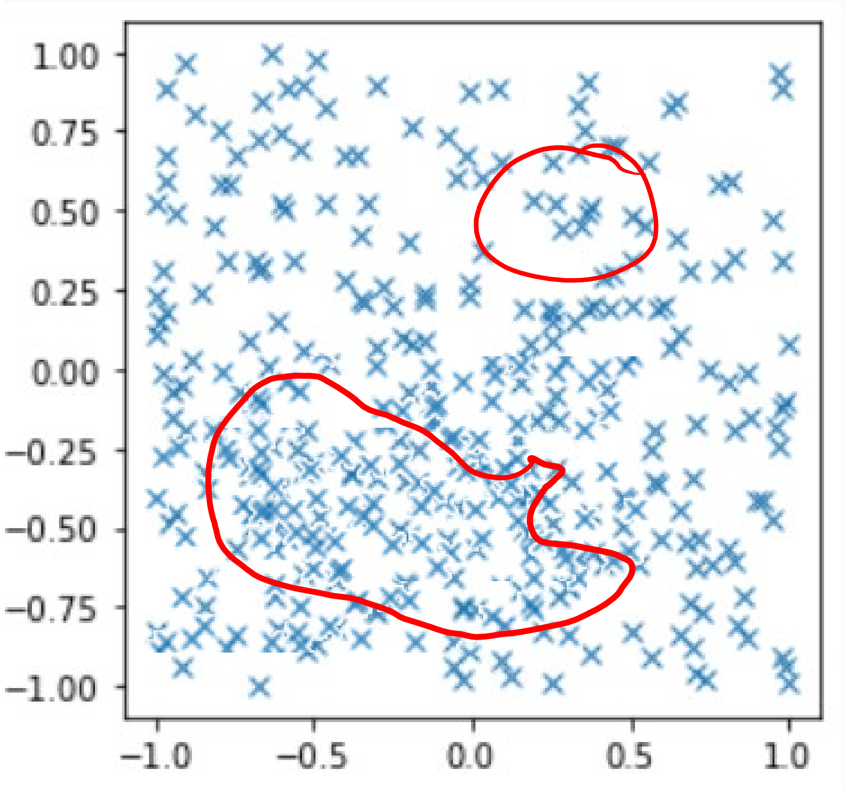
Importance sampling

Put more samples where the function f is bigger:



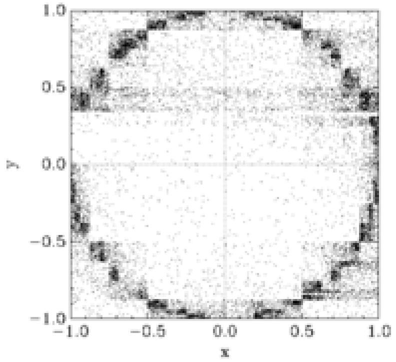
Choose a sampling density p such that

$$p \sim |f|$$



Other strategies:

- Recursive stratified sampling



- Mixed methods (using both stratified sampling and importance sampling)