New Monte Carlo Methods Based on Hamiltonian Dynamics

Radford M. Neal, University of Toronto

`http://www.cs.utoronto.ca/∼radford`

Talk presented at MaxEnt 2011, 14 July 2011

Part I: The Hamiltonian Monte Carlo Method

("Hybrid Monte Carlo", Duane, Kennedy, Pendleton, and Roweth, 1987)

Sampling with Hamiltonian Dynamics

1. Let the distribution of interest have density $\pi(q) = (1/Z)\exp(-U(q))$, where $U(q)$ is the "potential energy" at "position" q.

2. Introduce extra momentum variables, p, of the same dimension as q, and define a "kinetic energy", $K(p)$. Typically, $K(p) = p^Tp/2$.

3. Define the "Hamiltonian" to be $H(q, p) = U(q) + K(p)$, and let the joint density of q and p be proportional to $\exp(-H(q, p))$.

   Note: q and p are independent, and the marginal density for q is π.

4. Repeatedly do the following:
   - Sample p from its density, which is $N(0, I)$ when $K(p) = p^Tp/2$.
   - Find a proposal (q^*, p^*) from (q, p) by simulating *Hamiltonian dynamics* for some amount of fictitious time.
   - Accept or reject (q^*, p^*) as the next state according to the usual Metropolis acceptance probability, $\min[1, \exp(-H(q^*, p^*) + H(q, p))]$.

Hamilton's Dynamical Equations

Hamiltonian dynamics is defined by the following differential equations for how the state (q, p) evolves with "time", t:

$$\frac{dq_i}{dt} = \frac{\partial H}{\partial p_i}, \qquad \frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i}$$

When $H(q, p) = U(q) + K(p)$ and $K(p) = p^T p/2$, we see that

$$\frac{dq_i}{dt} = p_i, \qquad \frac{dp_i}{dt} = -\frac{\partial U}{\partial q_i}$$

so the "position" variables get pushed by the "momentum" variables, which are themselves controlled by the gradient of the potential energy.

The mapping $(q, p) \to (q^*, p^*)$ obtained by applying these dynamical equations for some time interval has two crucial properties:

- The mapping leaves H invariant: $H(q^*, p^*) = H(q, p)$.
- The mapping preserves volume in (q, p) space — ie, the determinant of the Jacobian matrix for the transformation is one.

The Leapfrog Discretization

These dynamical equations can seldom be solved exactly. We need to approximate them using some stepsize, ε, for time. The "leapfrog" discretization is usually used:

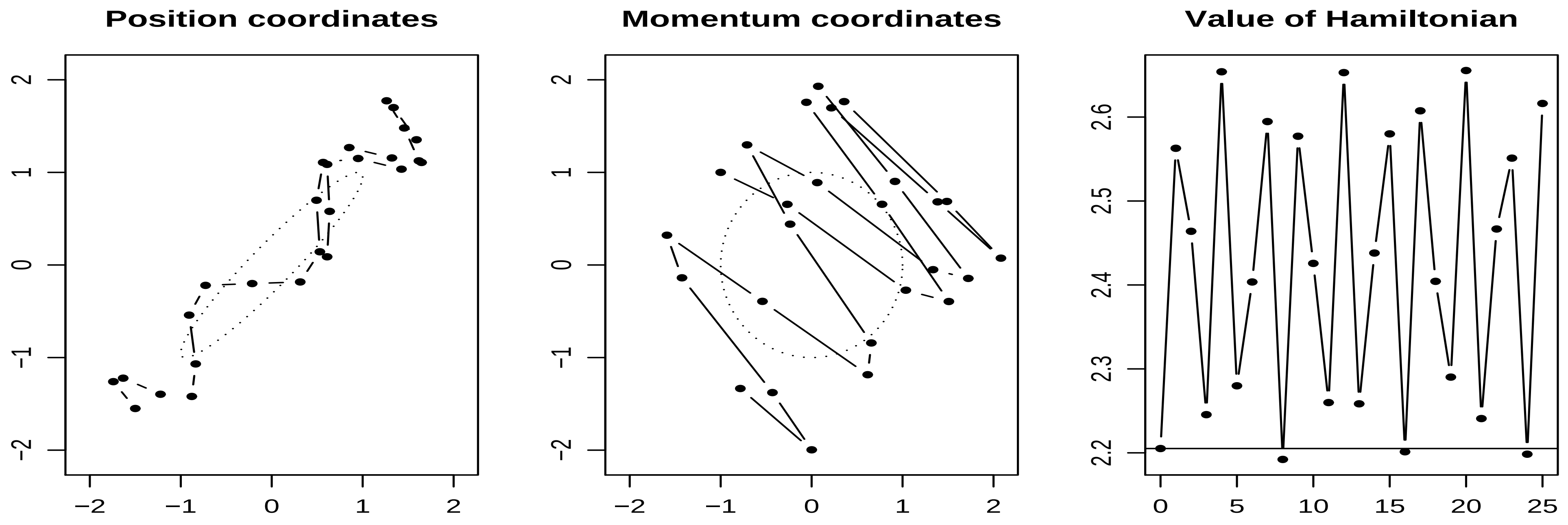
$$\begin{aligned}
p_i(t+\varepsilon/2) &= p_i(t) \;-\; (\varepsilon/2)\,\frac{\partial U}{\partial q_i}(q(t)) \\
q_i(t+\varepsilon) &= q_i(t) \;+\; \varepsilon\, p_i(t+\varepsilon/2) \\
p_i(t+\varepsilon) &= p_i(t+\varepsilon/2) \;-\; (\varepsilon/2)\,\frac{\partial U}{\partial q_i}(q(t+\varepsilon))
\end{aligned}$$

Although this approximation does not keep H constant, crucially, it *does* preserve volume exactly (apart from floating-point roundoff error).

The error in H is corrected by the Metropolis accept/reject decision, so the final result is exact despite the approximate dynamics. (Although if ε is too big, the error in H will be large, and acceptances will be rare.)

HMC for a Bivariate Gaussian

Here is a trajectory that produces an HMC proposal for a bivariate Gaussian distribution, with means of 0, standard deviations of 1, and correlation 0.95. The trajectory is simulated with 25 leapfrog steps with $\varepsilon = 0.25$, from initial state $q = [-1.50, -1.55]^T$ and $p = [-1, 1]^T$:



The error in H at the end of the trajectory is $+0.41$, so the probability of accepting the end-point as the next state is $\exp(-0.41) = 0.66$.

Some Properties of Hamiltonian Monte Carlo

- The trajectory used to propose a new state proceeds systematically in one direction — until forced to turn when it reaches a low-density region — rather than in a random walk. Hence, with stepsize ϵ a trajectory of L leapfrog steps will often move a distance εL.

- The stepsize ε must be tuned to be small enough for the dynamics to be stable, and the error in H to be small, big enough for efficient movement.

- The number of leapfrog steps, L, must be tuned so that a trajectory (of duration εL in fictitious time) leads to a nearly independent point.

- HMC is invariant to translation and rotation of the coordinate system for q (while the kinetic energy for p is kept fixed).

  Arbitary invertible linear transformations for q (changing $U(q)$) have no effect if the inverse transformation is applied to p (changing $K(p)$).

Scaling Characteristics of Hamiltonian Monte Carlo

If we increase the dimensionality, d, by adding independent replicas...

- The optimal HMC stepsize decreases as $d^{-1/4}$.
- The optimal number of leapfrog steps in a trajectory increases as $d^{1/4}$ (maintaining the length of the trajectory in fictitious time).
- The number of leapfrog steps needed to reach a nearly independent point grows as $d^{1/4}$.

Compare this with random-walk Metropolis updates, where the optimal proposal standard deviation decreases as $d^{-1/2}$, and the number of updates needed to reach a nearly independent point grows as d.

Part II: Billiard Monte Carlo

HMC with Reflection off Boundaries

As a prelude to the new “billiard Monte Carlo” method, consider using HMC when some q_i are constrained by $q_i \geq \ell_i$ and/or by $q_i \leq u_i$. Can Hamiltonian dynamics be modified to handle this?

Yes. We just imagine that $U(q)$ increases very rapidly over a short range as q_i starts to violate the constraint. The large value of $\partial U/\partial q_i$ will reduce p_i, until it reaches zero and then changes sign, at which point q_i will move away from the boundary and p_i will increase with opposite sign.

End result: q_i bounces off the boundary, and p_i is negated.

This process replaces the update for q_i in the leapfrog steps. If $U(q)$ is constant within the boundaries, this bouncing off the walls is all that happens — a special case of Hamiltonian dynamics called “billiards”.

The Idea of Billiard Monte Carlo

A potential energy function that is constant where it is not infinite gives a distribution of interest that is uniform over some bounded region. So it seems that the special case of billiards will be useful only fairly rarely.

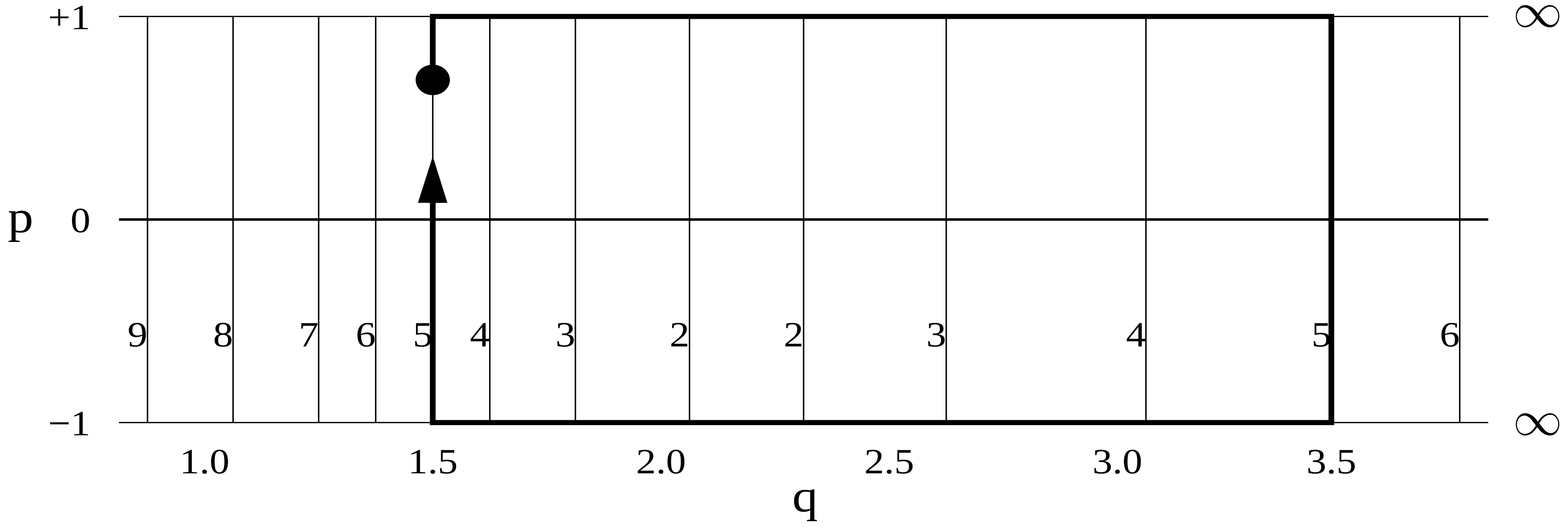
But when we are using Hamiltonian dynamics for sampling, we are free to choose the *kinetic energy* however we wish!

We can choose a kinetic energy that is constant inside some region, producing a sort of "reverse billiards", in which p bounces off walls, while q flips to an opposite point of equal potential energy.

Billiard Monte Carlo in One Dimension

Suppose that q is one-dimensional, with density $\pi(q) \propto \exp(-U(q))$. The momentum, p, is also one-dimensional. Let its kinetic energy be 0 for $p \in [-1, +1]$ and ∞ elsewhere, giving a uniform distribution over $[-1, +1]$.

Here is an example contour plot of $H(q, p) = U(q) + K(p)$, along with a trajectory following Hamiltonian dynamics:



The speed along the left part of the trajectory is higher than along the right part, because $|U'(1.5)|$ is greater than $|U'(3.5)|$. Movement from $q = 1.5$ to $q = 3.5$ (and back) is instantaneous, since $|K'(\pm1)| = \infty$.

Simulating Trajectories by Solving Equations (1D Case)

We can compute a trajectory of duration D starting at (q_0, p_0) as follows:

1) Set $t \leftarrow 0$, $p \leftarrow p_0$, $q \leftarrow q_0$.

2) Set $v \leftarrow -U'(q)$.

3) Find the time, δ, until p reaches ± 1 as it moves with velocity v.
   If $t + \delta \leq D$, set $p \leftarrow p + (D - t)v$ and exit.

4) Otherwise, set $t \leftarrow t + \delta$, set $p \leftarrow p + \delta v$ (p will be ± 1), solve the equation $U(q^*) = U(q)$ for $q^* \neq q$, and set $q \leftarrow q^*$.
   Return to step (2).

Solving the equation in step (4) is the crucial part of the computation. The simulated trajectory will be exact if this equation is solved exactly. This can done reasonably efficiently, to machine precision, using a superlinearly-convergent method such as Newton-Raphson iteration. There is then no need for an accept/reject test.

BMC with Momentum Uniform Over a Hypersphere

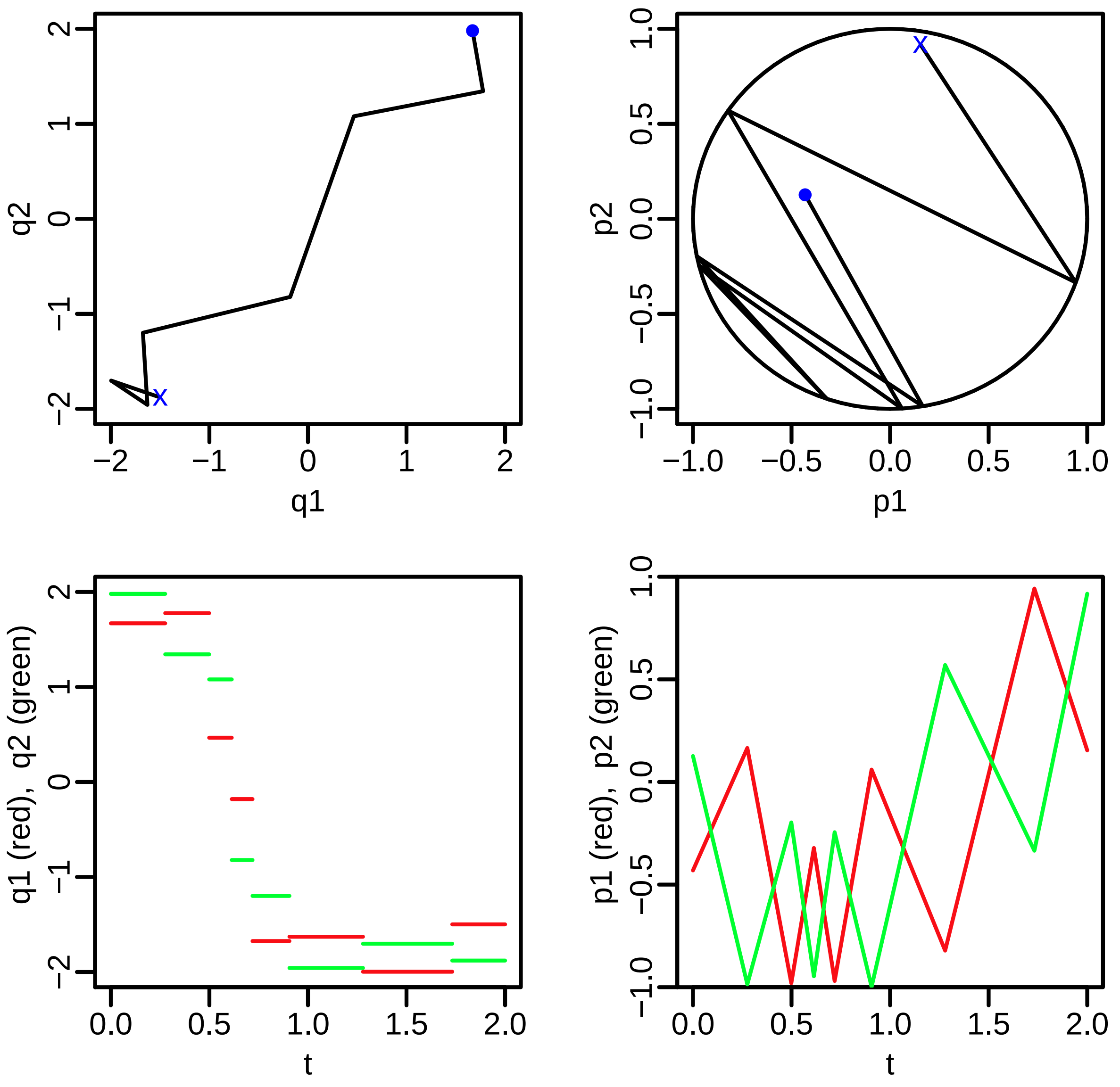
When the dimension of p is more than one, there are many possible choices for the region in which $K(p)$ is zero (with $K(p) = \infty$ elsewhere). One natural choice is a hypersphere of radius 1 centred at $p = 0$.

With this choice, a trajectory of duration D starting at (q_0, p_0) is computed as follows:

1) Set $t \leftarrow 0$, $p \leftarrow p_0$, $q \leftarrow q_0$.
2) Set $v \leftarrow -\nabla U(q)$.
3) Find the time, δ, until p intersects the surface of the hypersphere, as it moves with velocity v. If $t + \delta \leq D$, set $p \leftarrow p + (D - t)v$ and exit.
4) Otherwise, set $t \leftarrow t + \delta$, set $p \leftarrow p + \delta v$, solve the equation $U(q + px) = U(q)$ for scalar $x > 0$, and set $q \leftarrow q + px$. Return to step (2).

Note: In step (4), the new p will be on the surface of the hypersphere, and will also be normal to the surface. So it gives the direction for q to move.

Hypersphere BMC for a Bivariate Gaussian



Example of a hypersphere BMC trajectory sampling a bivariate Gaussian with variances of 1 and correlation 0.95.

The top plots show the paths of q and p. The bottom plots show how q and p vary with fictitious time, t.

Moving to Different Potential Energy Contours

A BMC trajectory stays on one contour of the potential energy (ie, at a fixed value of the log density for q). Clearly, we need to move between contours to properly sample the distribution.

We can do this by alternating BMC updates with any update that can change the potential energy, such as a simple random-walk Metropolis update. We hope that the BMC updates will be able to quickly move to distant points, while the other updates take care of moving between values for the density.

Fixing the Trajectory Length by Number of Bounces

It can be difficult to guess how long a trajectory should be in fictitious time. It seems better to fix the number of bounces, which determines the amount of computation time.

Here's how to do this validly, for a trajectory with K bounces:

- Pick B uniformly from $\{0, \ldots, K\}$.
- Simulate backwards in time from the start point until just before bounce $B+1$.
- Simulate forwards in time from the start point until just before bounce $K-B+1$.
- Pick a point from the whole simulated trajectory uniformly with respect to time.

This is analogous to the "windowed" variant of HMC; more elaborate versions can reduce the probability of picking a point close to the start.

Properties of Hypersphere BMC

- Hypersphere BMC is invariant to translation and rotation of the coordinates for q.
- If the trajectory length is determined by the number of bounces, hypersphere BMC is invariant to equal scaling of the coordinates for q. There is therefore no “stepsize” parameter to tune, as in HMC or random-walk Metropolis.
- The time for a trajectory (or number of bounces) should be tuned so that a trajectory leads to a distant point.
- The number of bounces needed to reach a distant point with hypersphere BMC seems to scale with dimensionality as $d^{1/2}$.
- Unfortunately, if simple a random-walk update (eg, Metropolis) is used to move between contours of potential energy, the number needed to reach a nearly independent point scales as d. A better method for changing the potential energy is needed.

BMC with Momentum Uniform Over a Hypercube

Another option is $K(p) = 0$ for $p \in [-1, +1]^d$ and $K(p) = \infty$ elsewhere.

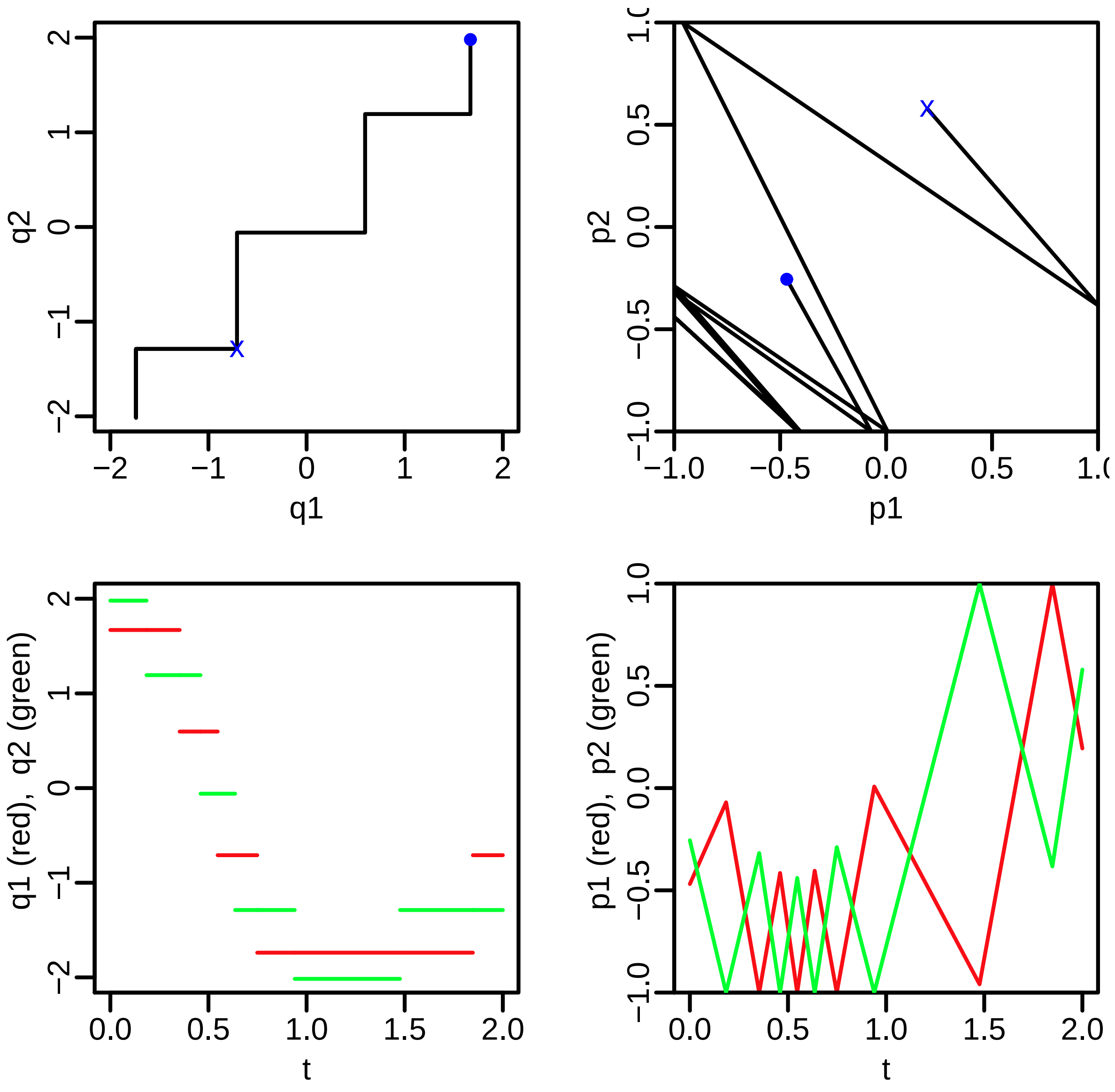
With this choice, a bounce will change only one coordinate of q.

Often, recomputing the potential energy when only one coordinate of q has changed is much faster than when all of q has changed (assuming intermediate results from the previous computation were saved).

Hypercube BMC resembles "overrelaxation" methods for Gaussian distributions, but is applicable to any distribution.

Like overrelaxation, analysis of hypercube BMC seems difficult. However, empirically it explore some distributions less efficiently than hypersphere BMC, at least before accounting for the gain from incremental computation.

Hypercube BMC for a Bivariate Gaussian



Example of a hypercube BMC trajectory sampling a bivariate Gaussian with variances of 1 and correlation 0.95. The start point is the same as in the hypersphere BMC example.

The top plots show the paths of q and p. The bottom plots show how q and p vary with fictitious time, t.

Part III: Hamiltonian Importance Sampling

Review of Importance Sampling

We want to estimate expectations with respect to the distribution with probability density $\pi(x) = f(x)/Z_f$, where $Z_f = \int f(x)dx$.

Suppose we can't sample from $\pi(x)$. Instead, we sample from the distribution with density proportional to $g(x)$, with normalizing constant $Z_g = \int g(x)dx$.

Given points $x_1, \ldots, x_n$ drawn from g, we can estimate $E_\pi[a]$, the expectation of $a(x)$ with respect to π, by

$$\sum_{i=1}^{n} w_i a(x_i) \Big/ \sum_{i=1}^{n} w_i$$

Here, $w_i = f(x_i)/g(x_i)$ is the *importance weight* for point x_i.

We can estimate the ratio Z_f/Z_g by $(1/n) \sum\limits_{i=1}^{n} w_i$.

Difficulties with Importance Sampling

When $\pi(x)$ is complex and high-dimensional, it is difficult to choose a distribution $g(x)$ that satisfies all of the following requirements:

1) *It is a good approximation to π.* If not, the importance weights will be highly variable, and the effective sample size when estimating $E_\pi[a]$ will be very small.

2) *We can feasibly sample from it (independently).* Easily-sampled distributions like Gaussians aren't good approximations. We need something like the distribution defined by K Metropolis updates from a start state drawn from a broad distribution.

3) *We can compute $g(x)$, and hence the importance weights.* Sadly, the density for the distribution defined by K Metropolis updates involves an infeasible integral over all intermediate states.

Probability Densities for Transformations of Variables

Before introducing a new importance sampling scheme, I'll review a crucial topic: How probabability densities transform.

Let the multi-dimensional variable x have density $\pi_x(x)$. Define a transformed variable $y = h(x)$, where h is differentiable and invertible.

The probability density for y is given by

$$\pi_y(y) \quad = \quad \pi_x(h^{-1}(y)) \,/\, |\det\, h'(h^{-1}(y))|$$

where $h'(x)$ is the Jacobian matrix for the transformation.

Simple example: If $y = \alpha x$, then $\pi_y(y) = \pi_x(y/\alpha)/\alpha^d$, where d is the dimensionality of x and y.

Basic Hamiltonian Importance Sampling

Let $x = (q, p)$ and make $\pi(x)$ proportional to $f(x) = \exp(-H(q, p)/T)$, with $H(q, p) = U(q) + p^T p/2$ and "temperature" $T = 1$,

We define an importance sampling distribution for (q, p) as follows:

- Generate an initial q from some simple, broad distribution, and an initial p from its distribution at some high T_0 (which is $N(\mathbf{0}, T_0\mathbf{I})$).
- Apply K leapfrog steps to move from this initial (q, p) to a final (q, p). **Note:** The Jacobian for each such transformation is one.
- After each leapfrog step, multiply p by some factor, α, less than one. This cools the system towards the desired temperature of $T = 1$. **Note:** The Jacobian for this multiplication is α^d, where $d = \dim(p)$.

Randomness comes only from the generation of the initial state. The subsequent deterministic transformation has Jacobian α^{Kd}, so we can easily compute the density of the final point, and its importance weight.

Details of Basic Hamiltonian Importance Sampling

We generate each $x_i = (q_i, p_i)$ and associated weight, w_i, as follows:

1. Generate $q_i^{(0)}$, say uniformly over some bounded region of volume V_0. Generate $p_i^{(0)}$ at temperature T_0, with density $N_0(p)$.

2. For $k = 1, \ldots, K$:

   Perform one (or more) leapfrog steps with stepsize ϵ to produce $(q_i^{(k)}, \tilde{p}_i^{(k)})$ from $(q_i^{(k-1)}, p_i^{(k-1)})$.

   Let $p_i^{(k)} = \alpha \tilde{p}_i^{(k)}$.

3. Let $q_i = q_i^{(K)}$ and $p_i = p_i^{(K)}$.

4. Let $w_i = \exp(-H(q_i, p_i)) \;/\; (N_0(p_i^{(0)})/\alpha^{Kd} V_0)$, where d is the dimensionality of p (and q).

We will need to tune T_0, ϵ, α, and K to get good performance.

Properties of Hamiltonian Importance Sampling

- It can be used to estimate the normalizing constant for Z_f, by $(1/n) \sum_i w_i$, as well as expectations with respect to π
- It's exact, apart from round-off and statistical errors (no error from using a finite leapfrog stepsize).
- It uses a annealing-style importance sampling distribution that will tend to visit various different local modes.
- We *can* compute the correct weights for this importance sampling distribution, even though it is very complex.
- It cools the system by extracting energy (from the momentum) a bit at a time, so the system passes through all intermediate energy states.

This last property is important for some distibutions with a “phase transition”, and also because it eliminates the need to determine a detailed schedule of temperatures for intermediate distributions.

When Would We Expect This to Work?

For importance sampling to work well,

- All points typical of $\pi(x)$ must have a reasonably high probability of being sampled. **This is crucial.**
- Points not typical of $\pi(x)$ must not be sampled too often. But this is less crucial.

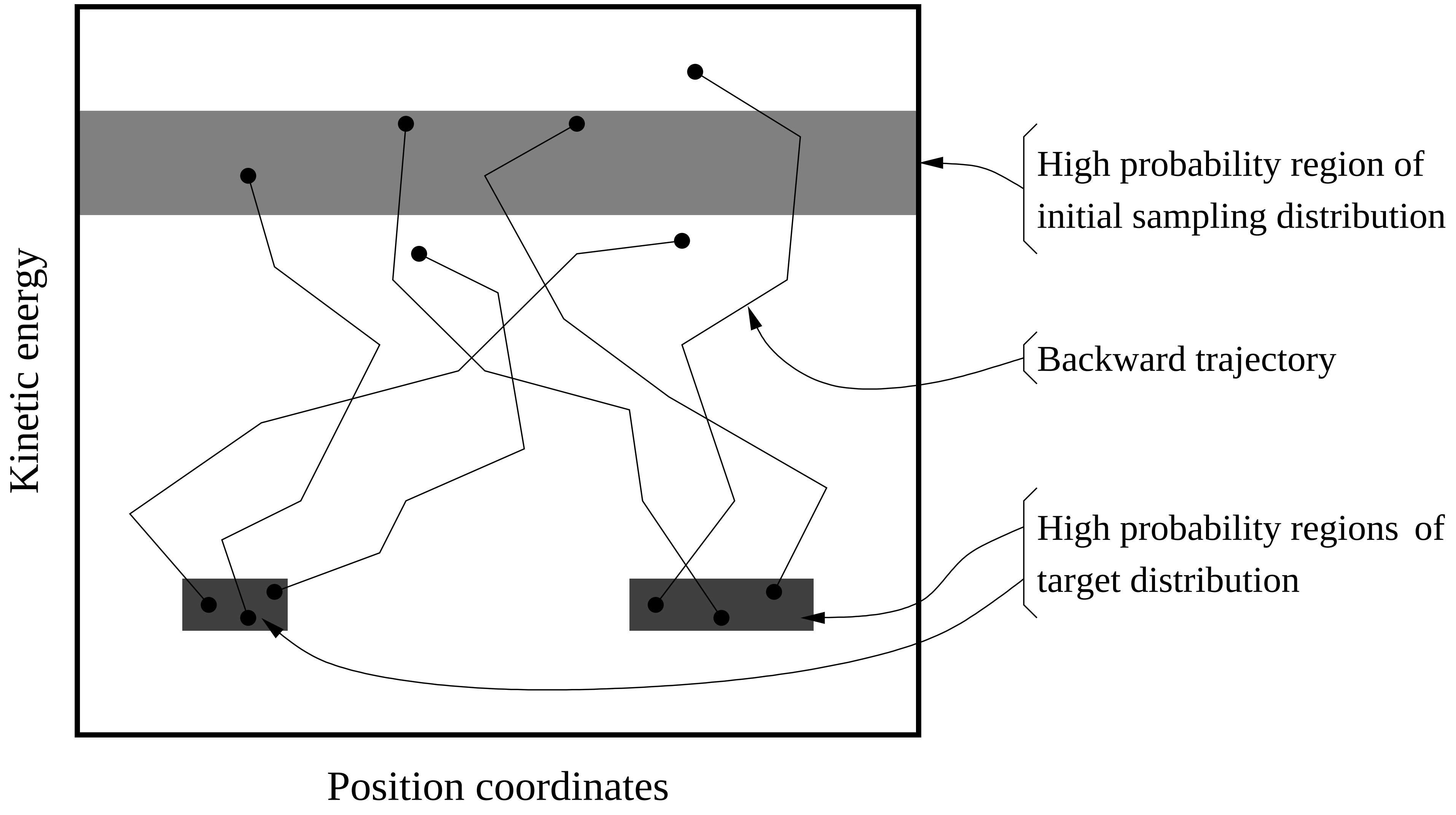
To check how well Hamiltonian importance sampling will work, we can imagine *backward* trajectories with *division* of p by α, starting from points drawn according to π. These backward trajectories must lead to points typical of the initial distribution (uniform for q, temperature T_0 for p).

There's reason to doubt this:

- We'd need to make a good guess at K to match the cooling time.
- There may be *no* good value for K, if there are multiple modes with different characteristics.

Picturing the Problem

Here's a picture of how the backward trajectories might not reach the region of high initial probability:



High probability region of initial sampling distribution
Backward trajectory
High probability regions of target distribution
Kinetic energy
Position coordinates

Picking the Number of Steps Randomly

We can fix this problem by choosing the number of leapfrog steps randomly, from some range, $K_{\min}, \ldots, K_{\max}$. If we choose K_i, we then use the same procedure as before to produce $(q_i, p_i) = (q_i^{(K_i)}, p_i^{(K_i)})$.

But: To compute the importance weight, we now need to add together the probability of generating (q_i, p_i) using *any* value for K, not just K_i.

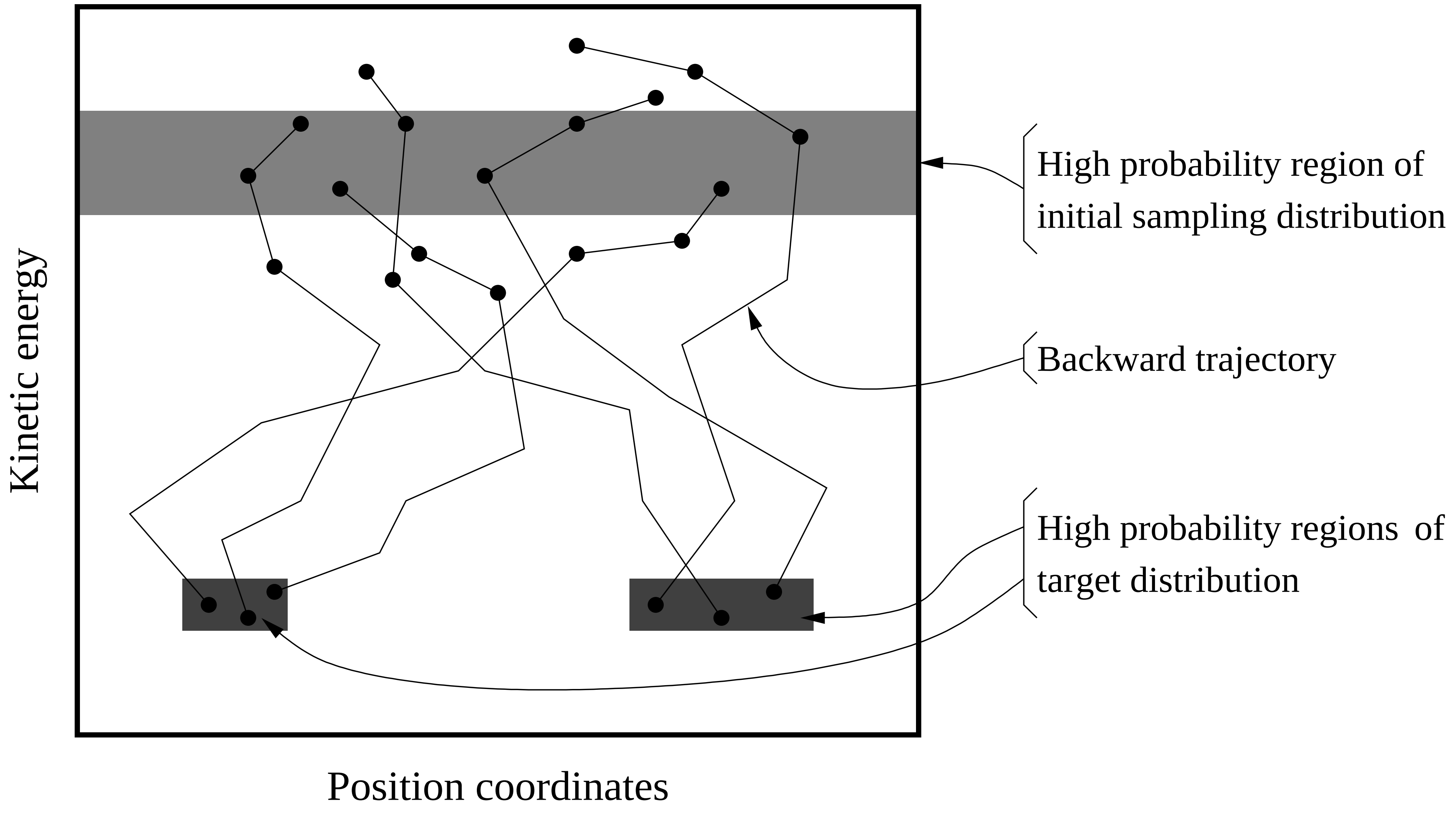
To do this, we simulate backwards (dividing p by α) from $(q_i^{(0)}, p_i^{(0)})$ for $K_{\max} - K_i$ leapfrog steps, to get $(q_i^{(-1)}, p_i^{(-1)}), \ldots, (q_i^{(K_i - K_{\max})}, p_i^{(K_i - K_{\max})})$.

The total probability of generating (q_i, p_i) can then be computed as

$$\frac{1}{K_{\max} - K_{\min} + 1} \sum_{K=K_{\min}}^{K_{\max}} K_0(p_i^{(K_i - K)}) \,/\, \alpha^{Kd} V_0$$

Picturing this Solution

Here's how the problem seen before goes away if we randomizing the number of leapfrog steps to the previous number plus -1, 0, or $+1$:


High probability region of initial sampling distribution
Backward trajectory
High probability regions of target distribution
Kinetic energy
Position coordinates

Ensuring Equipartition of Kinetic Energy

Another potential problem: Backward trajectories from typical points may result in states at the initial temperature that aren't in equilibrium with respect to partition of kinetic energy among momentum variables.

Example: When simulating a molecular cluster, backward trajectories will lead to atoms escaping from the cluster at various times, with various kinetic energies, which may be unlikely to interact thereafter.

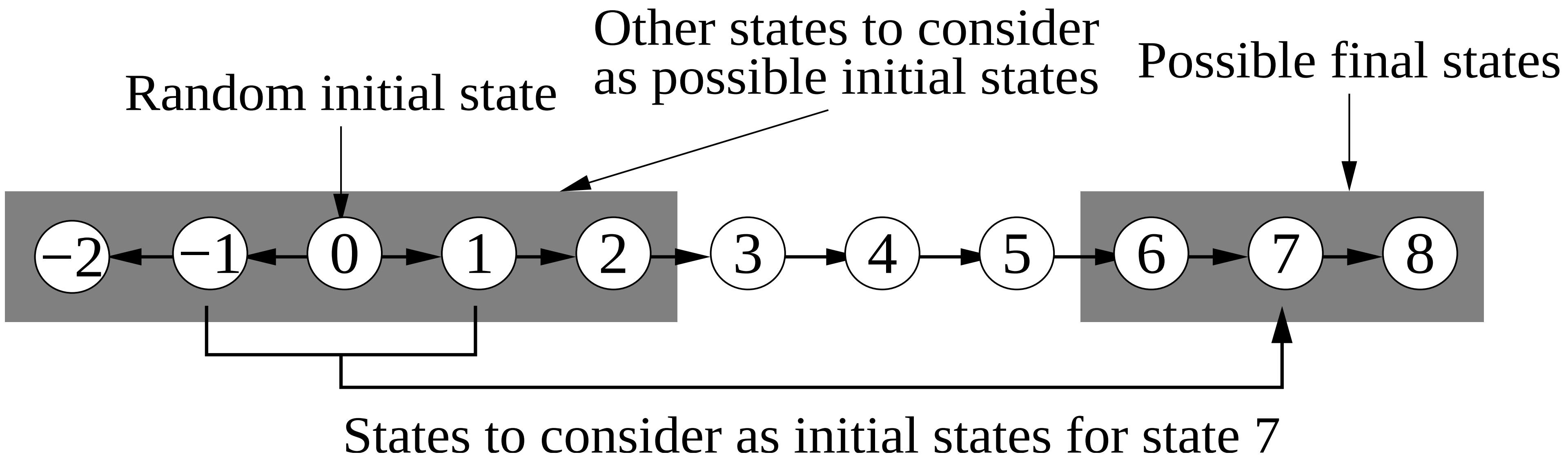
A solution: Periodically mix the momentum variables by doing a rotation in momentum space, using a series of random rotation axes and angles. Choosing randomly avoids the possibility that we're unlucky enough to fix on some particularly bad rotations, but for good performance, almost all sequences of random choices must be good.

Simultaneously Producing Multiple Trajectories

Rather than get just one sampled state from a trajectory K_i steps long, with K_i randomly chosen from $K_{\min}$ to $K_{\max}$, we can with little extra effort get sampled states for *all* trajectory lengths from $K_{\min}$ to $K_{\max}$.

We just simulate forward for $K_{\max}$ steps, and backward for $K_{\max}-K_{\min}$ steps, then look at the $K_{\max}-K_{\min}+1$ trajectories that start at the random initial state.

Here's a picture when $K_{\min} = 6$ and $K_{\max} = 8$:



Keeping the Prior at High Temperatures

For a Bayesian model, it's not desirable for the prior to be downweighted at high temperature — only the likelihood should have less effect. But if the potential energy is the log posterior density, the large momentum at the start of a Hamiltonian importance sampling run will push the parameters to values with low prior probability.

Solution: Define the potential energy as the log likelihood, and incorporate the prior via a form of "slice sampling".

At the start of each leapfrog step, sample a "slice level" uniformly distributed from zero to the prior density of the current point. After the leapfrog step, check whether the prior density is below this level, and if so, restore the state before the leapfrog step and negate the momentum.

Problems with Hamiltonian Importance Sampling

After all this work, Hamiltonian importance sampling still has problems:

- As for other annealing methods, it's hard to decide what temperature to start at, and how fast to cool.
- It's hard to set the leapfrog stepsize. Different stepsizes may be appropriate at different points in the run, but a schedule of varying stepsizes would be hard to tune, and may not work since different runs can go to different modes. In the MCMC context, one can do updates with several different stepsizes, hoping at least one is appropriate, but that's not an option here (or is it?).
- As for other importance sampling methods, it's hard to tell how well it's working. The variance of importance weights can be very high without this being apparent in the sample actually obtained.

Applying the Idea to Other MCMC Methods

Producing an importance sampling distribution by "cooling" a simple high-temperature distribution is natural when sampling by Hamiltonian dynamics, where momentum variables already provide a "heat reservoir".

Can we do similar things for other MCMC methods, such as Metropolis?

We can define a heat reservoir variable, h, that is positive, with energy equal to h, and initialize h to a large value.

We can accept Metropolis proposals that increase the energy by Δ when $\Delta < h$, and change h to $h - \Delta$ if we accept (h will increase if $\Delta < 0$).

We can then cool the system after each Metropolis update by multiplying h by some factor α less than one.

Unfortunately, even if we consider the random proposal offsets to be given, the mapping this defines is not invertible.

Is there a fix? Does it work better with other methods, such as slice sampling?