

Looking for smart proposals that explore the sample space quickly

- Auxilliary variables.
- Use of derivatives of the target distribution.
- Hamiltonian Metropolis Hasting (MH).

Auxiliary variables in MH - hit and run

Proposed by Smith, R. L. (1984).¹

One iteration of hit-and-run.

- Draw a direction.
- Sample length along direction. (This defines proposal.)
- Accept or reject move with MH rate.

(Could be better than Gibbs sampling if directions are sampled wisely, and reasonable accept probability, which depends on the chance of getting back from $\mathbf{x}^*$ to $\mathbf{x}$.)

¹Efficient Monte Carlo procedures for generating points uniformly distributed over bounded regions. Operations Research, 32, 1296–1308.

Auxiliary variables in MH - slice sampling

Proposed by Neal, R. M. (2003).²

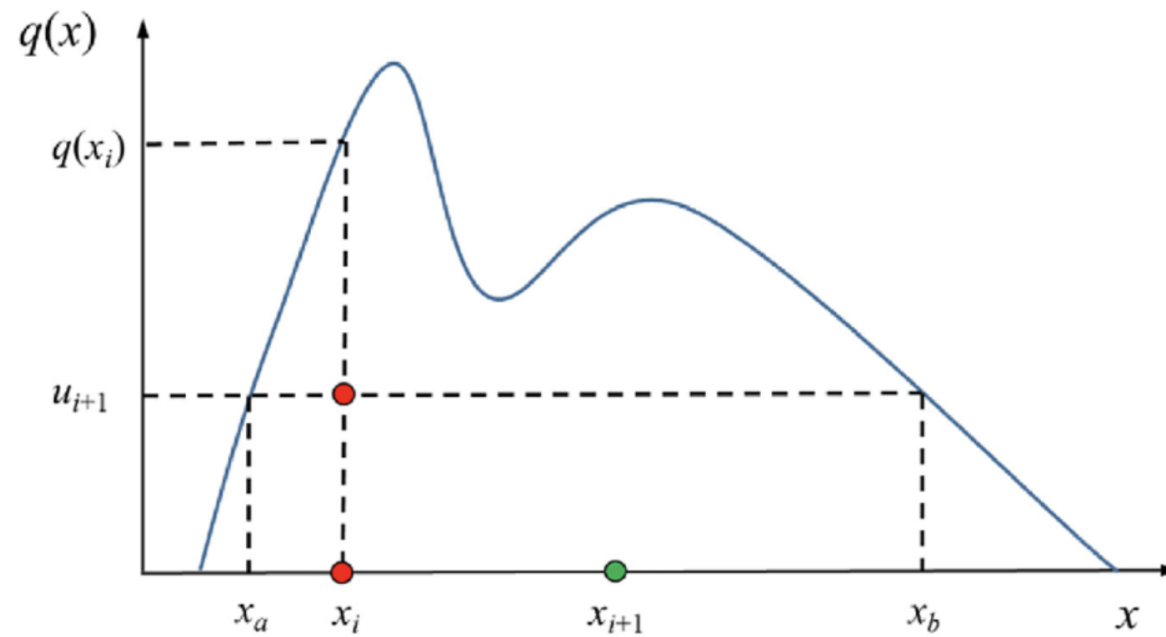
One iteration of slice sampling.

- Draw a level at the density axis $u \sim U(0, \pi(\mathbf{x}^b))$.
- Sample a variable that has a least this density level,
 $\mathbf{x} \sim U(\mathbf{x} : \pi(\mathbf{x}^{b+1}) > u)$.

(Can induce large steps in sample space, much larger than Gibbs sampling, if possible to sample at a level of the density.

²Slice sampling. Annals of Statistics, 705-741.

Illustration slice sampling



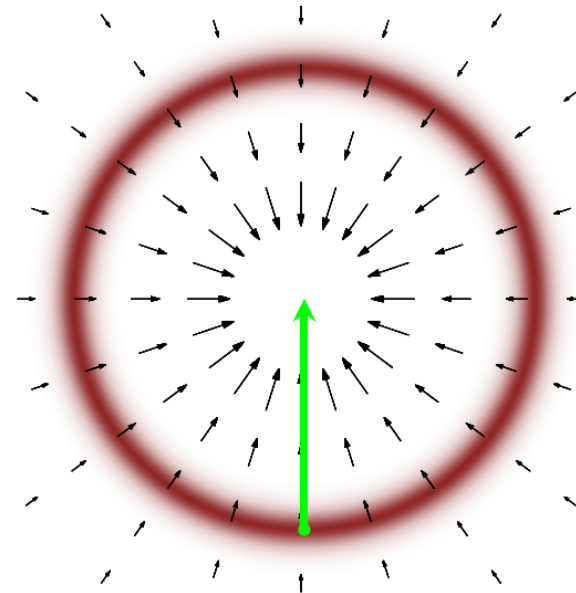
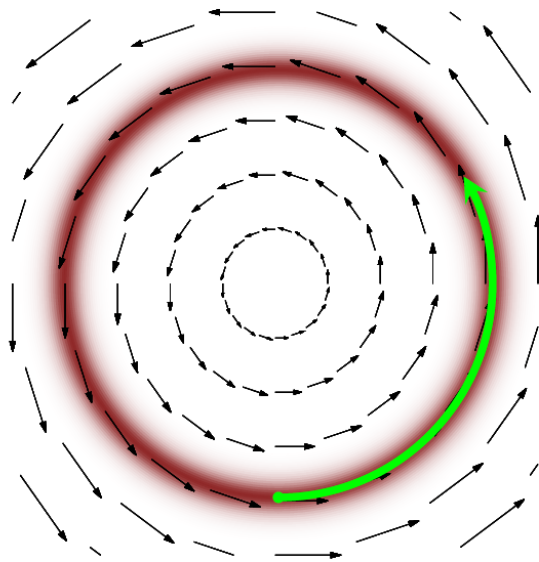
Hamiltonian Monte Carlo

Originally proposed by Duane et al. (1987).³

- Typical MCMC updates can only make small changes to the state (otherwise most updates will be rejected)
- Hamiltonian Monte Carlo uses derivative information on $\log(\pi(\mathbf{x}))$. In addition it introduces auxiliary variables to avoid a random walk behaviour and help to explore target distribution efficiently.
- Inspired from physics.

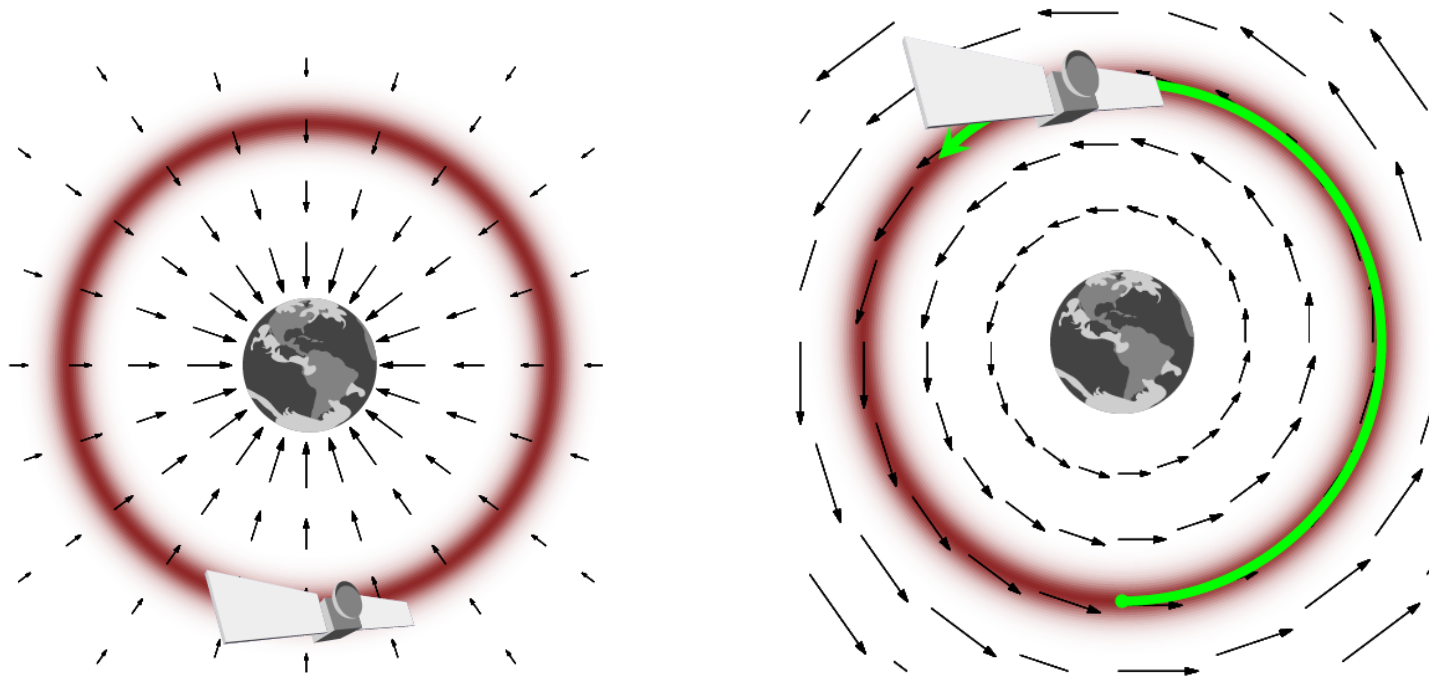
³Hybrid Monte Carlo. Physics Letters B, 195:216–222.

Exploration of the "typical set"



Finding a physical analogy

Consider a planet, a gravitational field and an orbit. Exploring the typical set then becomes a physical endeavor of placing a satellite in a stable orbit around the hypothetical planet.



Add auxiliary momentum variables

We switch to the notation of Betancourt (2018), arXiv preprint arXiv:1701.02434. Assume our target density is $\pi(q)$, then

$$q_n \rightarrow (q_n, p_n),$$

where p_n is an auxiliary *momentum* parameter.

We expand the D-dimensional parameter space into a 2D-dimensional space.

The Hamiltonian function

The canonical density of p and q is defined as:

$$\pi(p, q) \propto \exp(-H(p, q))$$

where $H(p, q)$ is the Hamiltonian function:

$$\begin{aligned} H(p, q) &= -\log(\pi(p, q)) \\ &= -\log(\pi(p \mid q)) - \log(\pi(q)) \\ &= \underbrace{K(p, q)}_{\text{Kinetic Energy}} + \underbrace{V(q)}_{\text{Potential Energy}}, \end{aligned}$$

Transition via Hamilton's equations

1. Sample the auxiliary momenta $p \sim \pi(p \mid q)$ (to be specified! Usually Gaussian).
2. Evolve the joint system via Hamilton's equations:

$$\begin{aligned}\frac{dq}{dt} &= +\frac{\partial H}{\partial p} = +\frac{\partial K}{\partial p} \\ \frac{dp}{dt} &= -\frac{\partial H}{\partial q} = -\frac{\partial K}{\partial q} - \frac{\partial V}{\partial q}\end{aligned}$$

after integrating Hamilton's equations we can return to the target parameter space by simply projecting away the momentum: $(q, p) \rightarrow q$.

Hamiltonian exact

Consider the simple example:

$$q \sim N(0, 1), \quad p|q \sim N(0, 1).$$

$$V(q) = q^2/2, \quad K(p|q) = p^2/2,$$

$$\frac{dq}{dt} = p, \quad \frac{dp}{dt} = -q$$

$$q(t) = r \cos(a + t), \quad p(t) = -r \sin(a + t)$$

For most systems Hamilton's equations cannot be solved like this.
Instead discrete-time numerical integrators are used.

Transition via Hamilton's equations: Implementation

The partial derivatives determine how q (position) and p (momentum) change over time and thus guide the transitions through regions of high probability and admit the efficient exploration of the entire target distribution.

In practice

- We can simulate differential equations by discretising time.
- This introduces errors, which are adjusted for by treating the whole procedure as a Metropolis-Hastings proposal, and an accept/reject step.

The Leapfrog algorithm

Leapfrog is the most popular method to propagate Hamiltonian dynamics.

Start by $q = q_0$, $p = p_0$.

For $i = 0, 1, \dots, T/\epsilon$,

- $p_{i+1/2} = p_i - \frac{\epsilon}{2} \frac{dV}{dq}(q_i)$
- $q_{i+1} = q_i + \epsilon p_{i+1/2}$
- $p_{i+1} = p_{i+1/2} - \frac{\epsilon}{2} \frac{dV}{dq}(q_{i+1})$

Tuning parameters are ϵ and T .

(There are adaptive heuristic methods for setting T and ϵ preserving detailed balance; NUTS (No-U-turn sampling.))

Mathematics of path for : $N_2(0, \Sigma)$

Start by $q = q_0$, $p = p_0$.

For $i = 0, 1, \dots, T/\epsilon$,

- $p_{i+1/2} = p_i - \frac{\epsilon}{2} \Sigma^{-1} q_i$
- $q_{i+1} = q_i + \epsilon p_{i+1/2}$
- $p_{i+1} = p_{i+1/2} - \frac{\epsilon}{2} \Sigma^{-1} q_{i+1}$

Illustration of Hamiltonian

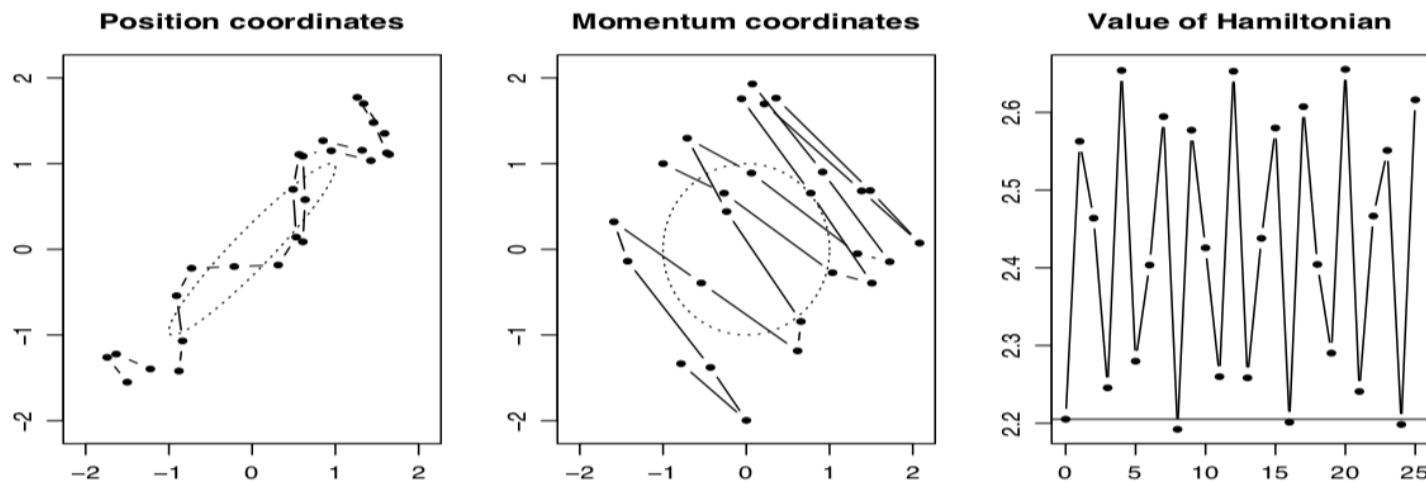


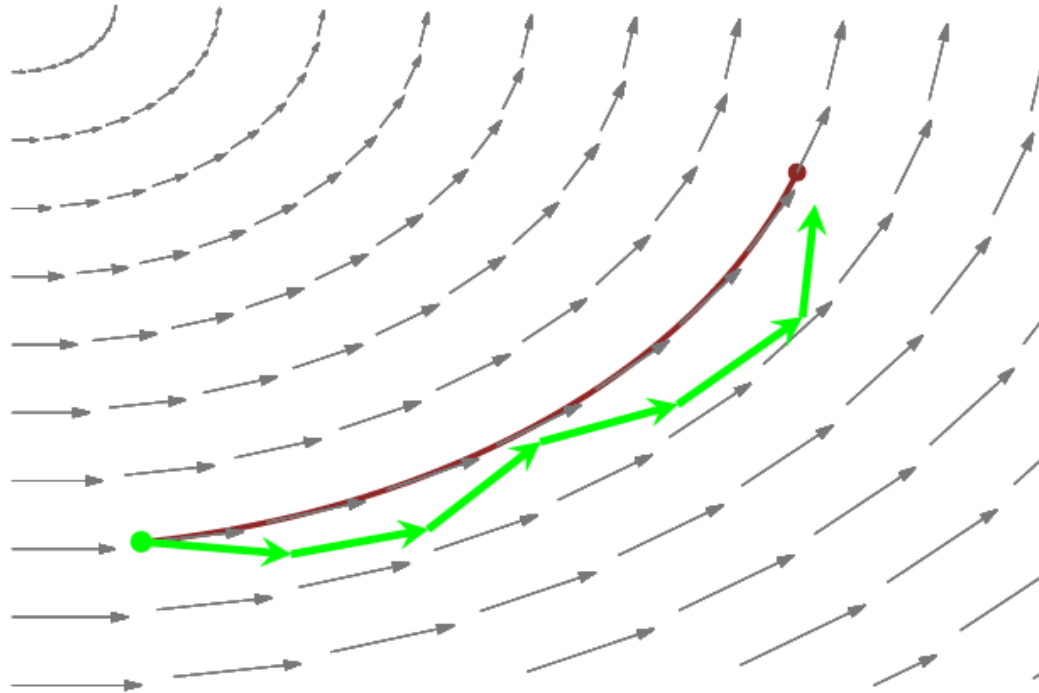
Figure 3: A trajectory for a 2D Gaussian distribution, simulated using 25 leapfrog steps with a stepsize of 0.25. The ellipses plotted are one standard deviation from the means. The initial state had $q = [-1.50, -1.55]^T$ and $p = [-1, 1]^T$.

(Neal 2013)

Hamiltonian does not change during leapfrog steps.

(But position and momentum changes.)

Approximation of the exact Hamiltonian trajectory



We correct for the bias introduced by the leapfrog integrator by using a Metropolis-Hastings scheme.

Hamiltonian MH algorithm

Start by $q = q_0$, $p = p_0$, $b = 0$,

Iterate the following:

- Set $q = q^b$.
- Propose new momentum, $p \mid q \sim \exp(-K(p, q))$.
- Run leapfrog Hamiltonian dynamics for T/ϵ steps to get proposal (q^*, p^*) .
- Calculate acceptance probability $\alpha = \min \left[1, \frac{\pi(q^*)}{\pi(q^b)} \right]$
- Accept $q^{b+1} = q^*$ if $U < \alpha$, else set $q^{b+1} = q^b$.
- Set $b = b + 1$.

(Proposal cancels because of symmetry in leapfrog dynamics.)

Choice of kinetic energy

Which conditional probability distribution over the momentum do we choose?

Usually a Gaussian distribution is chosen, for example using a Riemannian structure

$$\pi(p \mid q) = N(p \mid 0, \Sigma(q))$$

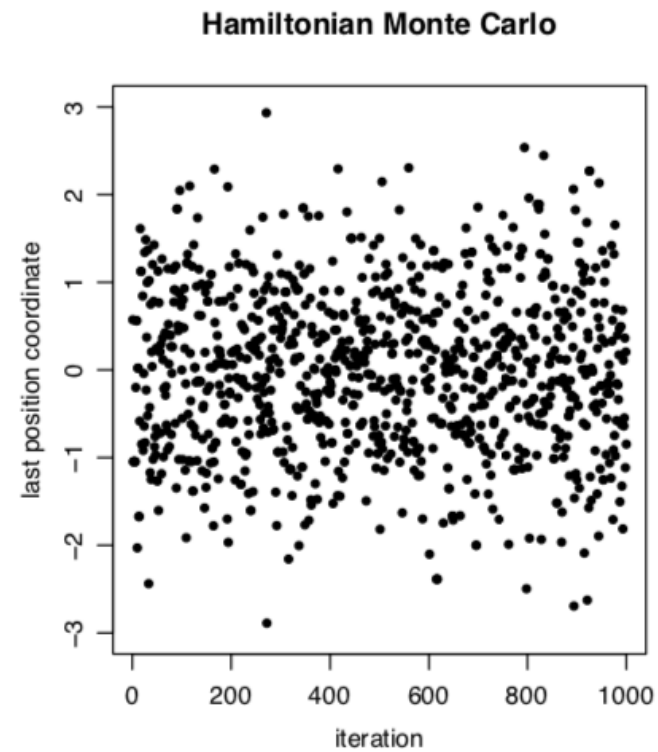
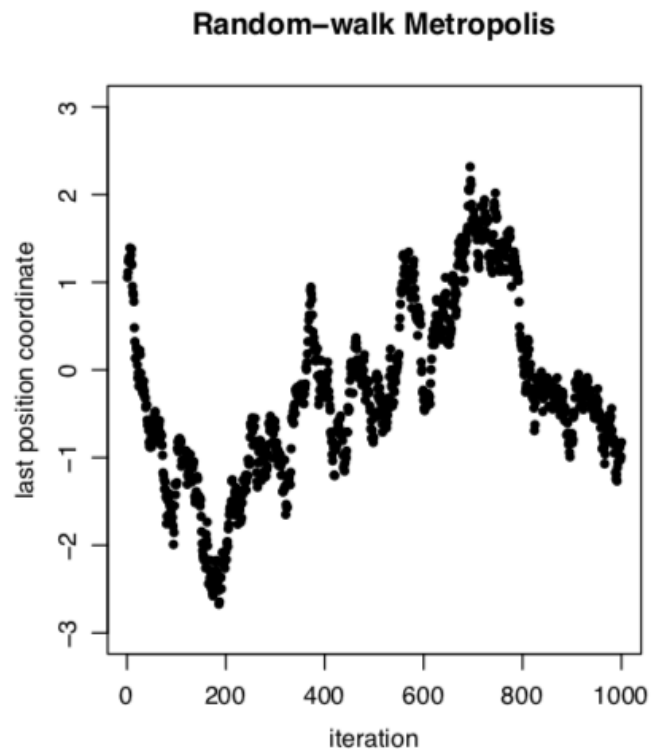
so that

$$K(p, q) = \frac{1}{2} \log |\Sigma(q)| + \frac{1}{2} p^\top \Sigma(q)^{-1} p + \text{const.}$$

leading to Riemannian Hamiltonian Monte Carlo.

Often $\pi(p \mid q) = N(p \mid 0, I)$ is used.

Illustration mixing



(From Neal (2012) <https://arxiv.org/pdf/1206.1901.pdf>)

Pros and cons of Hamiltonian MC

- Makes large jumps in the target density. Good mixing.
- Natural proposal - motivated by differential equations and physics
- Code exists (NUTS, Stan) for efficient implementation.
- Depends on time required to get derivatives.
- Not obvious how to tune (improve) proposal for momentum.
- Still struggles with difficult targets (multimodal densities, ridge densities).

Effective sample size is usually large for Hamiltonian MH. (Could scale this with evaluation cost per iteration, which is larger when derivatives are required.)

A nice illustration of various MCMC schemes:

<http://chi-feng.github.io/mcmc-demo/app.html>

Introduction to Stan



- Stan (mc-stan.org) is a probabilistic programming language implementing [full Bayesian statistical inference with MCMC sampling](#).
- It is named in honour of [Stanislaw Ulam](#), pioneer of the Monte Carlo method, and was developed by a team of 34 members.
- The MCMC algorithm that is implemented is a variant of Hamiltonian Monte Carlo, the [No-U-Turn Sampler \(NUTS\)](#).

No-U-turn Sampler (NUTS)

We have learnt that tuning the Hamiltonian Monte Carlo algorithm is tricky. One solution is NUTS, the default sampler of Stan, which automatically adapts the number of steps and step size in the leapfrog algorithm, eliminating the need of user-specified tuning

NUTS uses a recursive algorithm to build a set of likely candidate points that spans a wide swath of the target distribution, stopping automatically when it starts to double back and retrace its steps.

Automatic Differentiation

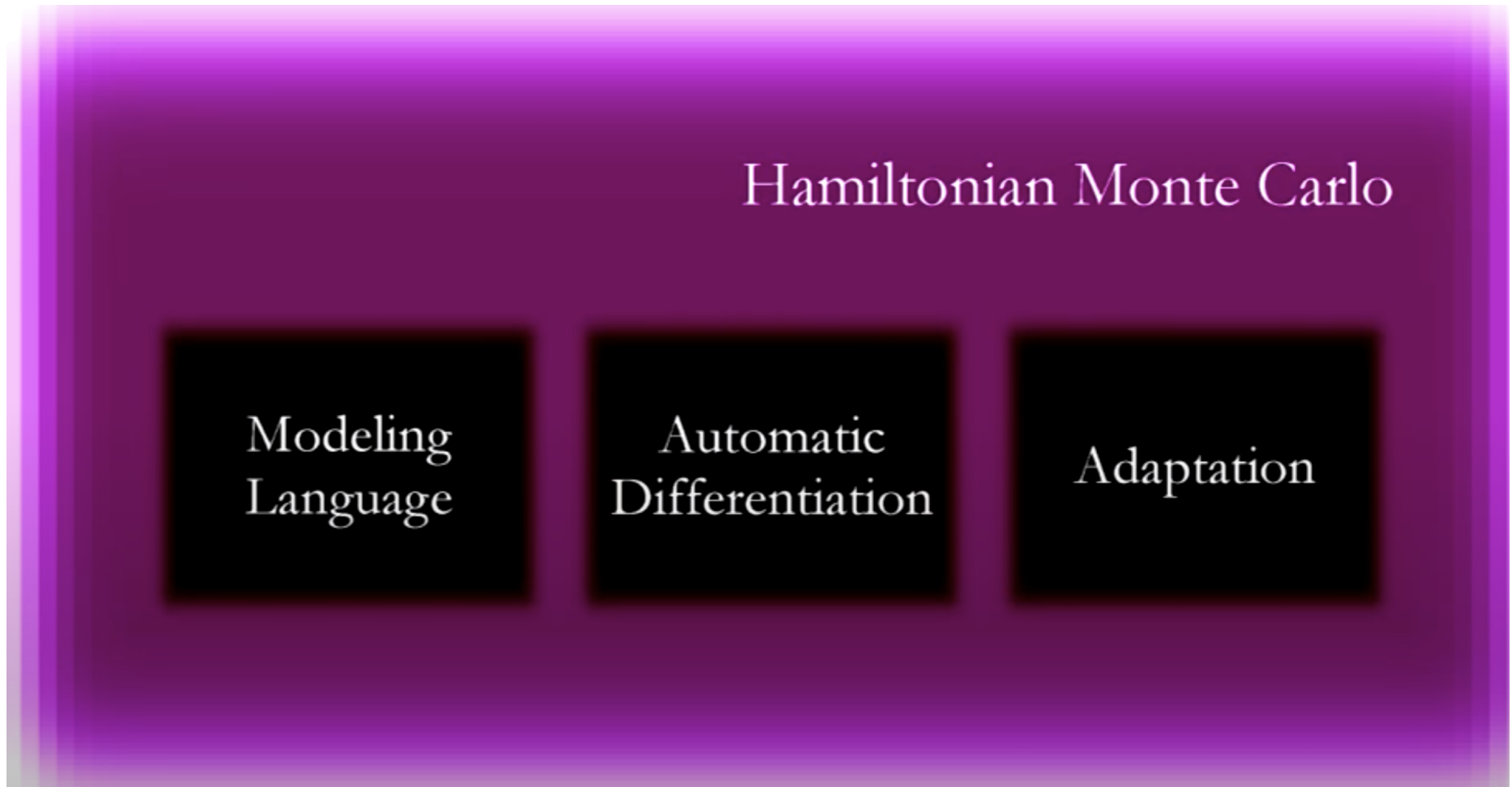
Stan implements reverse-mode [automatic differentiation](#) to calculate gradients as need in NUTS:

$$\left(\frac{f}{x_1}(x_1, \dots, x_N), \dots, \frac{f}{x_N}(x_1, \dots, x_N) \right)$$

Automatic differentiation computes these values automatically, using only a representation of f as a computer program.

For details see: Carpenter, Bob, et al. The Stan math library: Reverse-mode automatic differentiation in C++. arXiv preprint arXiv:1509.07164, 2015.

Summary



Why Stan?

There are many “black-box” MCMC sampler (BUGS, JAGS, PyMC, etc)

- Stan is open source
- Built to be fast
- “Stan can handle problems that choke BUGS and JAGS” – Andrew Gelman

Platforms and Interfaces

Platforms:

- Linux, Mac OS X, Windows

Interfaces:

- CmdStan - command-line executable for the shell
- RStan - Integration with R
- PyStan - Integration with Python
- MatlabStan - Integration with Matlab
- Stan.jl - Integration with Julia
- StataStan - Integration with Stata

Installation

<https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started>

I have installed RStan on my Mac using:

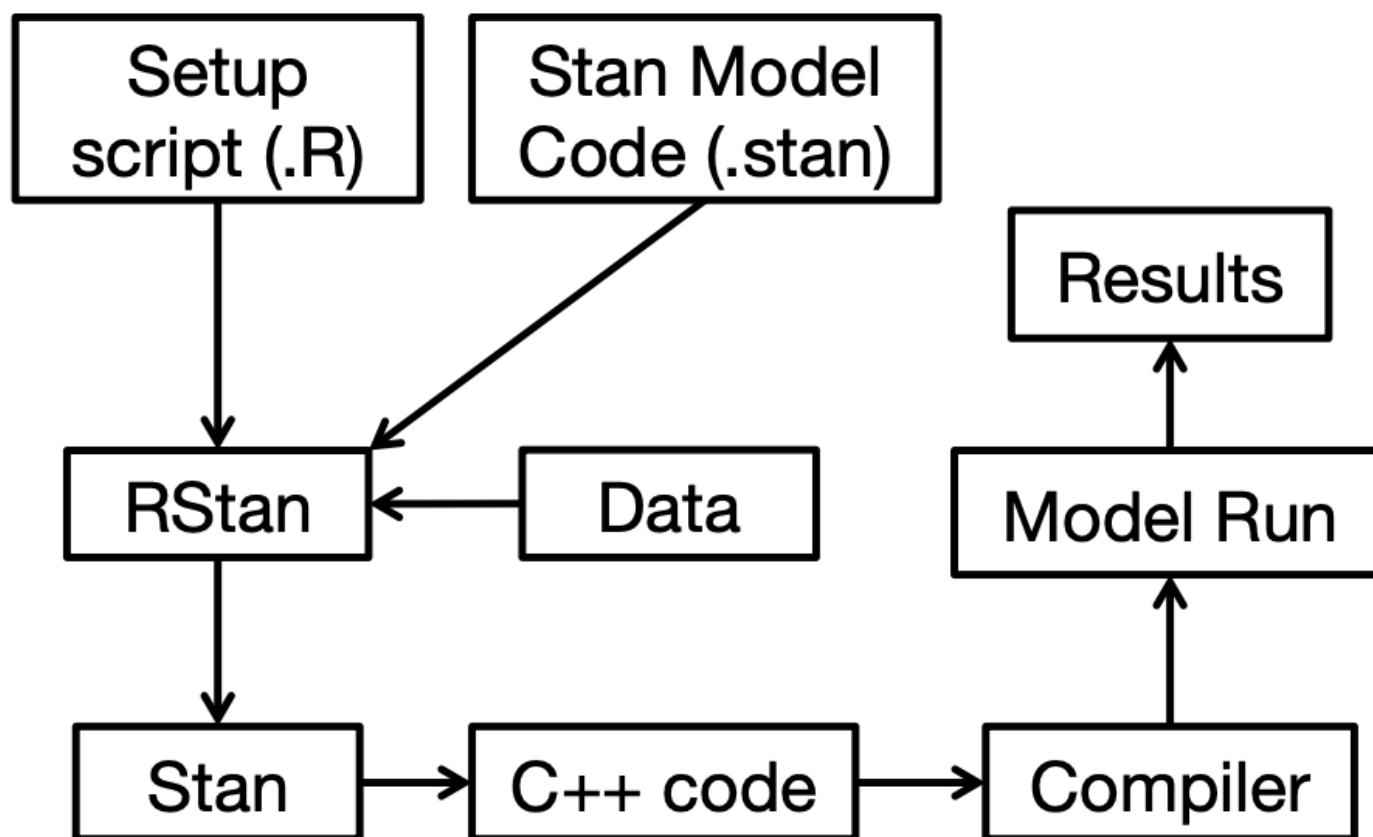
```
# Compile packages using all cores
Sys.setenv(MAKEFLAGS = paste0("-j",parallel::detectCores()))
install.packages(c("StanHeaders","rstan"),type="source")
```

and then used

```
library("rstan") # observe startup messages
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = TRUE)
```

Using Stan

We will use the R interface called RStan



Getting started: The eight schools data set

Example from Section 5.5 in Gelman et al. (2003)⁵

Goal: Investigates effectiveness of classes that were designed to improve students' performance on college admission tests

School	Estimated effect of coaching (y_j)	Standard error of effect (σ_j)
A	28	15
B	8	10
C	-3	16
D	7	11
E	-1	9
F	1	11
G	18	10
H	12	18

⁵Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (2003). Bayesian Data Analysis, CRC Press, London, 2nd Edition.

The eight schools data set

Model:

$$y_j \sim \text{Normal}(\theta_j, \sigma_j) \quad j = 1, \dots, 8$$

$$\theta_j \sim \text{Normal}(\mu, \tau) \quad j = 1, \dots, 8$$

$$p(\mu, \tau) \propto 1$$

schools.stan

```
// saved as schools.stan
data {
  int<lower=0> J;          // number of schools
  real y[J];              // estimated treatment effects
  real<lower=0> sigma[J]; // standard error of effect estimates
}
parameters {
  real mu;                // population treatment effect
  real<lower=0> tau;       // standard deviation in treatment effects
  vector[J] eta;          // unscaled deviation from mu by school
}
transformed parameters {
  vector[J] theta = mu + tau * eta; // school treatment effects
}
model {
  target += normal_lpdf(eta | 0, 1); // prior log-density
  target += normal_lpdf(y | theta, sigma); // log-likelihood
}
```

Challenges regarding parameterisations are discussed here:

https://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html

Conventions in Stan

- Comments are indicated by double slashes, “//”
- Stan uses semi-colon (i.e., ;) to mark end of line or command,
- Order matters!

Now let us look how we fit this model in R :-)

Program Blocks used in .stan file

- **data** (once)
declare data types, sizes etc. It will read external information.
- **parameters** (per leapfrog step) declare parameter types, size etc.
Here we define the sampling space.
- **model** (per leapfrog step) define the model/posterior
- transformed data (once after data)
declare and define transformed data variables
- transformed parameters (per leapfrog step) declare and define transformed parameter vars
- generated quantities (per sample) declare and define generated quantity variables

An additional "functions" block (compiled with model) can be used for user-defined functions.

Basic variable types

- Primitive: int, real
- Matrix: matrix[M,N], vector[M], row_vector[N]
- Bounded: primitive or matrix, with $\langle \text{lower}=\text{L} \rangle$, $\langle \text{upper}=\text{U} \rangle$,
 $\langle \text{lower}=\text{L}, \text{upper}=\text{U} \rangle$

Which functions do exist in RStan?

Stan has versions of many of the most useful R functions for statistical modeling, including probability distributions, matrix operations, and various special functions. However, the names of the Stan functions and its parameterisation may differ from their R counterparts. Here, the `lookup` function is handy:

```
lookup("dnorm")
```

A functions reference can be found here:

https://mc-stan.org/docs/2_26/functions-reference

This example is very basic ...

A variety of models is discussed in the Stan Users' guide

https://mc-stan.org/docs/2_26/stan-users-guide/index.html

For the discussion of various case studies see:

<https://mc-stan.org/users/documentation/case-studies>

Examples include:

- Spatial models
- Modelling disease transmission
- Hidden Markov Models
- Population dynamics

Also of interest: rstanarm

rstanarm is an R package that emulates other R model-fitting functions but uses Stan (via the rstan package) for the back-end estimation. The primary target audience is people who would be open to Bayesian inference if using Bayesian software were easier but would use frequentist software otherwise.

see <https://mc-stan.org/rstanarm/>