A detailed reference of MCMC algorithms

Siddharth Bhat (20161105) siddu.druid@gmail.com

April 21, 2020

1 Why do we need MCMC? A practitioner's perspective

Consider that we are plonked down in the `C` programming language, and our only method to generate random numbers is to call `int rand(void)`. However, *the type is a lie*, since we are able to modify global mutable state. So, really, to have a mathematical discussion about the whole state of affairs, we will write the type of `rand` as $\texttt{rand} : S \rightarrow S \times \texttt{int}$ — that is, it receives the entire state of the program, and then returns the `int`, along with the new state of the program. $\texttt{uniformbool} : \texttt{S} \rightarrow \texttt{S} \times \{0, 1\}$.

When we say that `rand` generates random numbers, we need to be a little more specific: what does it mean to generate random numbers? we need to describe the *distribution* according to which we are receiving the random numbers from the random number generator *rand*. What does that mean? Well, it means that as we generate more numbers, the *empirical distribution* of the list of numbers we get from the successive calls to `uniformbool` tends to some *true distribution*. We will call this *true distribution* succincty as **the** distribution of the random number generator. Formally, let us define $F(t) \equiv \int_0^t P(x)dx$ to be the cumulative distribution of P.

In the case of `uniformbool`, we are receiving random numbers according to the distribution:

$$P_{\texttt{uniformbool}} : \{0, 1\} \rightarrow [0, 1]; \qquad P_{\texttt{uniformbool}}(x) = 1/2$$

That is, both 0 and 1 are *equally likely*. However, this is extremely boring. What we are *usually* interested in is to sample $\{0, 1\}$ in some *biased* fashion:

$$P^{bias}_{\texttt{uniformbool}} : \{0, 1\} \rightarrow [0, 1]; \qquad P^{bias}_{\texttt{uniformbool}}(0) = bias; \qquad P^{bias}_{\texttt{uniformbool}}(1) = 1 - bias$$

And far more generally, we want to sample from *arbitrary domains* with *arbitrary distributions*:

$$\texttt{sampler}^P_X : S \rightarrow S \times X;$$

This function is boring. What we *really* want to sample from are more interesting distributions. For example:

- The normal distribution $P(x : \mathbb{R}) = e^{-x^2}$.
- The poisson distribution $P(x : \mathbb{N}) = e^{-\lambda}\lambda^n/n!$.
- A custom distribution $P(x : \mathbb{R}) = |sin(x)|$.

1.1 Fundamental Problem of MCMC sampling

Given a weak, simple sampler of the form $\texttt{rand} : S \rightarrow S \times \texttt{int}$, build a sampler $\texttt{sampler}(P, X) : T \rightarrow T \times X$ which returns value distributed according to some *un-normalized* distribution of choice $P : X \rightarrow \mathbb{R}$.

The un-normalized constraint is important: it is what allows us to exploit MCMC to a wide variety of scienarios.

1.2 Sampling use case 1. Simulation And Sampling

Often, we do really want samples from a particular distribution. For example, we might often want to apply Bayes' rule and sample from the posterior distribution. Recall the formla:

$$P(Y|X) = P(X|Y).P(Y)/P(X)$$

1.3 Sampling use case 2. Gradient Free Optimisation

We want to maximize a function $f : X \to \mathbb{R}$. However, we lack gradients for f, hence we cannot use techniques such as gradient descent, or other techniques from convex optimisation.

In such a case, we can consider f as some sort of unnormalized probability distribution, and use MCMC to sample from f.

1.4 Sampling use case 3. Numerical Integration

2 Where it all begins: The Metropolis Hastings sampler

2.1 The big idea

We wish to sample from a distribution $\mathfrak{P}$, but we do not know how to do so. The idea is that we build a markov chain $M[\mathfrak{P}, \texttt{Prop}]$ where $\mathfrak{P}, \texttt{Prop}$ are supplied by the user. We will show that the *stationary distribution* of $M[\mathfrak{P}, \texttt{Prop}]$ is going to be $\mathfrak{P}$. This will ensure that if we interpret *states of* M as samples, these samples will the distriubted according to $\mathfrak{P}$.

2.2 Detail balance: A tool for proofs

We will first require a condition that will enable to rapidly establish that some distribution $\mathfrak{P} : X \to \mathbb{R}$ is the stationary distribution of a markov chain M. If the transition kernel $K : X \times X \to \mathbb{R}$ of $M \equiv (X, K)$ is such that:

$$\forall x, x' \in X, \mathfrak{P}(x)K(x, x') = \mathfrak{P}(x')K(x', x)$$

Then $\mathfrak{P}$ is said to be **detail balanced** with respect to M.

Theorem 1 *If $\mathfrak{P}$ is detail balanced to $M \equiv (X, K : X \to (X \to [0, 1])$, then $\mathfrak{P}$ is the stationary distribution of M.*

Proof 1.1 *Let $\mathfrak{P}$ be detail balanced to M. This means that:*

$$\forall x, x' \in X, \mathfrak{P}(x)K(x)(x') = \mathfrak{P}(x')K(x')(x)$$

Let us say say that we are in state $\mathfrak{P}$. We wish the find the probability distribution after one step of transition. The probability of being in some state x'_0 is going to be:

$$Pnext(x'_0) \equiv \sum_x \mathfrak{P}(x) K(x)(x'_0)$$

since we have $\mathfrak{P}(x)$ probability to be at a given x, and $K(x, x'_0)$ probability to go from x to x'_0. If we add over all possible $x \in X$, we get the probability of all states to enter in x'_0. Manipulating $Pnext$, we get:

$$\begin{aligned}
&Pnext(x'_0) \equiv \sum_x \mathfrak{P}(x) K(x)(x'_0) \\
&= \sum_x \mathfrak{P}(x'_0) K(x'_0)(x) \quad \textit{(by detail balance)} \\
&= \mathfrak{P}(x'_0) \sum_x K(x'_0)(x) \quad \textit{(}P(x_0)\textit{ is constant} \\
&= \mathfrak{P}(x'_0) \cdot 1 \quad \textit{(}K(x'_0)\textit{ is a distribution which is being summed over)} \\
&= \mathfrak{P}(x'_0) \quad \textit{(eliminate multiplication with 1)}
\end{aligned}$$

Hence, $Pnext(n'_0) = \mathfrak{P}(x'_0)$ if $\mathfrak{P}$ is the current state, and $\mathfrak{P}$ is in detail balance with the kernel K. This means that $\mathfrak{P}$ is the stationary distribution of M. $\triangle$

2.3 Metropolis Hastings

There are three key players in the metropolis hasting sampler:

1 $\mathfrak{P} : X \to \mathbb{R}$: the probability distribution we wish to sample from.

2 $\texttt{Prop} : X \to (X \to \mathbb{R})$. For each $x_0 \in X$, provide a distribution $\texttt{Prop}(x) : X \to \mathbb{R}$ that is used to sample points around x_0. `Prop` for *proposal*.

3 $M[\mathfrak{P}, \texttt{Prop}] \equiv (X, K[\mathfrak{P}, \texttt{Prop}] : X \to \mathbb{R})$: The Metropolis Markov chain we will sample from, whose stationary distribution is $\mathfrak{P}$ — M for *Markov*.

We want the stationary distribution of $M[\mathfrak{P}, \texttt{Prop}]$ to be $\mathfrak{P}$. We also wish for $K[\mathfrak{P}, \texttt{Prop}](x_0) \sim \texttt{Prop}(x_0)$: That is, at a point x_0, we want to choose new points in a way that is 'controlled' by the proposal distribution $\texttt{Prop}(x_0)$, since this will allow us to 'guide' the markov chain towards regions where $\mathfrak{P}$ is high. If we had a gradient, then we could use $\mathfrak{P}'$ to 'move' from the current point x_0 to a new point. Since we lack a gradient, we will provide a custom $\texttt{Prop}(x_0)$ for each x_0 that will tell us how to pick a new x', in a way that will improve $\mathfrak{P}$. So, we tentatively define

$$K[\mathfrak{P}, \texttt{Prop}](x)(x') \overset{?}{\equiv} \texttt{Prop}(x)(x').$$

Recall that for $\mathfrak{P}$ to be a stationary distribution of K, it is sufficient for $\mathfrak{P}$ to be in detail balance for K. So, we write:

$$\mathfrak{P}(x)K(x,x') \overset{?}{=} \mathfrak{P}(x')K(x',x) \quad \text{(going forward equally likely as coming back)}$$
$$\mathfrak{P}(x)Prop(x)(x') \overset{?}{=} \mathfrak{P}(x')Prop(x')(x) \quad \text{(this is a hard condition to satisfy)}$$

This is far too complicated a condition to impose on $Prop$ and $\mathfrak{P}$, and there is no reason for this condition to be satisfied in general. Hence, we add a custom "fudge factor" $\alpha \in X \to (X \to \mathbb{R})$ that tells us how often to transition from x to x'. We redefine the kernel as:

$$K[\mathfrak{P}, \texttt{Prop}](x)(x') \equiv \texttt{Prop}(x)(x')\alpha(x)(x')$$

Redoing detail balance with this new K, we get:

$$\mathfrak{P}(x)K(x,x') \overset{?}{=} \mathfrak{P}(x')K(x',x) \quad \text{(going forward equally likely as coming back)}$$
$$\mathfrak{P}(x)Prop(x)(x')\alpha(x)(x') = \mathfrak{P}(x')Prop(x')(x)\alpha(x')(x) \quad \text{(Fudge a hard condition with } \alpha\text{)}$$
$$\frac{\alpha(x)(x')}{\alpha(x')(x)} = \frac{\mathfrak{P}(x')Prop(x)(x')}{\mathfrak{P}(x)Prop(x')(x)} \quad \text{(Find conditions for } \alpha\text{)}$$

What we have above is a *constraint* for α. We now need to *pick* an α that satisfies this. A reasonable choice is:

$$\alpha(x)(x') \equiv \min\left(1, \frac{\mathfrak{P}(x')\texttt{Prop}(x')(x)}{\mathfrak{P}(x)\texttt{Prop}(x)(x')}\right)$$

.

since K is a transition kernel, we canot have its entries be greater tha n 1. Hence, we choose to clamp it with a $\min(1, \cdot)$. This finally gives us the kernel as:

$$K[\mathfrak{P}, \texttt{Prop}](x)(x') \equiv \texttt{Prop}(x)(x')\alpha(x)(x')$$
$$K[\mathfrak{P}, \texttt{Prop}](x)(x') = \texttt{Prop}(x)(x')\min\left(1, \frac{\mathfrak{P}(x')\texttt{Prop}(x')(x)}{\mathfrak{P}(x)\texttt{Prop}(x)(x')}\right)$$
$$K[\mathfrak{P}, \texttt{Prop}](x)(x') = \min\left(\texttt{Prop}(x)(x'), \frac{\mathfrak{P}(x')\texttt{Prop}(x')(x)}{\mathfrak{P}(x)}\right)$$

We can make sure that detail balance is satisfied:

$$\mathfrak{P}(x)K[\mathfrak{P}, \texttt{Prop}](x)(x') = \mathfrak{P}(x)\min\left(\texttt{Prop}(x)(x'), \frac{\mathfrak{P}(x')\texttt{Prop}(x')(x)}{\mathfrak{P}(x)}\right)$$
$$= \min\left(\mathfrak{P}(x)\texttt{Prop}(x)(x'), \mathfrak{P}(x)\frac{\mathfrak{P}(x')\texttt{Prop}(x')(x)}{\mathfrak{P}(x)}\right)$$
$$= \min\left(\mathfrak{P}(x)\texttt{Prop}(x)(x'), \mathfrak{P}(x')\texttt{Prop}(x')(x)\right)$$

Note that the above right-hand-side is symmetric in x and x', and hence we can state that:

$$\mathfrak{P}(x)K[\mathfrak{P},\texttt{Prop}](x)(x') = \min\left(\mathfrak{P}(x)\texttt{Prop}(x)(x'), \mathfrak{P}(x')\texttt{Prop}(x')(x)\right) = \mathfrak{P}(x')K[\mathfrak{P},\texttt{Prop}](x')(x)$$

Hence, we can wrap up, stating that our design of K does indeed give us a markov chain whose stationary distribution is $\mathfrak{P}$, since $\mathfrak{P}$ is detail balanced with $K[\mathfrak{P},\texttt{Prop}]$. As an upshot, we also gained a level of control with `Prop`, where we are able to provide "good" samples for a given point.

2.4 Simplification when proposal is symmetric

If our function `Prop` is symmetric: $\forall x, x' \texttt{Prop}(x)(x') = \texttt{Prop}(x')(x)$, then a lot of the above derivation becomes much simpler. We will perform those simplifications here for pedagogy.

When we have $\texttt{Prop}(x)(x') = \texttt{Prop}(x')(x)$, we can simply α:

$$\alpha(x)(x') \equiv \min\left(1, \frac{\mathfrak{P}(x')\texttt{Prop}(x')(x)}{\mathfrak{P}(x)\texttt{Prop}(x)(x')}\right)$$
$$\alpha(x)(x') = \min\left(1, \frac{\mathfrak{P}(x')}{\mathfrak{P}(x)}\right) \quad \text{[cancelling: } \texttt{Prop}(x)(x') = \texttt{Prop}(x')(x)\text{]}$$

This also makes the kernel look a lot more pleasing:

$$K[\mathfrak{P},\texttt{Prop}](x)(x') \equiv \texttt{Prop}(x)(x')\alpha(x)(x')$$
$$K[\mathfrak{P},\texttt{Prop}](x)(x') = \texttt{Prop}(x)(x')\min\left(1, \frac{\mathfrak{P}(x')}{\mathfrak{P}(x)}\right)$$

2.4.1 code

```
# prob is the distribution to sample from;
# symproposal is the *symmetric* proposal function
# symproposal: X -> X; produces a new 'X' from a
#              given 'X' with some distribution.
# prob: X -> |R: gives probability of point 'xi'.
# N: number of markov chain walks before returning a new sample.
def metropolis_hastings(prob, symproposal, x0, N):
  x = x0
  while True:
    for i in range(N):
      # xnext chosen with Prop(x)(x') prob.
      xnext = symproposal(x); px = prob(x); pxnext = prob(xnext);
      # x' chosen with Prop(x)(x') * alpha prob.
      r = uniform01(); alpha = min(1, pxnext / px); if r < alpha: x = xnext
    yield x
```

3 Gibbs sampling

We wish to sample from a joint distribution $P(X_1, X_2, X_3)$. However, it might be far cheaper to sample $P(X_1|X_2, X_3)$, $P(X_2|X_1, X_3)$, and $P(X_3|X_1, X_2)$. If it is indeed cheaper, then we can use a Gibbs sampler to draw from the actual distribution $P(X_1, X_2, X_3)$ by combining samples from the *conditional* distribution cleverly. The code is:

3.1 Gibbs sampling maintains detail balance

Consider a two-variate state where we sample the first variable from its conditional distribution. A move between (x_1, x_2) and (y_1, y_2) has zero probability in both directions if $x_2 \neq y_2$, and thus detail balance automatically holds. If $x_2 = y_2$, then we can consider:

$$
\begin{aligned}
&\pi(x_1, x_2) Prob((x_1, x_2) \rightarrow (y_1, x_2)) \\
&= \pi(x_1, x_2) p(y_1 | X_2 = x_2) \\
&= \pi(x_1, x_2) \frac{\pi(y_1, x_2)}{\sum_z \pi(z, x_2)} \\
&= \pi(y_1, x_2) \frac{\pi(x_1, x_2)}{\sum_z \pi(z, x_2)} \quad \text{[move } (y_1, x_2) \text{ from the fraction outside]} \\
&= \pi(y_1, x_2) p(x_1 | X_2 = x_2) \\
&= \pi(y_1, x_2) Prob((y_1, x_2) \rightarrow (x_1, x_2))
\end{aligned}
$$

3.2 code

```
# sampler_x: y, z -> new x
# sampler_y: x, z -> new y
# sampler_z: x, y -> new z
# N: number of iterations.
# returns: a generator of the new (x, y, z)
def sampler_xyz(sampler_x, sampler_y, sampler_z, x0, y0, z0, N):
    (x, y, z) = (x0, y0, z0)
    while True:
      for i in range(N):
        x = sampler_x(y, z)
        y = sampler_y(x, z) # NOTE: use *new* x
        z = sampler_z(x, y) # NOTE: use *new* x, y
      yield (x, y, z)
```

4 Hamiltonian Monte Carlo

If our target probability density $\mathfrak{P} : X \to [0, 1]$ is *differentiable*, then we can use the derivative of $\mathfrak{P}(x)$ to provide better proposals. The idea is as follows:

- Interpret the probability landscape as a potential, with points of high probability being "valleys" and points of low probability being "peaks" (ie, invert the probability density with a transform such as $U[\mathfrak{P}](x) = e^{-\mathfrak{P}(x)}$. This way, a ball rolling on this terrain will try to move towards the valleys — which are the locations of high probability $\mathfrak{P}$.
- For a proposal at a position $x_0 \in X$, keep a ball at x_0, *randomly choose its velocity*, simulate the ball according to classical mechanics (Newton's laws of motion) for a fixed duration $D \in \mathbb{R}$ and propose the final position as the final position of the ball. This is reversible and detail balanced because *classical mechanics is reversible and detail balanced.*

We need the derivative of $\mathfrak{P}(x)$ to be able to simulate the ball according to Newton's laws. If we can do this, though, we are able to cover large amounts of terrain.

4.1 Newton's laws of motion: Hamilton's equations

$$\frac{\partial \mathbf{q}}{\partial t} = \frac{\partial H}{\partial \mathbf{p}} \qquad \frac{\partial \mathbf{p}}{\partial t} = -\frac{\partial H}{\partial \mathbf{q}}$$

In our case, the choice of the hamiltonian will consider the negative-log-probability-density to be the *energy*. So, higher probability has lower energy. We know that particles like to go towards lower energy states.

$$H(\mathbf{q}, \mathbf{p}) \equiv U(\mathbf{q}) + K(\mathbf{p}) \quad K(\mathbf{p}) \equiv \mathbf{p}^T M^{-1} \mathbf{p}/2$$

where M is a symmetric positive-definite matrix known as a "mass matrix".

4.2 Modifying Hamilton's equations for generating proposals

We wish to maximize $\mathfrak{P}(x)$. To do this, we will consider a new energy function $U[\mathfrak{P}](x) \equiv e^{-\mathfrak{P}(x)}$. When $U[\mathfrak{P}] = e^{-U[\mathfrak{P}](x)}$ is minimized, then $\mathfrak{P}(x)$ will be maximized.

We use the proposal function as:

- We are currently at the location $\mathbf{q}_0$.
- Pick a uniform randomly momentum $\mathbf{p}$.
- Simulate the system according to the differential equations mentioned above, to produce a new $(\mathbf{q}', \mathbf{p}')$.
- Use this $(\mathbf{q}', \mathbf{p}')$ as the proposed point from the proposal.
- Use metropolis-hastings accept-reject.

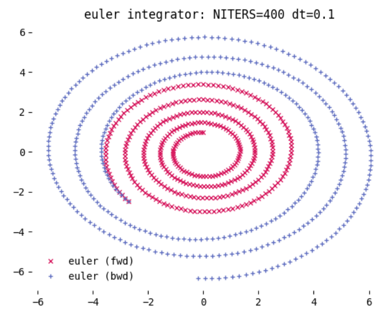
4.3 Hamiltonian for our simulation

If we have a a target probability distribution $Prob(\mathbf{q}) : \mathbb{R}^n \to \mathbb{R}$ we choose the Hamiltonian to be $H(\mathbf{p}, \mathbf{q}) \equiv Prob(\mathbf{q}) + \mathbf{p}^T\mathbf{p}/2$.

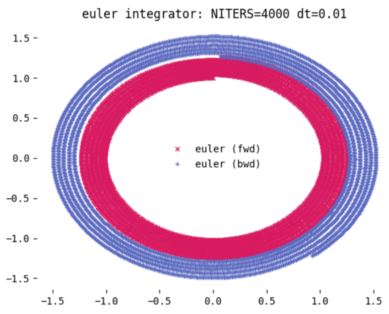
4.4 Simulation: Euler integration

Unfortunately, it turns out that running this simulation is in fact numerically unstable on using a naive simulation schemes. For example, let us say that we wish to simulate the orbit of a planet. Recall that we want the proposal to be symmetric: so, if we simulate the trajectory of the planet for N timesteps, each timestep of time δt, and then *reverse* the momentum of the planet, run the next phse of the simulation for N timesteps with timestep Δt, we should begin where we started. However, if we try to use the euler integration equations, here is what we see:

```
def euler(dhdp, dhdq, q, p, dt):
   pnew = p + -dhdq(q, p) * dt
   qnew = q + dhdp(q, p) * dt
   return (qnew, pnew)
```



We can clearly see the forward trajectory (in pink) spiralling out, and the backward trajectory (in blue), spiralling out even more. We can attempt to fix this by making Δt smaller:

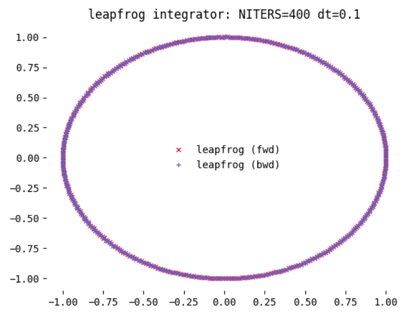


to no avail. Indeed, this is a **fundamental limitation of euler integration**. Hence, we will need to explore more refined integration schemes.

4.5 Simulation: Symplectic integrator

The type of integrators that will allow us to get 'reasonable orbits' that do not decay with time are knows as *symplectic integrators*. (An aside: the word *symplectic* comes from Weyl, who substituted the latin root in the word *complex* by the corresponding greek root. It is a branch of differential geometry which formalizes the consructs needed to carry out hamiltonian mechanics on spaces that are more complicated that Euclidian $\mathbb{R}^n$).

```
## dq/dt = dH/dp|_{p0, q0}, dp/dt = -dH/dq|_{p0, q0}
def leapfrog(dhdp, dhdq, q0, p0, dt):
    p0 += -dhdq(q0, p0) * 0.5 * dt # kick: half step momentum
    q0 += dhdp(q0, p0) * dt # drift: full step position
    p0 += -dhdq(q0, p0) * 0.5 * dt # kick: half step momentum
    return (q0, p0)
```



It is clear from the plots that the leapfrog integrator is stable: orbits stay as orbits.

4.6 Simulation: Final

We no longer need to choose how many steps to walk with a no-U-turn-sampler. It prevents us from re-walking energy orbits, by detecting when we have completed traversing an orbit and are going to take a "U-turn". The details are quite complex, so we may not cover this here.

5 Discontinuous Hamiltonian monte carlo

What if our distribution P is *discontinuouse*? How do we perform MCMC in that case?

5.1 Hamilton's equations under laplace momntum

$$\frac{d\mathbf{q}}{dt} = \mathbf{m}^{-1} \odot sign(\mathbf{p}) \qquad \frac{d\mathbf{p}}{dt} = -\nabla_q V(q)$$

where $\odot$ is component-wise multiplication. If we know that $\mathbf{p}$ does not change sign, then our dynamics are correct; This is very different from the usual hamilton's equations, where we need to know the *magnitude* of $\mathbf{p}$.

So, as long as we know that $\mathbf{p}$ has not changed sign, we can hold $sign(\mathbf{p})$ consant and use:

$$q(t+\epsilon) = q(t) + \epsilon m^{-1} \odot sign(\mathbf{p}(t))$$

Thus, we can jump across multiple discontinuities is $V(q)$ as long as we are aware that $sign(p(t))$ does not change.

TODO: write about the sudden drop in U when we cross a barrier.

6 Low discrepancy sequences

Low discrepancy sequences are sequences of numbers that more evenly distributed than pseudorandom numbers in high dimensional space. Hence, simulations which use Low discrepancy sequences generally approximate integrals faster than psuedo-randomly generated points.

Formally, let us consider the S dimensional half-open cube $\mathbb{I}^S \equiv [0,1)^S$. assume we have a set of points $P \subseteq \mathbb{I}^S$, and a sub-interval $B \subseteq I^S$, where a sub-interval is a subset of the form $B \equiv \prod_{i=1}^{S} \{x \in \mathbb{I} : a_i \leq x \leq b_i\}$.

Given a universe set X and two subsets $Large, Small \subseteq X$, we define the amount of containment of $Small$ in $Large$ to be $C(Small, Large) \equiv \frac{|Large \cap Small|}{|Large|}$. Intuitively, this measures the fraction of $Small$ that is in $Large$. We now define the discrepancy of the set of points P relative to the sub-interval B as:

$$\begin{aligned} D(B,P) &\equiv \left|C(P,B) - C(B,\mathbb{I}^S)\right| \\ &= \left|\frac{|B \cap P|}{|P|} - \frac{|B \cap \mathbb{I}^S|}{|\mathbb{I}^S|}\right| \\ &= \left|\frac{|B \cap P|}{|P|} - \frac{Volume(B)}{1}\right| \\ &= \left|\frac{|B \cap P|}{|P|} - Volume(B)\right| \end{aligned}$$

So, the discrepancy is measuring if P fits within B the way B fits within the full space.

Now, the **worst-case-discrepancy** is defined as the maximum discrepancy over all sub-intervals:

$$D^\star(P) \equiv \max_{B \in \mathcal{J}} D(B,P)$$

where $\mathcal{J} \subseteq 2^{\mathbb{I}^S}$ is the set of all sub-intervals:

$$\mathcal{J} \equiv \{\{x \in \mathbb{I}^S : l[i] \leq x[i] \leq r[i]\ \forall i\} : \vec{l}, \vec{r} \in \mathbb{I}^S\}$$

The goal is a *low discrepancy sequence* is to minimise the worst-case-discrepancy.

6.1 Error bounds for Numerical integration

6.2 Sobol sequences

Sobol sequences are an example of low-discrepancy sequences

7 Future Work!

I've left the topic names up because they have interesting ideas, which I have sketched out in brief. Writing them down formally will take far too long; Hence, they've been left here [for the purposes of the project]. I will probably update this document even after the project, once I have grokked this material better.

8 Slice Sampling

9 No-U-Turn sampling