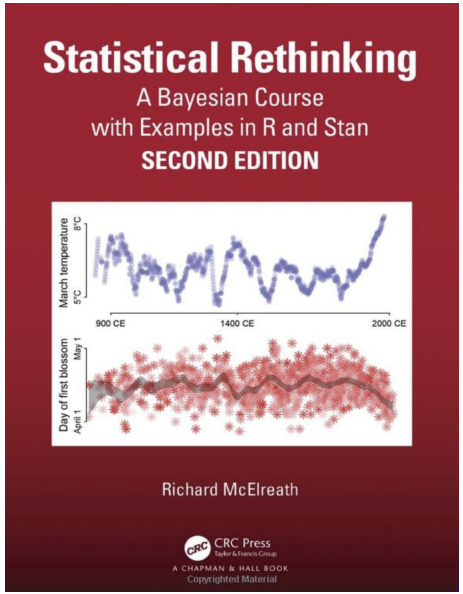
Introduction to Data Science

MCMC

Prof. Dr. Ralf Lämmel & **M.Sc. Johannes Härtel**
(johanneshaertel@uni-koblenz.de)



[McElreath20]

The major source for this lecture.

Grid approximation

Specification of a statistic model and data
(comparable to assignment 5)

$$D_i \sim \text{Normal}(\mu, \sigma) \qquad \text{[likelihood]}$$

$$\mu \sim \text{Uniform}(0, 1) \qquad \text{[μ prior]}$$
$$\sigma \sim \text{Uniform}(0, 1) \qquad \text{[σ prior]}$$

$$D = [0.312, 0.555, 0.24, 0.979]$$

The **posterior**, given in terms of priors and likelihood, as a function of parameter mu and sigma

`dunif` and `dnorm` are probability density functions.

$$dunif(sigma, 0, 1) \cdot dunif(mu, 0, 1) \cdot \prod_{y \in D} dnorm(y, mu, sigma)$$

[σ prior]

[μ prior]

[likelihood]

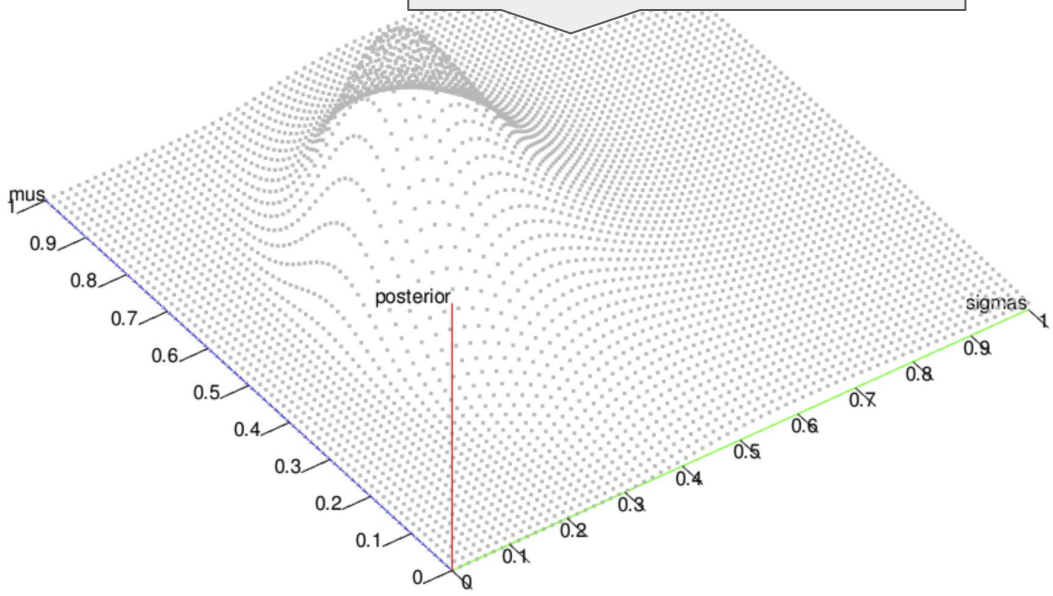
Grid approximation of the posterior

regarding parameter sigma σ and mu μ

- An approximation of the posterior can be implemented in terms of an **N-dimensional loop** over all combinations of parameters (sigma σ and mu μ).

See details in the reference solution to assignment 5.

Every point is the results of invoking the posterior function with two particular parameters.



$$dunif(sigma, 0, 1) \cdot dunif(mu, 0, 1) \cdot \prod_{y \in D} dnorm(y, mu, sigma)$$

The corresponding **log-posterior**

Sometimes it is better to work with the **log of the function for the posterior**, especially when we need its **gradient**, or to avoid **numerical errors**.

$$log(dunif(sigma, 0, 1)) + log(dunif(mu, 0, 1)) + \sum_{y \in D} log(dnorm(y, mu, sigma))$$

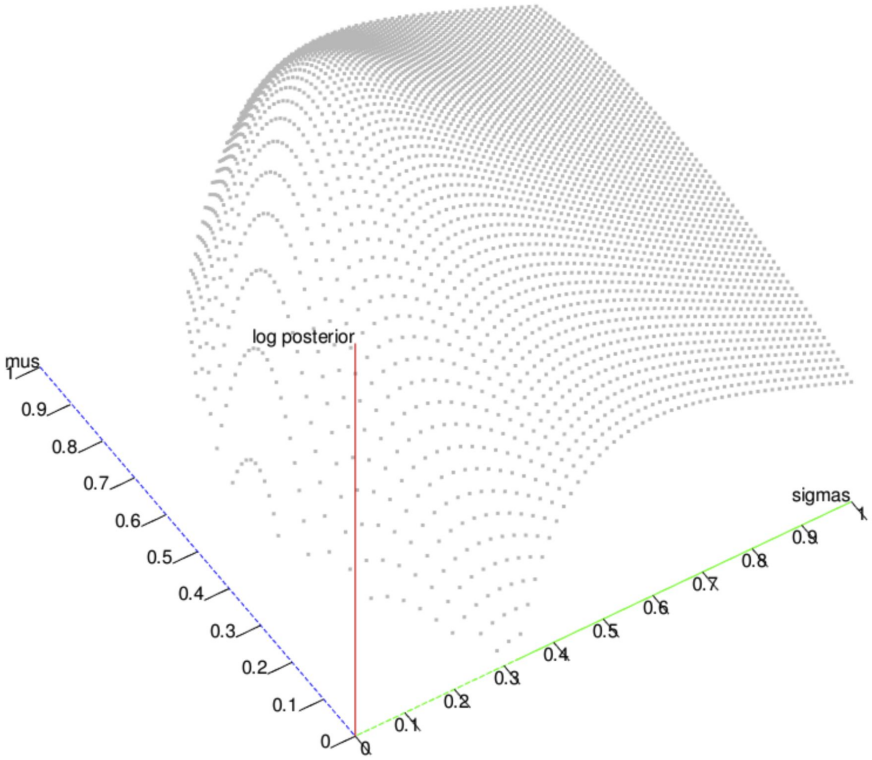
[log σ prior]

[log μ prior]

[log likelihood]

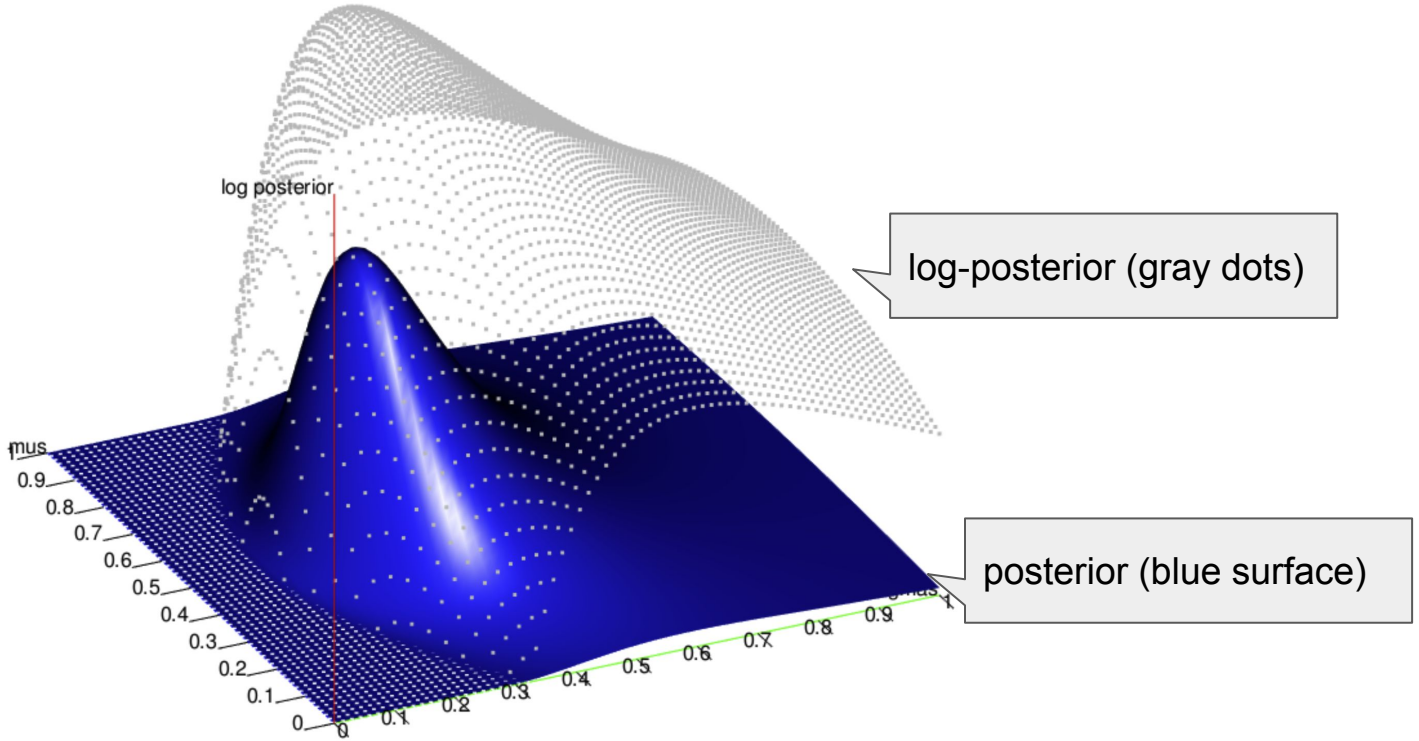
Grid approximation of the **log-posterior**

- The grid approximation of the log-posterior is comparable to the posterior.
- The grid approximation does not need the gradient.
- Computing the log posterior helps to **avoid numerical errors**.



Comparing **posterior and log-posterior**

We can flexibly switch: log-posterior = log(posterior), posterior = $e^{\text{log-posterior}}$



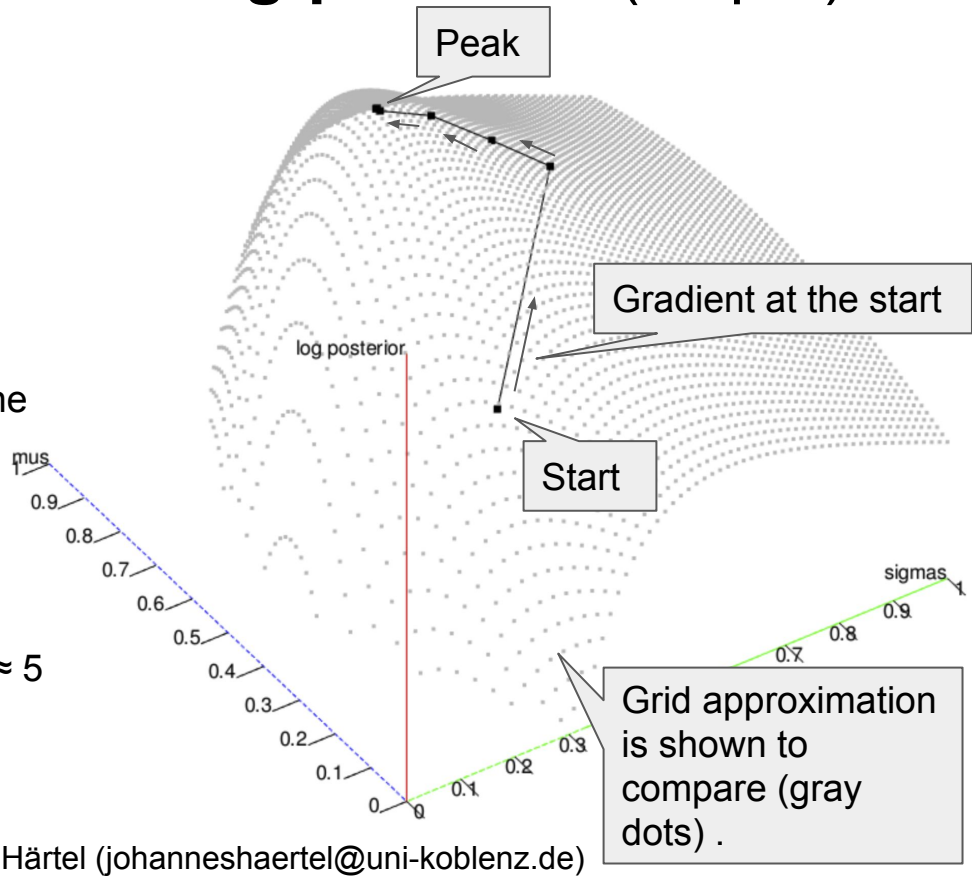
Benefits and Problems: Grid approximation.

- Benefits:
  - **Accurate** estimate of the (log-) posterior.
  - **Easy** to implement on your own.
- Problems:
  - Problematic **definition of start, end, and step size** for grid.
  - **Impossible** if number of **parameters increases** (exponential growth of the grid points that need to be examined with increasing parameters).

Quadratic approximation

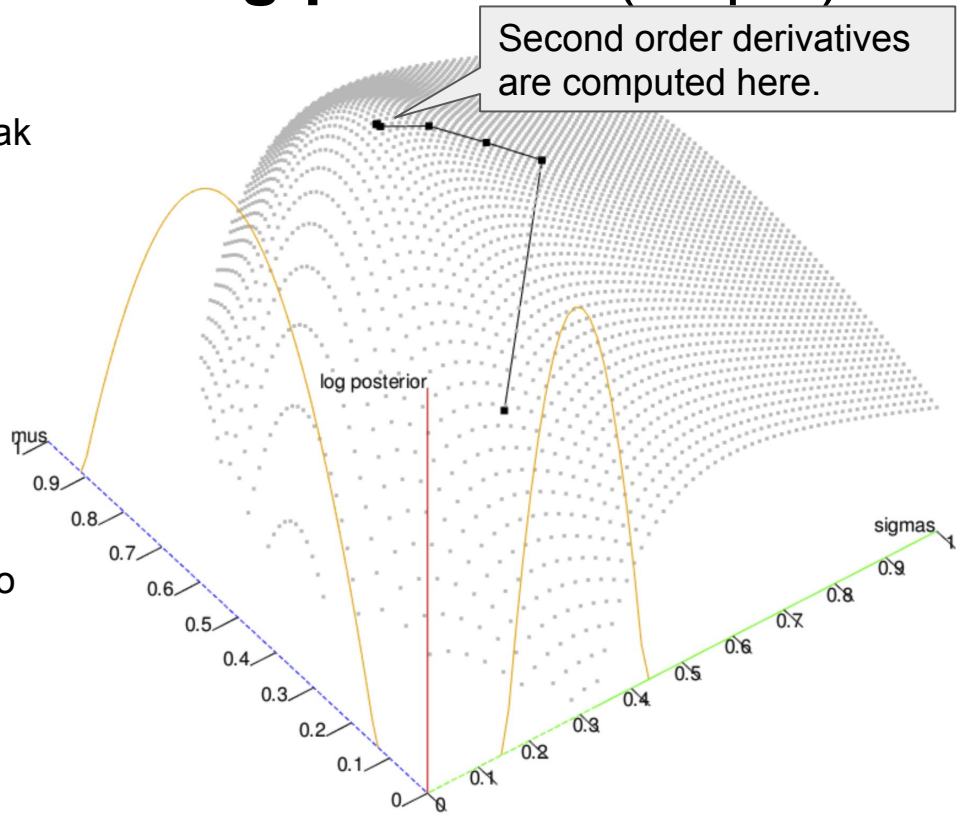
A **quadratic approximation** of the **log-posterior** (step 1)

- We start at some point and use the gradient of the surface to find the **peak of the log-posterior**.
- The gradient of the **log-posterior surface** is the **log-posterior derived** with respect to mu and sigma. There is an **analytical solution** to this.
- This computation is efficient (we just examine ≈ 5 points, compared to a grid with N×M points)

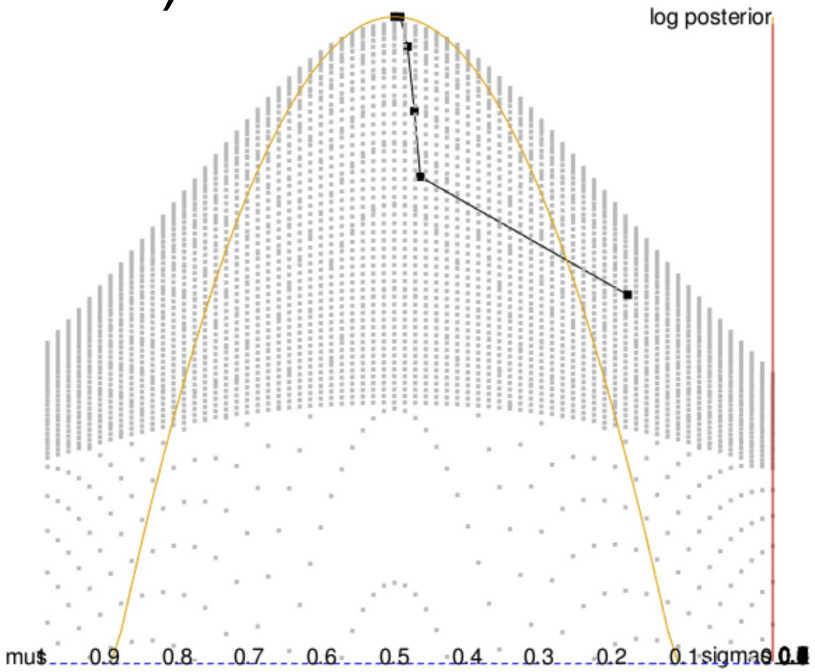


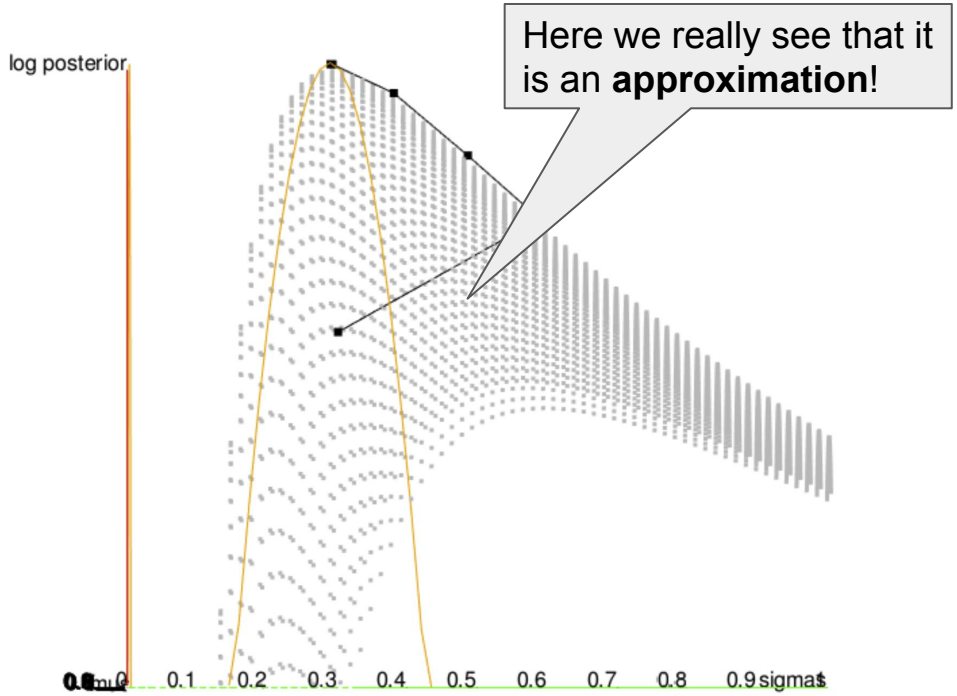
A **quadratic approximation** of the **log-posterior** (step 2)

- We use the **second order derivates** at the peak (the curvature of the log-posterior) to construct two parabolas (quadratic functions).
- These two parabolas **approximate** the log posterior to some extends (see next slide).
- If you use the rethinking package, you may also try QUAP call (instead of ULAM) doing this job for you. It is much faster but an approximation.



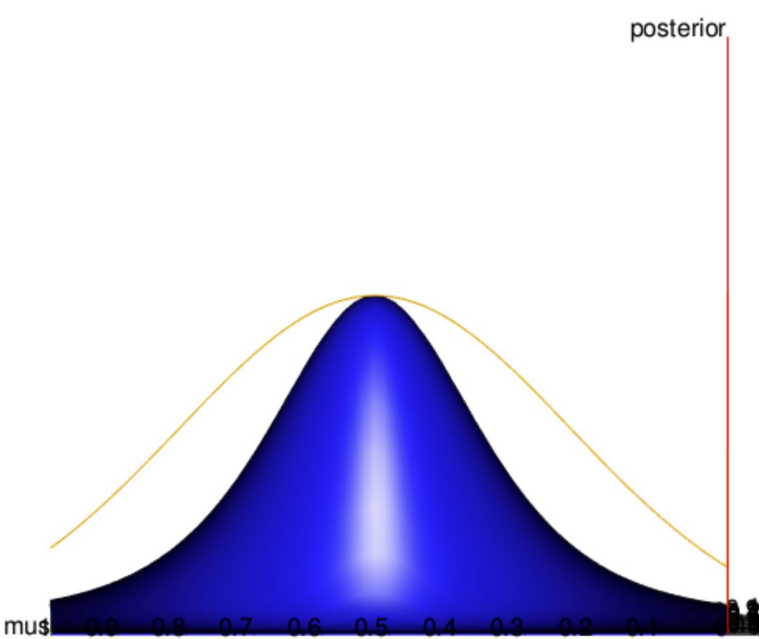
A **quadratic approximation** of the **log-posterior** (detailed view)

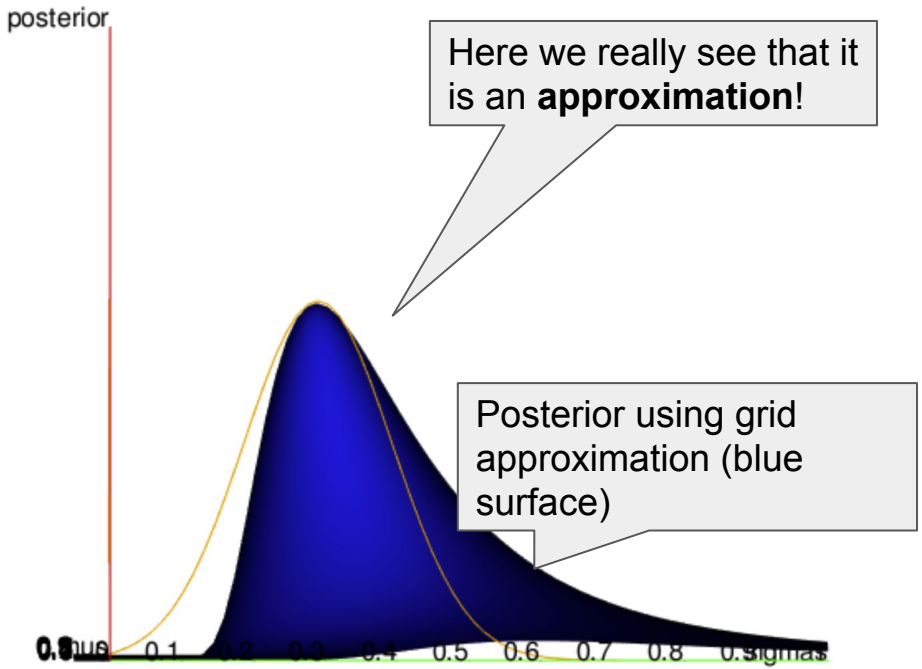




A **quadratic approximation** of the **posterior**

When **mapping** the approximated log-posterior **back** to the posterior (orange).





Benefits and Problems: Quadratic approximation

- Benefits:
  - It is **fast**.
  - Works for **basic linear models** quite well.
- Problems:
  - Posterior approximation is **limited** to (multivariate) normal distribution.
  - **Important models**, like the generalized linear and multilevel models, produce non-normal posteriors; hence, **quadratic approximation fails**.

Markov chain Monte Carlo (MCMC)

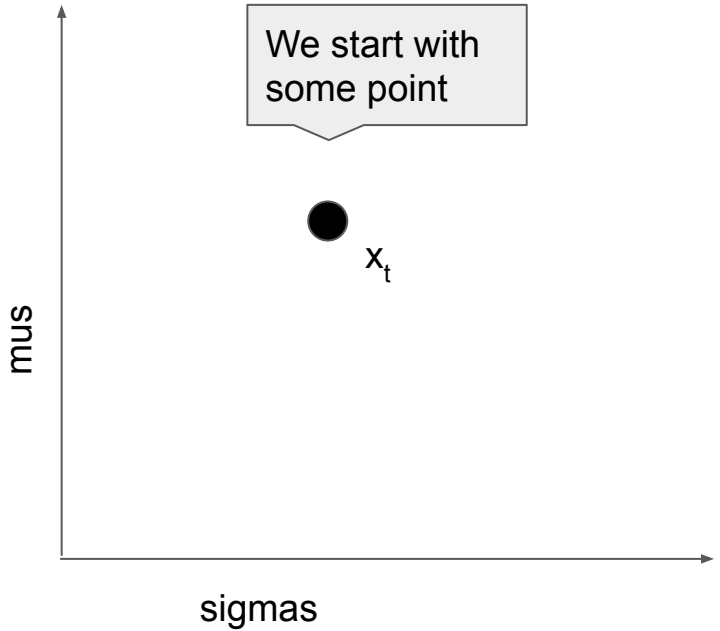
Metropolis–Hastings algorithm

Metropolis–Hastings

- Metropolis–Hastings: Draw **samples from** any **probability distribution** with probability density P(x) where we can **compute function f(x)** proportional to P(x).
- Since the **posterior** is a probability density function on the parameter space, we can use Metropolis–Hastings to sample from it.
- In the remainder of this lecture, we will see samples from the parameter space (depicted by small black dots) instead of the depicted surface of the (log-) posterior.

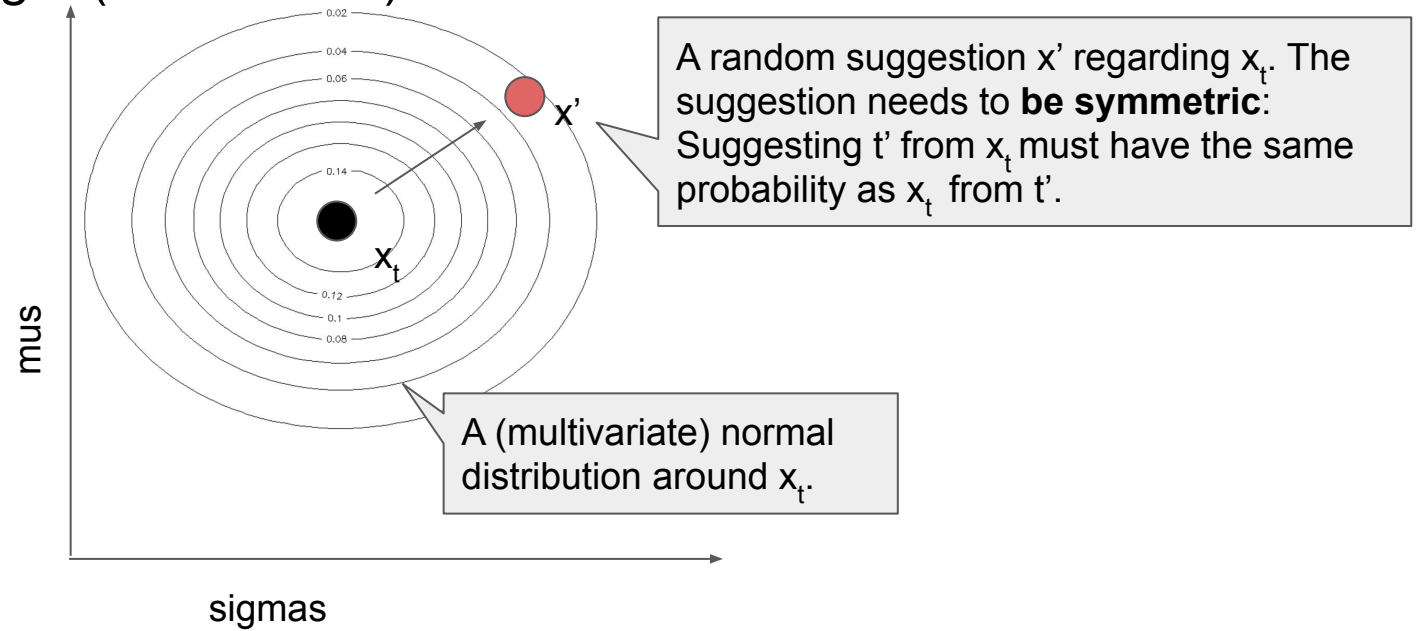
Metropolis–Hastings

Initialization at some point x_t in the parameter space.



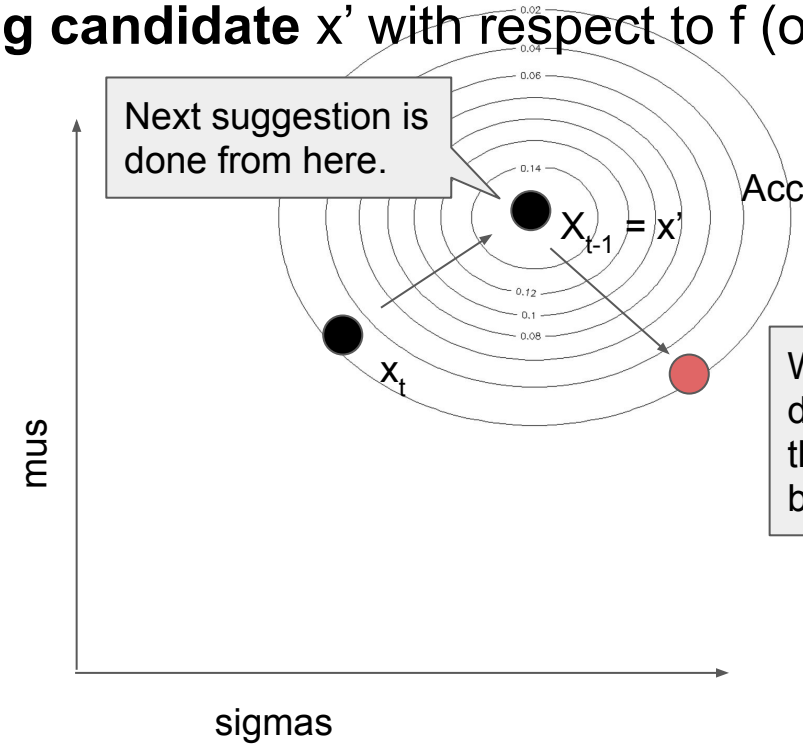
Metropolis–Hastings

Suggest new point x' regarding x_t. We typically suggest points x' around x_t following a (multivariate) normal distribution.



Metropolis–Hastings

Accepting candidate x’ with respect to f (our posterior function).

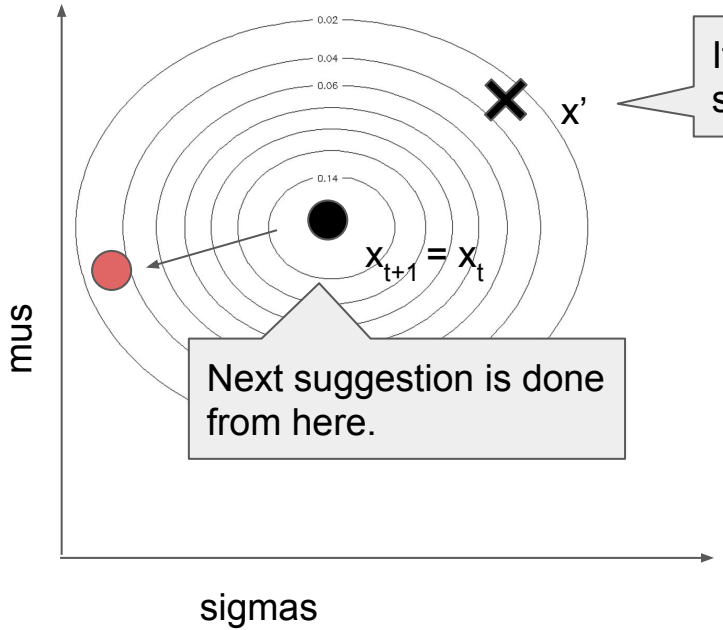


Accept if `runif(1,0,1)` ≤ `f(x')/f(`x_t`)`

We always **accept** the new point if the density f is higher at x’. If not, we accept the new point according to the ratio between new and old `f(x')/f(`x_t`)`.

Metropolis–Hastings

If we **do not accept** the candidate.



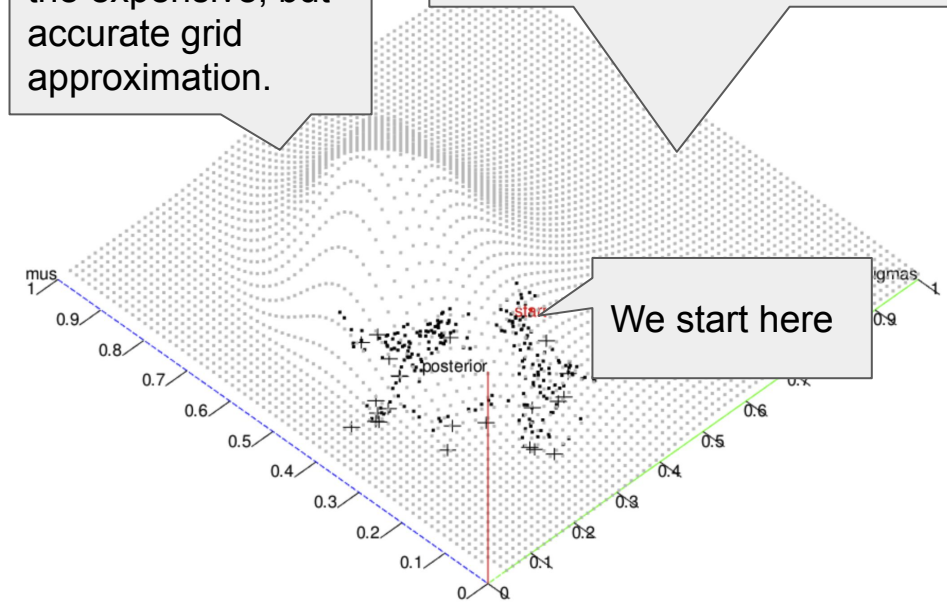
If we **do not accept**, we get rid of this suggestion and go back to continuing from x_t.

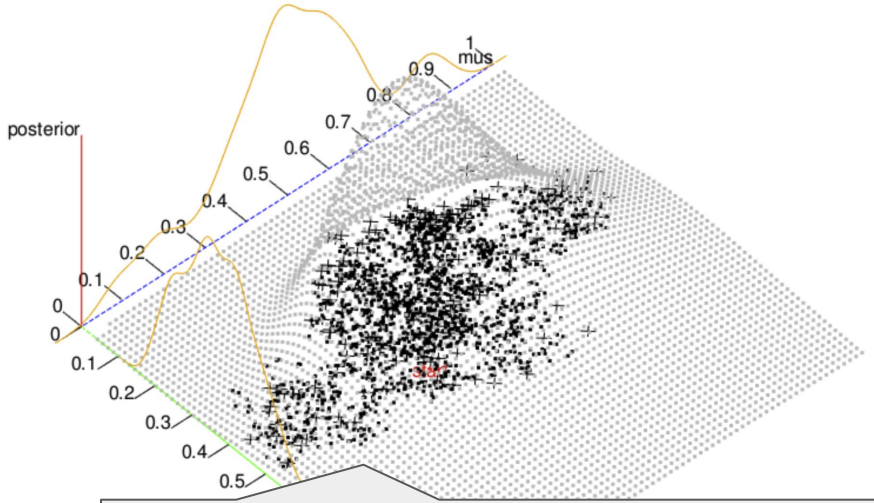
Next suggestion is done from here.

Metropolis–Hastings to run our initial model

Gray points depict the expensive, but accurate grid approximation.

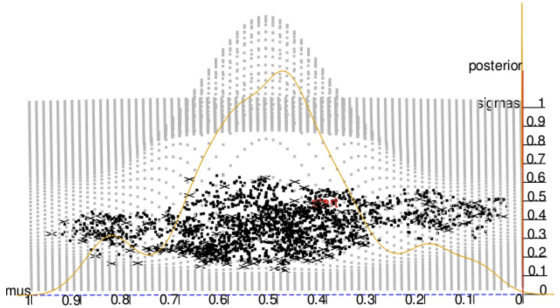
Accepted suggestions (samples) are points on the floor plane, rejected suggestions depicted as crosses.

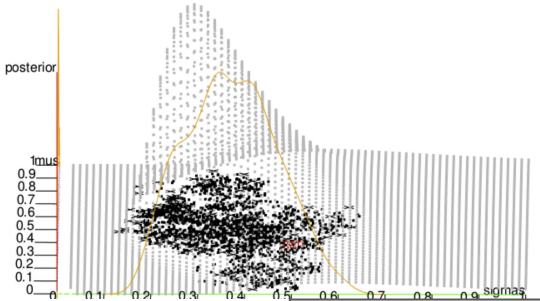


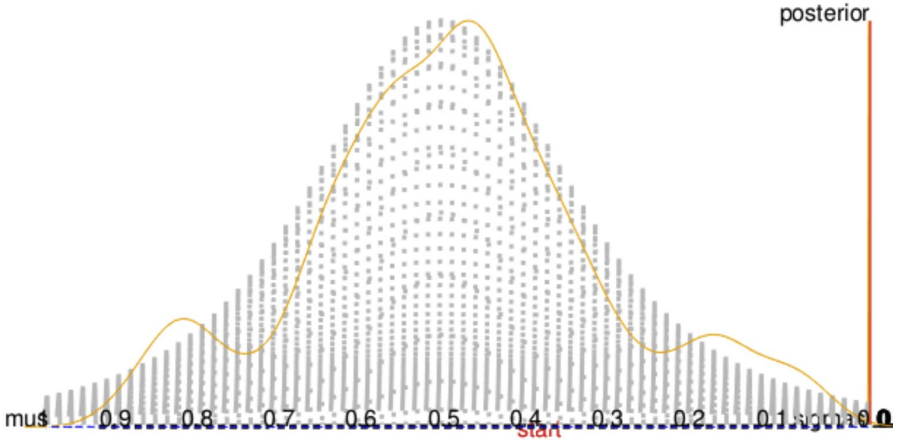


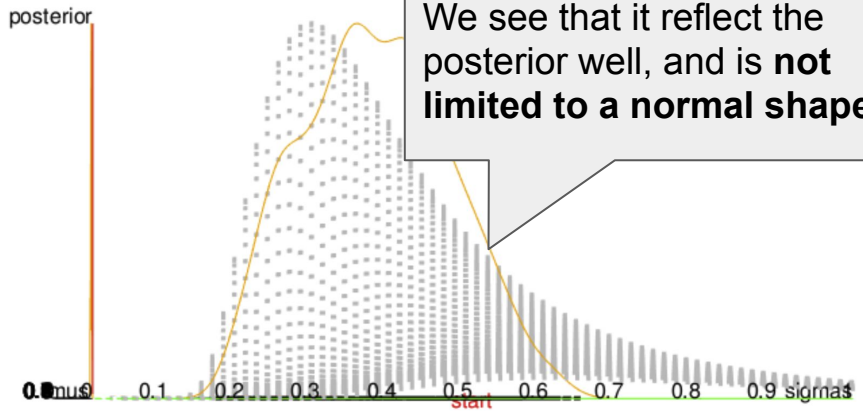
The accepted suggestions (samples) finally manages to approximate the posterior (orange shows density regarding parameters).

Metropolis–Hastings (σ and μ)









We see that it reflect the posterior well, and is **not limited to a normal shape**.

Benefits and Problems: Metropolis–Hastings

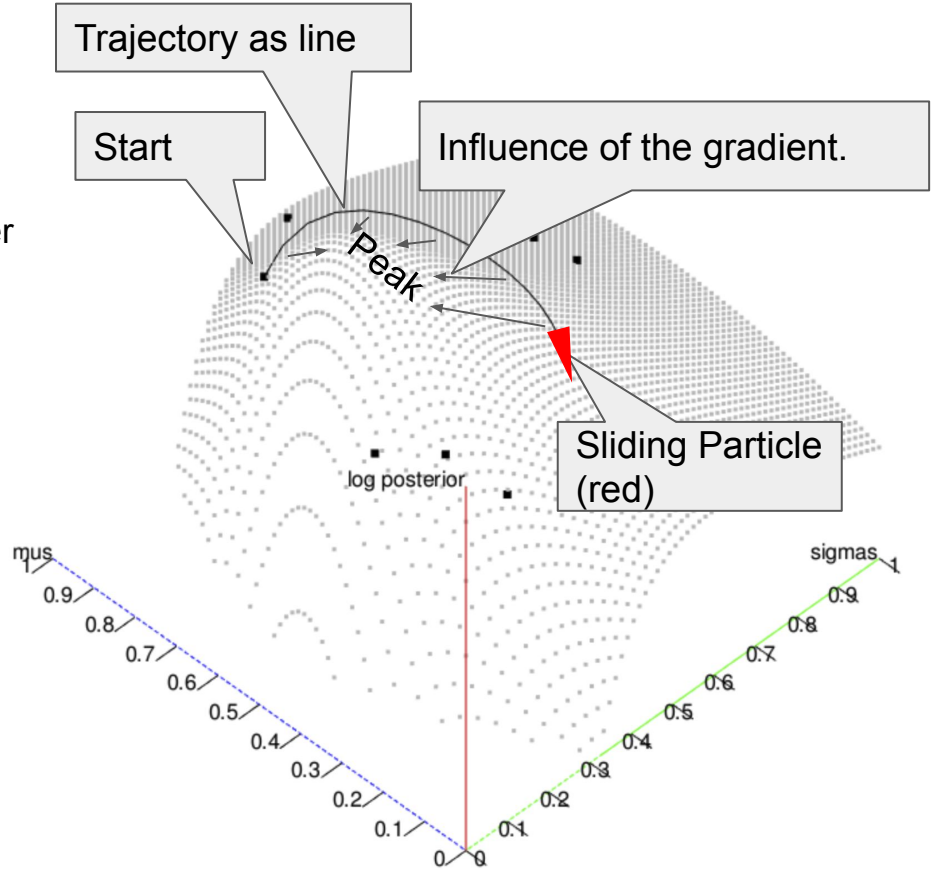
- Benefits:
  - **Not limited** to (multivariate) normal posterior.
  - **More efficient** than grid approximation when the number of parameters increases.
  - **Grandparent** of several advanced strategies.
- Problems:
  - High **correlation** between x_t and x_{t+1} (autocorrelated samples).
  - Inference **may get stuck** in regions of the posterior where parameters are correlating (see lecture on causal inference and multicollinearity).

Markov chain Monte Carlo (MCMC)

Hamiltonian Monte Carlo

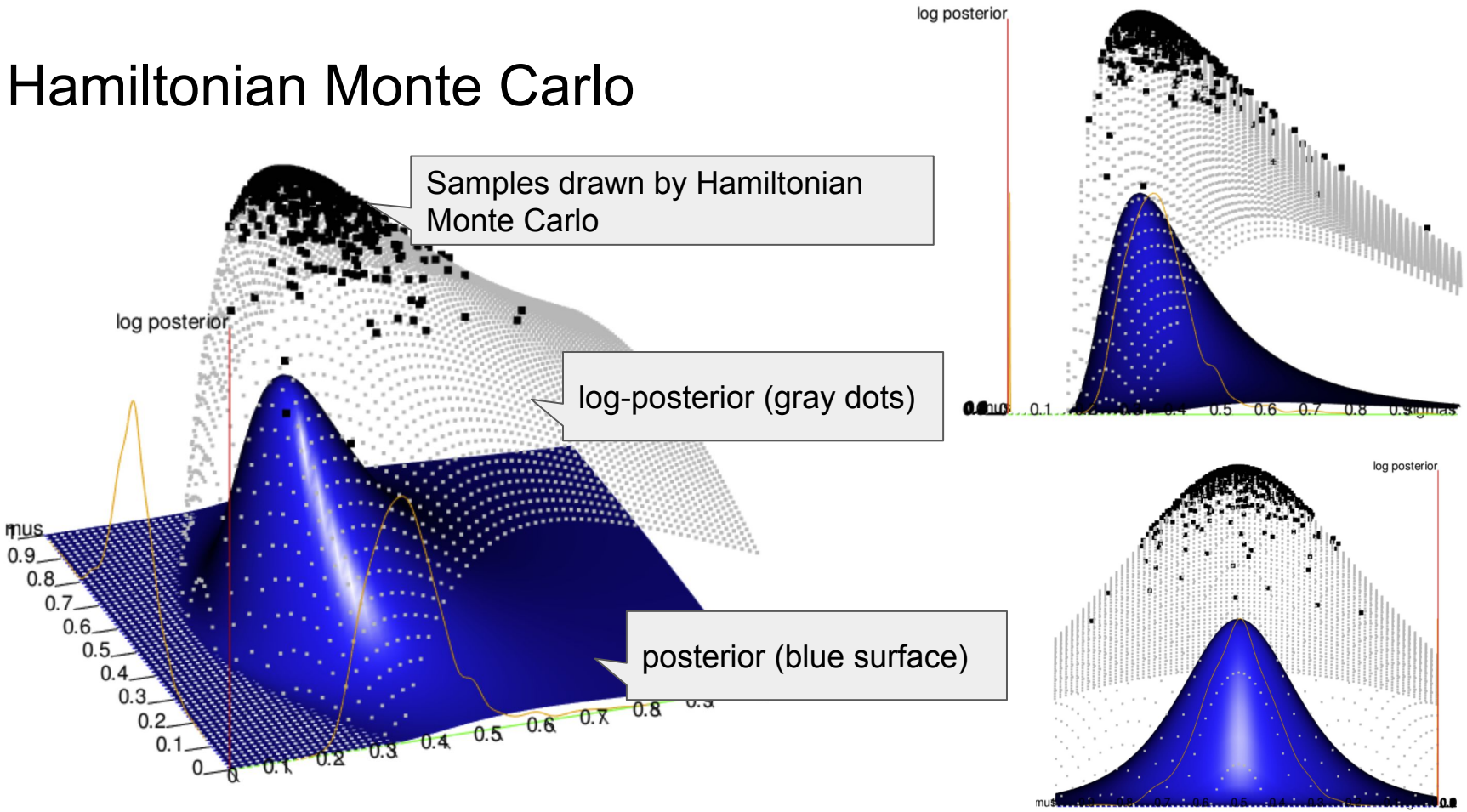
Hamiltonian Monte Carlo

- We simulate a **frictionless particle** sliding over the log posterior (like an inverted valley).
- The particle starts at some point.
- We give it a momentum by a **random flick**.
- The **gradient** influences the **trajectory**.
- After a constant time, the **particle stops**.
- **Accept/reject** (comparable to Metropolis–Hastings) and **record sample**.

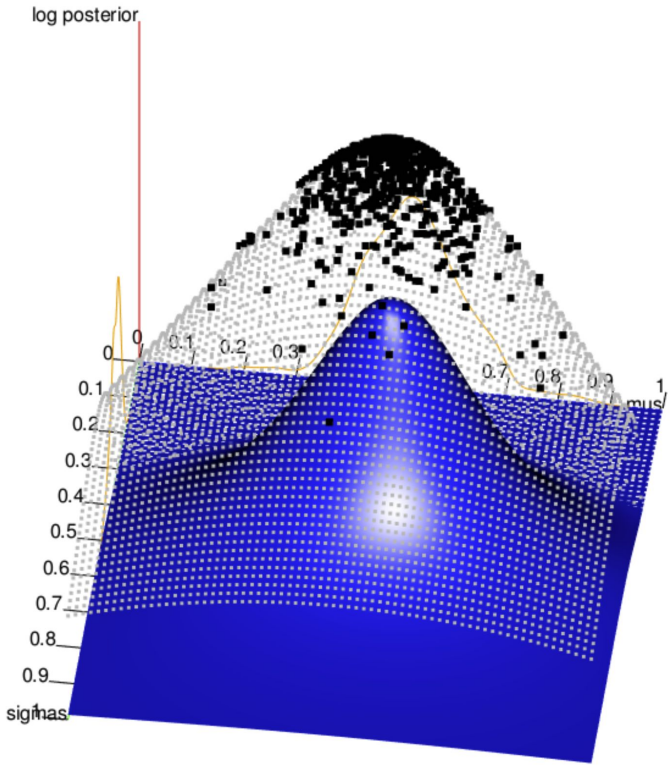


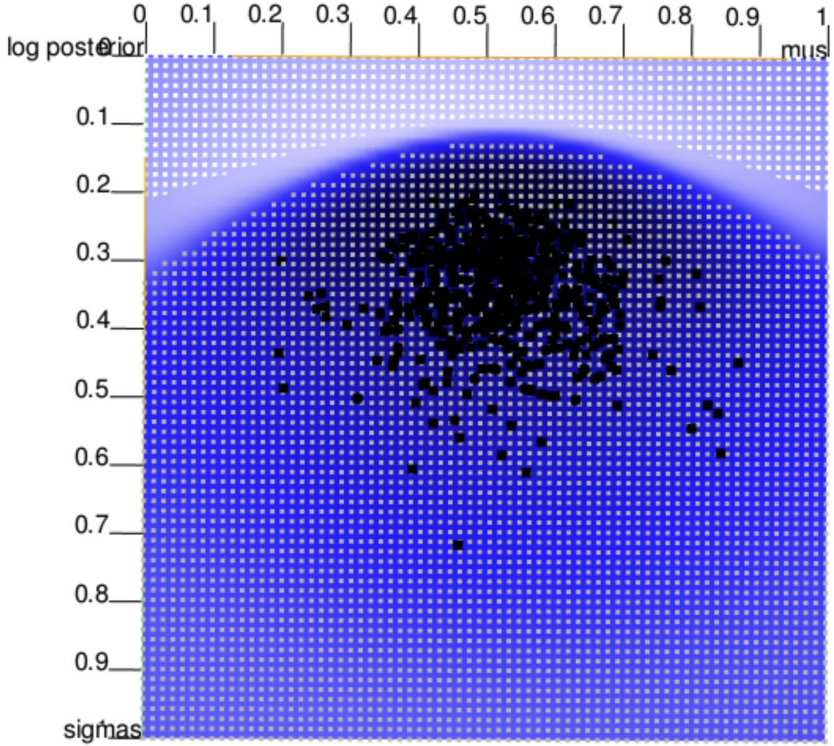
We recommend watching the video as it contains a running version of the particle.

Hamiltonian Monte Carlo



Hamiltonian Monte Carlo from the top.





Benefits and Problems: Hamiltonian Monte Carlo

- Benefits:
  - **Autocorrelation is low** between successive samples.
  - **Not limited** to normal distributed posterior.
  - Works under **correlation between parameters** in the posterior.
  - **Efficient when the number of parameters increases.**
- Problems:
  - Only works for **continuous parameters.**
  - Need **protection** against **U-turns** (NUTS). This is done in a warmup-phase, configuring the engine.
  - More **complicated configuration** and needs additional sanity checks.

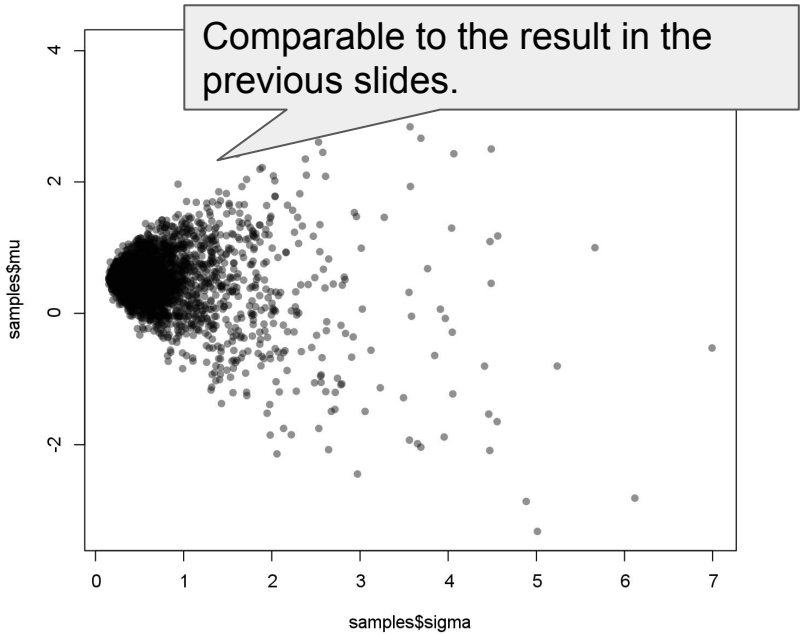
Stan

Running Hamiltonian Monte Carlo in STAN

```
library(rstan)
D <- c(0.312, 0.555, 0.24, 0.979)

model <- stan(model_code = "
data {
 vector[4] D;
}
parameters{
 real mu;
 real <lower=0> sigma;
}
model{
 D ~ normal(mu, sigma);
}
", data = list(D = D))

samples <- extract(model)
plot(samples$sigma, samples$mu, pch = 16, col = rgb(0, 0, 0, 0.4))
```



Execution log

We see information on the gradient computation.

```
Chain 1:
Chain 1: Gradient evaluation took 0 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0.025 seconds (Warm-up)
Chain 1:                0.018 seconds (Sampling)
Chain 1:                0.043 seconds (Total)
```

Running multiple chains in parallel is possible.

Warmup configure the engine, but warmup sample are not used.

Producing 1000 samples.

Summary

- Grid approximation
- Quadratic approximation
- Metropolis–Hastings
- Hamiltonian Monte Carlo