Markov Chain Monte Carlo III

PSYC 573

University of Southern California
February 24, 2022

Hamiltonian Monte Carlo (HMC)

From Hamiltonian mechanics

- Use *gradients* to generate better proposal values

Results:

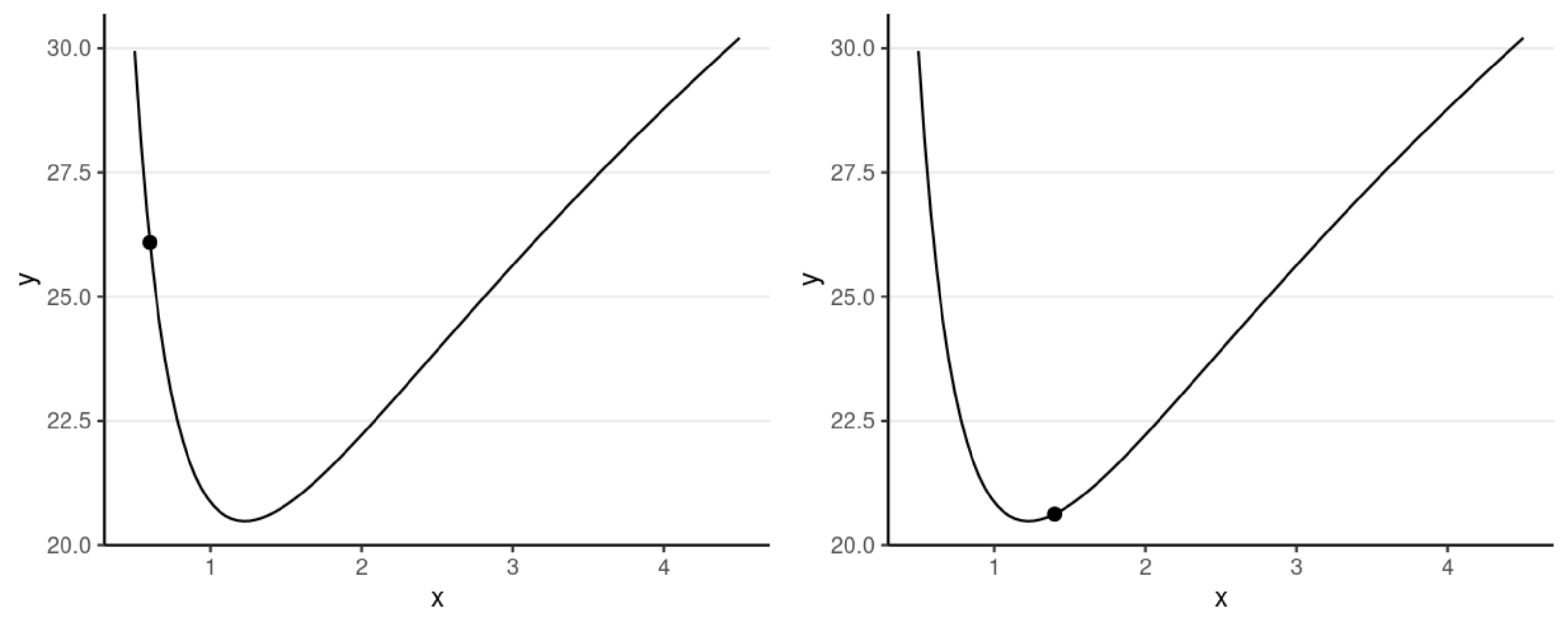
- Higher acceptance rate
- Less autocorrelation/higher ESS
- Better suited for high dimensional problems

Gradients of Log Density

Consider just σ^2

Potential energy = $-\log P(\theta)$

> Which one has a higher **potential energy**?



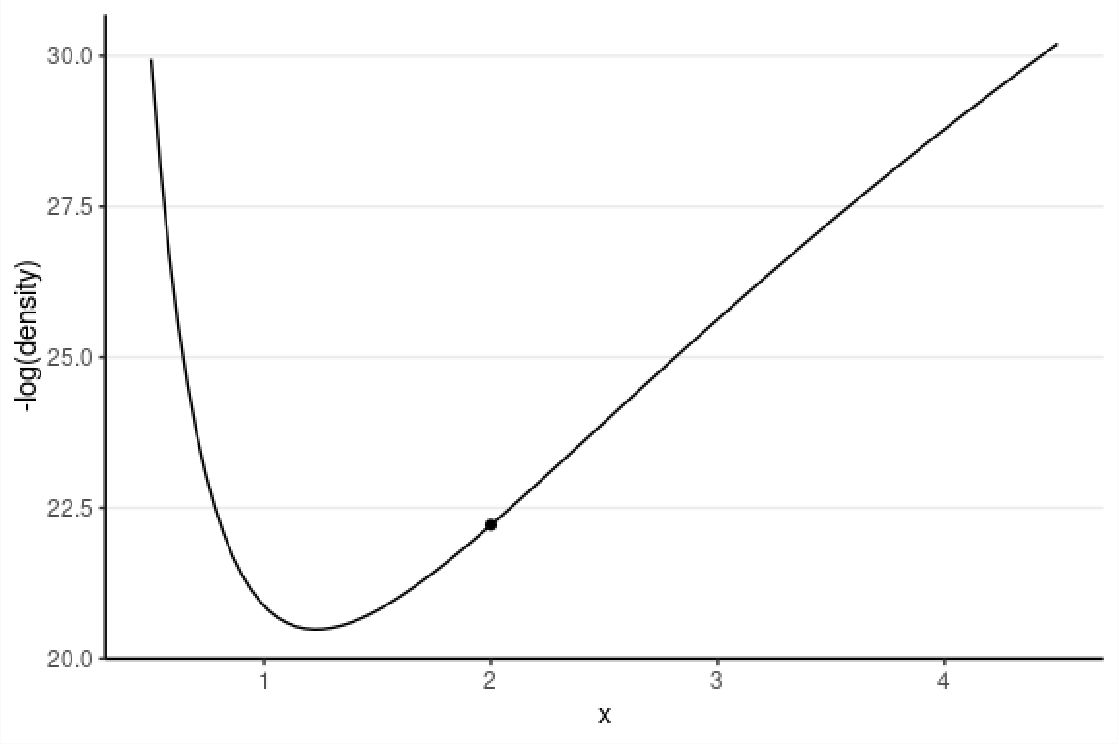
HMC Algorithm

Imagine a marble on a surface like the log posterior

1. Simulate a random *momentum* (usually from a normal distribution)
2. Apply the momentum to the marble to roll on the surface
3. Treat the position of the marble after some time as the *proposed value*
4. Accept the new position based on the Metropolis step
    - i.e., probabilistically using the posterior density ratio

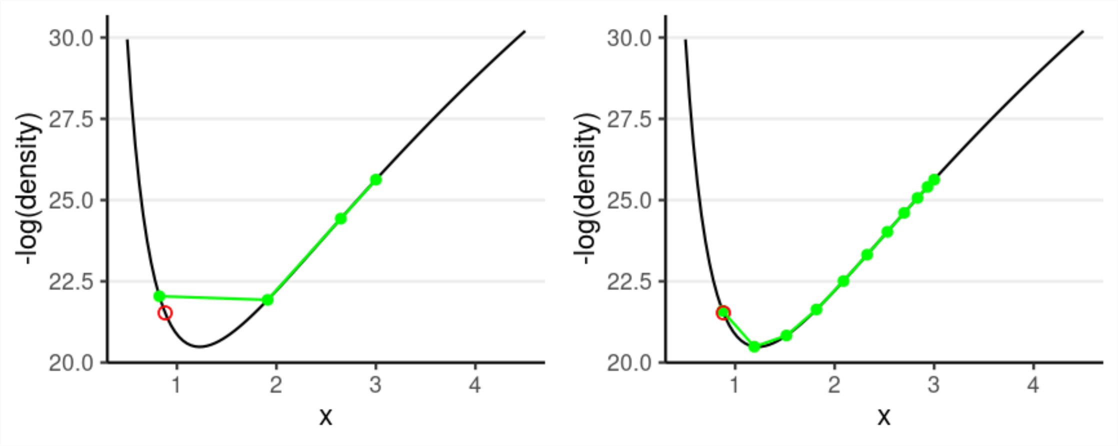
Leapfrog Integrator

Location and velocity constantly change



Leapfrog integrator

- Solve for the new location using L leapfrog steps
- Larger L, more accurate location
- Higher curvature requires larger L and smaller *step size*



Divergent transition: When the leapfrog approximation deviates substantially from where it should be

No-U-Turn Sampler (NUTS)

Algorithm used in STAN

Two problems of HMC

- Need fine-tuning L and **step size**
- Wasted steps when the marble makes a U-turn

NUTS uses a binary search tree to determine L and the **step size**

- The **maximum treedepth** determines how far it searches

See a demo here: https://chi-feng.github.io/mcmc-demo/app.html

Stan

Stan

A language for doing MCMC sampling (and other related methods, such as maximum likelihood estimation)

Current version (2.29): mainly uses NUTS

It supports a wide range of distributions and prior distributions

Written in C++ (faster than R)

Consider the example

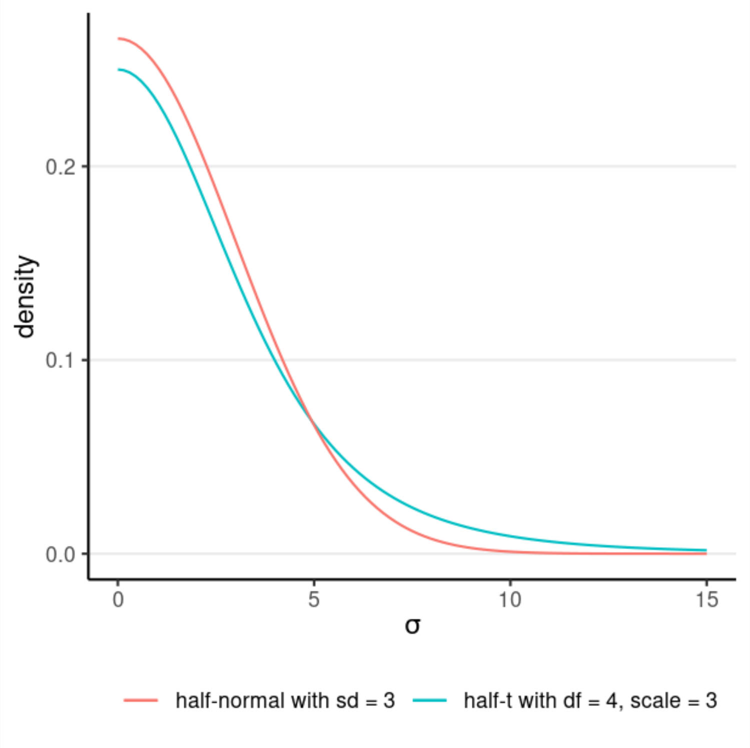
Model:

$$\mathbf{wc\_laptop}_i \sim N(\mu, \sigma)$$

Prior:

$$\mu \sim N(5, 10)$$
$$\sigma \sim t_4^+(0, 3)$$

$t_4^+(0, 3)$ is a half-t distribution with df = 4 and scale = 3



An example STAN model

```
data {
  int<lower=0> N;  // number of observations
  vector[N] y;  // data vector y
}
parameters {
  real mu;  // mean parameter
  real<lower=0> sigma;  // non-negative SD parameter
}
model {
  // model
  y ~ normal(mu, sigma);  // use vectorization
  // prior
  mu ~ normal(5, 10);
  sigma ~ student_t(4, 0, 3);
}
generated quantities {
  vector[N] y_rep;  // place holder
  for (n in 1:N)
    y_rep[n] = normal_rng(mu, sigma);
}
```

Components of a STAN Model

- `data`: Usually a list of different types
  - `int`, `real`, `matrix`, `vector`, `array`
    can set lower/upper bounds
- `parameters`
- `transformed parameters`: optional variables that are transformation of the model parameters
- `model`: definition of **priors** and the **likelihood**
- `generated quantities`: new quantities from the model (e.g., simulated data)

RStan

https://mc-stan.org/users/interfaces/rstan

An interface to call Stan from R, and import results from STAN to R

Call `rstan`

R code | Output

```r
library(rstan)
rstan_options(auto_write = TRUE)  # save compiled STAN object
nt_dat <- haven::read_sav("https://osf.io/qrs5y/download")
wc_laptop <- nt_dat$wordcount[nt_dat$condition == 0] / 100
# Data: a list with names matching the Stan program
nt_list <- list(
  N = length(wc_laptop),  # number of observations
  y = wc_laptop  # outcome variable (yellow card)
)
# Call Stan
norm_prior <- stan(
    file = "stan/normal_model.stan",
    data = nt_list,
    seed = 1234  # for reproducibility
)
```