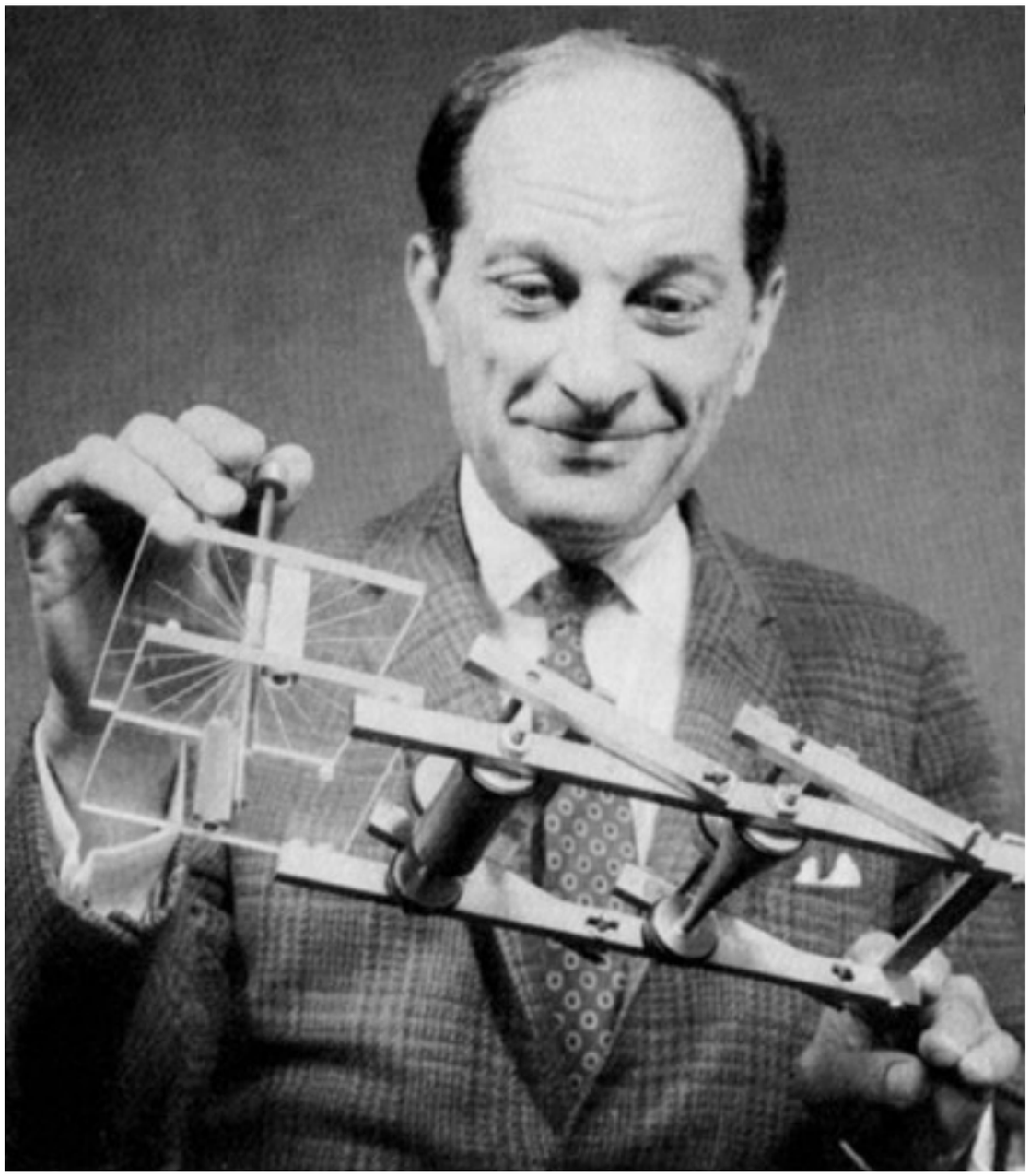
Intro to Stan



Stanislaw Ulam
H-Bomb
(1909-1984)

What is Stan?

- Probabilistic **programming language** and **inference algorithms**

- Stan **program**
  - declares data and (constrained) parameter variables
  - defines log posterior (or penalized likelihood)

- Stan **inference**
  - MCMC for full Bayes
  - VB for approximate Bayes
  - Optimization for (penalized) MLE

Why Stan?

- Fit rich Bayesian statistical models
- Efficiency
  - HMC + NUTS
  - Compiled to C++
- Flexible domain specific language
  - Extensible
- Open source
  - BSD

Who is using Stan?

Biological sciences

- clinical trials
- epidemiology
- genomics
- population ecology
- entomology
- ophthalmology
- neurology
- agriculture
- fisheries
- cancer biology

Who is using Stan?

Physical sciences

- astrophysics
  - LIGO gravitational wave observation
- molecular biology
- oceanography
- climatology

Who is using Stan?

Social sciences

- population dynamics
- psycholinguistics
- social networks
- political science
- human development
- economics
  - textbook coming soon! (ish)

Who is using Stan?

More…

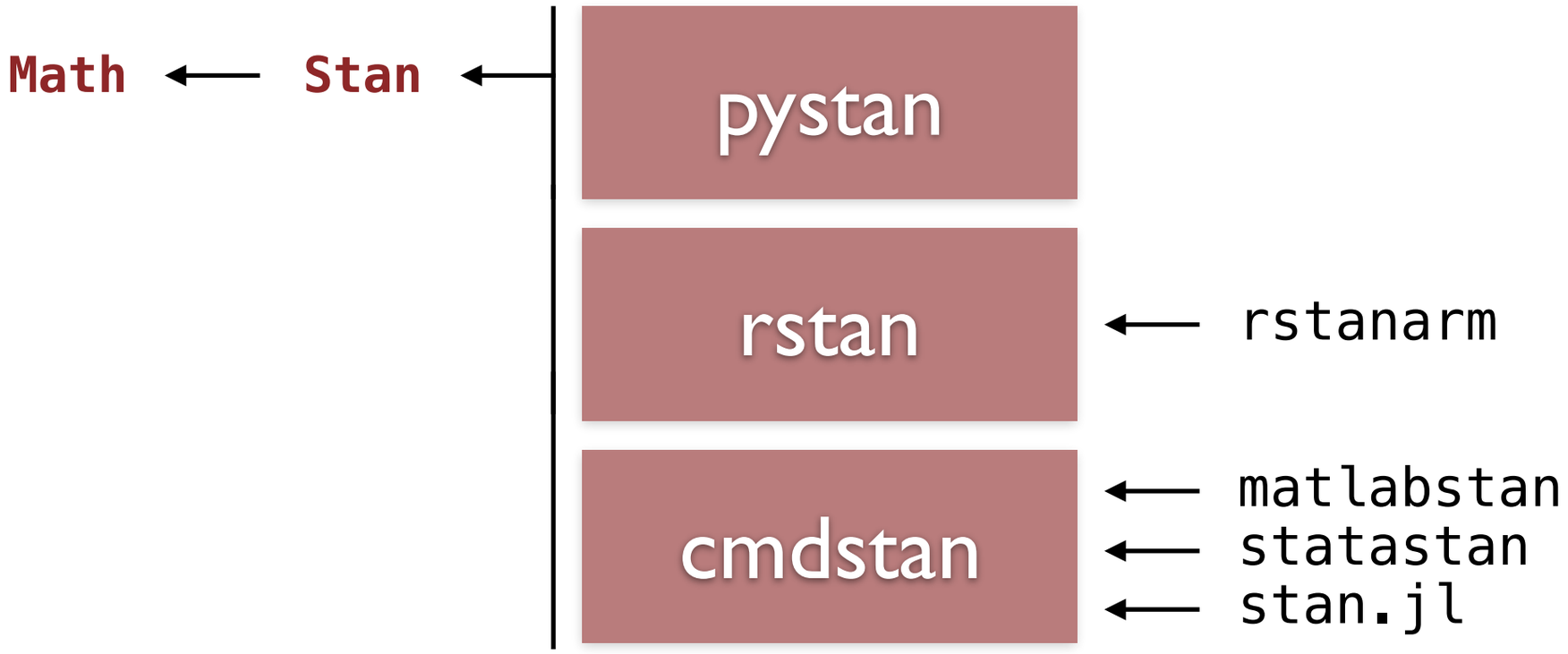
- sports
- public health
- publishing
- finance
- pharma
- actuarial
- recommender systems
- educational testing
- materials engineering

Who is using Stan?

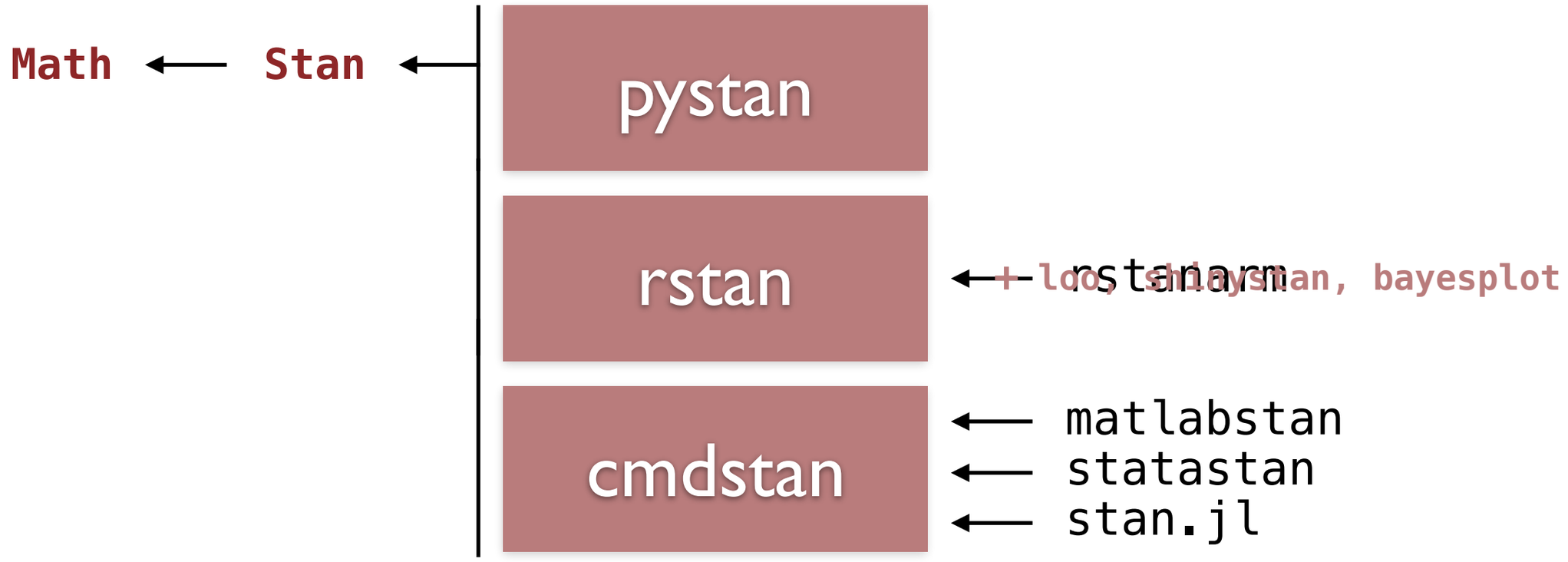
mc-stan.org/citations

Stan is many things

Interfaces


Math
Stan
pystan
rstan
rstanarm
cmdstan
matlabstan
statastan
stan.jl

Interfaces
+ Tools



Review of Bayes

Bayesian inference is concerned with computing the posterior density of interest and taking expectations

$$p(q|\mathcal{D}) = \frac{p(q) \times p(\mathcal{D}|q)}{p(\mathcal{D})}$$

$$p(\mathcal{D}) = \int p(q) \times p(\mathcal{D}|q) dq$$

$$\mathbb{E}_{p(q|\mathcal{D})}[f(q)]$$

We can't easily do these integrals so we use MCMC to generate a sequence of draws that approx. the posterior as S -> Infinity

$$\{q^{(1)}, q^{(2)}, \ldots, q^{(S)}\}$$

$$\mathbb{E}_{p(q|\mathcal{D})}[f(q)] \approx \frac{1}{S}\sum_{s=1}^{S} f(q_s)$$

Computation with MCMC draws

- Sequence of MCMC draws

$$\{q^{(1)}, q^{(2)}, \ldots, q^{(S)}\}$$

- Compute expectations

$$\mathbb{E}_{p(q|\mathcal{D})}[f(q)] \approx \frac{1}{S}\sum_{s=1}^{S} f(q_s)$$

- The degree of dependence among the draws governs how bad this approximation is for finite S
- **Effective sample size**: number of *independent* draws that would estimate a posterior mean with the same precision as the S *dependent* draws we actually have

To run HMC algorithm, must define the kernel of the posterior

$$p(q|\mathcal{D}) \propto p(q) \times p(\mathcal{D}|q)$$

To run HMC algorithm, must define the kernel of the posterior

$$p(q|\mathcal{D}) \propto p(q) \times p(\mathcal{D}|q)$$

$$\log(p(q|\mathcal{D})) \propto \log(p(q)) + \log(p(\mathcal{D}|q))$$

Posterior densities are specified in a comprehensive user-oriented probabilistic programming language.

$$p(q \mid \mathcal{D})$$

When writing a Stan program we always have three fundamental components

$$p(q \mid \mathcal{D})$$

What are we conditioning on?

What are the parameters?

How are they related?

We're now going to write a Stan program

- Open a new empty file in RStudio
- Save it as **`one-way-normal.stan`**

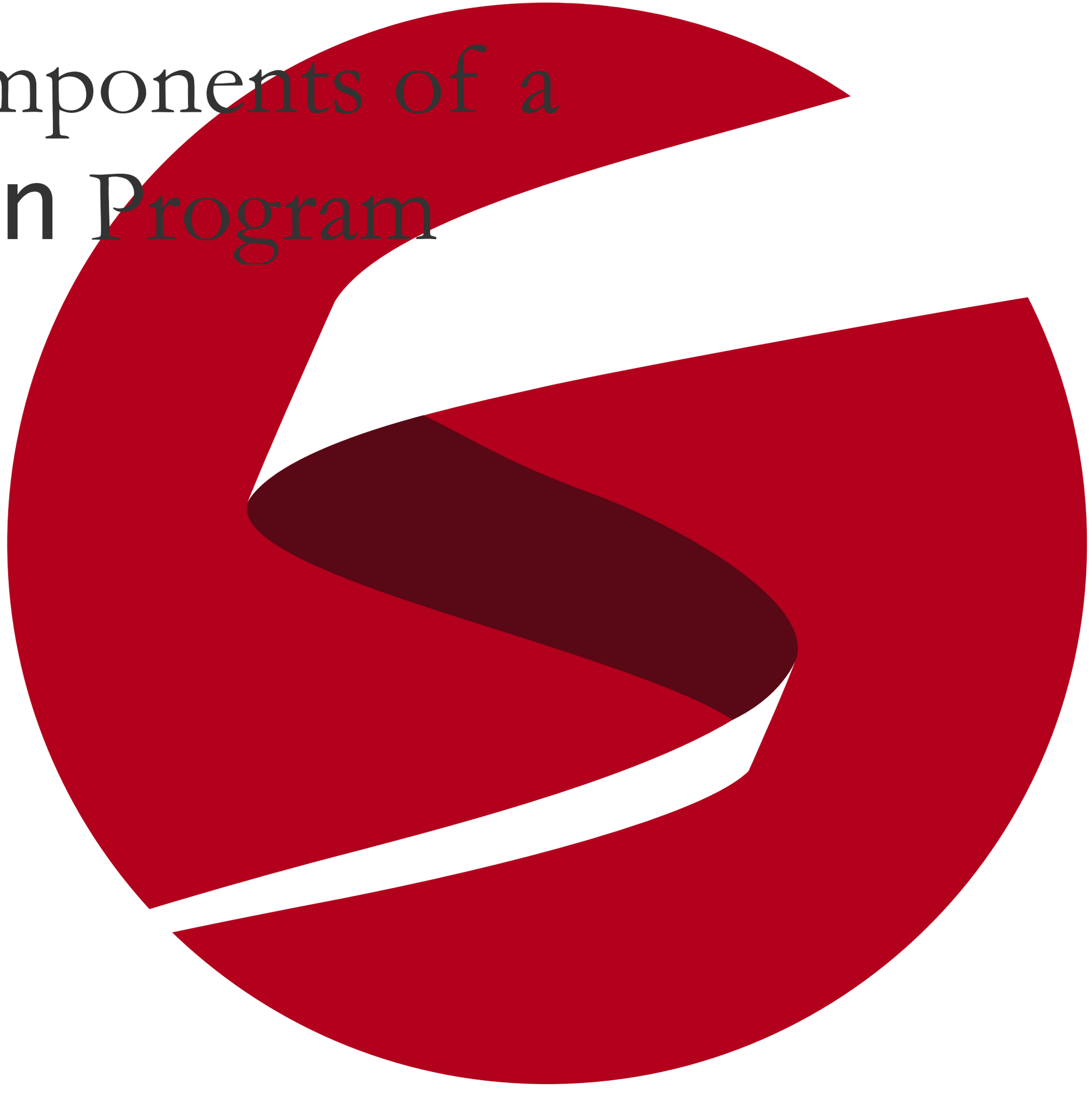
$$n \in \{1, \ldots, N\}$$

$$j \in \{1, \ldots, J\}$$

$$\theta_j \sim \text{Normal}(0, \sigma_\theta)$$

$$y_n \sim \text{Normal}(\alpha + \theta_{j[n]}, \sigma_y)$$

Components of a Stan Program



Stan programs are organized into *blocks*

```
block name {

  block contents

}
```

Data

- Declare data types, sizes, and constraints
- Read from data source and constraints validated
- Evaluated:
  - once

Data

```
data {
  // Dimensions
  int<lower=1> N;
  int<lower=1> J;

  // Variables
  int<lower=1, upper=J> idx_J[N];
  vector[N] y;
}
```

```
// single line comment
/* multiple lines of
comments */
```

Parameters

- Declare parameter types, sizes, and constraints
- Transformations (under the hood) for constrained parameters
- Evaluated:
  - every log prob evaluation

Parameters

```
parameters {
  real alpha;
  vector[J] theta;
  real<lower=0> sigma_y;
  real<lower=0> sigma_theta;
}
```

constraints *required* in **`parameters`** block

Model

- Statements defining the posterior density
  - log scale
- Evaluated:
  - every log prob evaluation

Model

```
model {
   for (n in 1:N)
     y[n] ~ normal(alpha + theta[idx_J[n]],
                   sigma_y);
}
```

Model

```
model {
   y ~ normal(alpha + theta[idx_J],
              sigma_y);
}
```

Multiple indexing

Model

```
model {
  y ~ normal(alpha + theta[idx_J],
             sigma_y);
}
```

Vectorized sampling statement

Model

```
model {
   y ~ normal(alpha + theta[idx_J],
              sigma_y);
   // priors  (flat, uniform, if omitted)
   theta ~ normal(0, sigma_theta);
}
```

Why is the default automatically uniform?

- $p(\theta) \propto 1$
- Nothing added to log prob

Model

```
model {
   y ~ normal(alpha + theta[idx_J],
              sigma_y);
   // priors  (flat, uniform, if omitted)
   theta ~ normal(0, sigma_theta);
   sigma_theta ~ normal(0, 3);
   sigma_y ~ cauchy(0, 10);
   alpha ~ normal(100, 20);

}
```

Generated Quantities

- Declare and define derived variables
  - (P)RNGs, predictions, event probabilities, decision making
- Constraints validated
- Evaluated:
  - once per draw

Generated Quantities

```
generated quantities {
   vector[N] y_rep;
   for (n in 1:N)
      y_rep[n] = normal_rng(theta[idx_J[n]], sigma_y);
}
```

one-way-normal.stan

```
data {
  int<lower=1>  N;
  int<lower=1>  J;
  int<lower=1, upper=J> idx_J[N];
  vector[N] y;
}
parameters {
  real alpha;
  vector[J] theta;
  real<lower=0> sigma_y;
  real<lower=0> sigma_theta;
}
model {
  y ~ normal(alpha + theta[idx_J], sigma_y);

  theta ~ normal(0, sigma_theta);
}
generated quantities {
  vector[N] y_rep;
  for (n in 1:N)
    y_rep[n] = normal_rng(alpha + theta[idx_J[n]],
                          sigma_y);
}
```

Transformed Data

- Declare and define transformed data variables
- Constraints validated
- Evaluated:
  - once (after `data`)

Transformed Data

If we wanted to collect group-level statistics we could do that here:

```
transformed data {
   vector[J] group_mean = rep_vector(0, J);
   vector[J] group_sd = rep_vector(0, J);
   vector[J] group_n = rep_vector(0, J);
   for (n in 1:N) {
     int j = idx_J[n];
     group_mean[j] = group_mean[j] + y[n];
     group_sd[j] = group_sd[j] + square(y[n]);
     group_n[j] = group_n[j] + 1;
   }
   group_sd = sqrt(inv(group_n - 1) * (group_sd
                    square(group_mean) ./ group_n));
   group_mean = group_mean ./ group_n;

}
```

Transformed Parameters

- Declare and define transformed parameter variables
- Constraints validated
- Evaluated:
  - every log prob evaluation

Transformed Parameters

```
transformed parameters {
   vector[J] theta_alt;
   theta_alt = alpha + theta;
}
```

Functions

- Declare and define functions to use in the body of the program
- Compiled with the model

Functions

```
functions {

  vector one_way_DGP_rng(real alpha, vector theta,
                         real sigma, int[] idx) {

    vector[size(idx)] y;
    for (i in 1:size(idx))
      y[i] = normal_rng(alpha + theta[idx[n]], sigma);

    return y;
  }

}
```

What's going on under the hood

- Stan always samples from a unconstrained parameter space

Remember the parameters block?

```
parameters {
  real alpha;
  vector[J] theta;
  real<lower=0> sigma_y;
  real<lower=0> sigma_theta;
}
```

constraints *required* in **`parameters`** block

Remember the parameters block?

```
parameters {
  real alpha;
  vector[J] theta;
  real<lower=0> sigma_y;
  real<lower=0> sigma_theta;
}
```

$$\sigma_y = \exp(\sigma'_y), \sigma'_y \in (-\infty, \infty)$$

$$\sigma_\theta = \exp(\sigma'_\theta), \sigma'_\theta \in (-\infty, \infty)$$

constraints *required* in **parameters** block

And now the model block defines the log-posterior

```
model {
   y ~ normal(alpha + theta[idx_J],
              sigma_y);
   // priors  (flat, uniform, if omitted)
   theta ~ normal(0, sigma_theta);
}
```

$$\begin{aligned}\text{target}(\theta_{1:J}, \sigma'_\theta, \sigma'_y, \alpha) = &\sum_{j=1}^{J} \text{Normal\_lpdf}(\theta_j | 0, \exp(\sigma'_\theta)) + \log(1) + \log(1) + \\ &\sum_{n=1}^{N} \text{Normal\_lpdf}(y_n | \alpha + \theta_{j[n]}, \exp(\sigma'_y)) + \\ &\log(\exp(\sigma'_\theta)) + \log(\exp(\sigma'_y))\end{aligned}$$

$$\text{Normal\_lpdf}(x | \mu, \sigma) := -0.5 \log(\sigma) - (x - \mu)^2 / (2\sigma)$$

The target log-density can be edited in the *model block*

```
model {

  target += expression

}
```

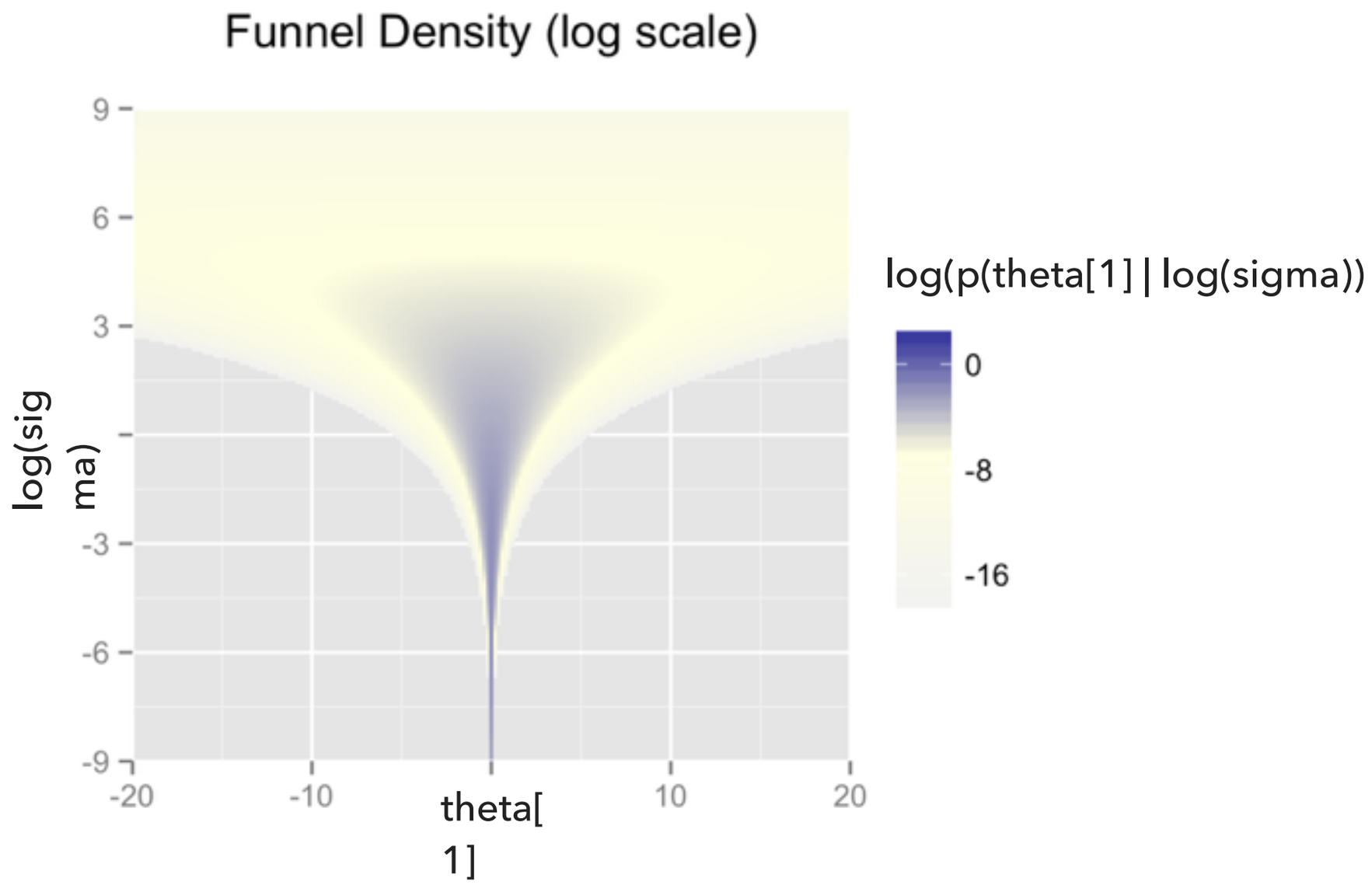
one-way-normal-suff-stat.stan

```
transformed data {
  vector[J] gp_mean = rep_vector(0, J);
  vector[J] gp_n = rep_vector(0, J);
  real ss = sum(square(y - mean(y)));
  for (n in 1:N) {
    j = idx_J[n];
    gp_mean[j] = gp_mean[j] + y[n];
    gp_n[j] = gp_n[j] + 1;
  }
  gp_mean = gp_mean ./ gp_n;
}

model {
  gp_mean ~ normal(alpha + theta[idx_J],
                   sigma_y * inv(sqrt(gp_n)));
  theta ~ normal(0, sigma_theta);
  target += (J - N) * log(sigma_y) - 0.5 * ss
            * inv(sigma_y);
}
```

one-way-normal-vectorized.stan

```
transformed data {
  int gp_n[J];
  int start[J];
  real y_sorted = y[sort_indices_asc(idx_J)];
  for (j in 1:J)
    gp_n[j] = sum(idx_J == j);
  {
    int c_n[J];
    c_n = cumsum(gp_n);
    start[1] = 1;
    start[2:J] = c_n[1:(J-1)] + 1;
  }
}

model {
  theta ~ normal(0, sigma_theta);
  for (j in 1:J)
    segment(y_sorted, start[j], gp_n[j]) ~
    normal(alpha + theta[j], sigma_y);
}
```

FUNNEL


Funnel Density (log scale)
9
6
3
-3
-6
-9
log(sig
ma)
-20
-10
10
20
theta[
1]
log(p(theta[1] | log(sigma))
0
-8
-16

What is going on? EHMC

- **Phase space**: q position (parameters); p momentum
- **Posterior density**: $\pi(q)$
- **Mass matrix**: M
- **Potential energy**: $V(q) = -\log \pi(q)$
- **Kinetic energy**: $T(p) = \frac{1}{2} p^\top M^{-1} p$
- **Hamiltonian**: $H(p,q) = V(q) + T(p)$
- **Diff eqs**:

$$\frac{dq}{dt} = +\frac{\partial H}{\partial p} \qquad \frac{dp}{dt} = -\frac{\partial H}{\partial q}$$

Leapfrog Integrator Steps

- Solves Hamilton's equations by **simulating dynamics**
  (symplectic [volume preserving]; ϵ^3 error per step, ϵ^2 total error)
- Given: **step size** ϵ, **mass matrix** M, **parameters** q
- **Initialize kinetic** energy, $p \sim \text{Normal}(0, \mathbf{I})$
- **Repeat** for L leapfrog steps:

$$p \leftarrow p - \frac{\epsilon}{2} \frac{\partial V(q)}{\partial q} \qquad \text{[half step in momentum]}$$

$$q \leftarrow q + \epsilon M^{-1} p \qquad \text{[full step in position]}$$

$$p \leftarrow p - \frac{\epsilon}{2} \frac{\partial V(q)}{\partial q} \qquad \text{[half step in momentum]}$$