A combinator library for MCMC kernels and proposals

Praveen Narayanan, May 8 2014

Motivation

MCMC samplers are complicated

$$\mathcal{A}\left(x^{(i)}, x^{\star}\right) = \min\left\{1, \frac{p(x^{\star})q\left(x^{(i)} \mid x^{\star}\right)}{p\left(x^{(i)}\right)q\left(x^{\star}|x^{(i)}\right)}\right\}$$

An Introduction to MCMC for Machine Learning, C. Andrieu et al., 2003

$$\mathcal{A}_{n\to m} = \min\left\{1, \frac{p(m, x_m^\star)}{p(n, x_n)} \times \frac{q(n \mid m)}{q(m \mid n)} \times \frac{q_{m\to n}(u_{m,n} \mid m, x_m^\star)}{q_{n\to m}(u_{n,m} \mid n, x_n)} \times \mathcal{J}_{f_{n\to m}}\right\}$$

An Introduction to MCMC for Machine Learning, C. Andrieu et al., 2003

$$\mathcal{A}_{split} = \min\left\{1, \frac{p(k+1, \mu_{k+1})}{p(k, \mu_k)} \times \frac{\frac{1}{k+1}}{\frac{1}{k}} \times \frac{1}{p(u_{n,m})} \times \mathcal{J}_{split}\right\}$$

An Introduction to MCMC for Machine Learning, C. Andrieu et al., 2003

$$\mathcal{A}_{merge} = \min\left\{1, \frac{p(k-1, \mu_{k-1})}{p(k, \mu_k)} \times \frac{\frac{1}{k-1}}{\frac{1}{k}} \times \mathcal{J}_{merge}\right\}$$

An Introduction to MCMC for Machine Learning, C. Andrieu et al., 2003

We would like predefined samplers

MCMC depends on “good” proposals

A 1-D target distribution

$$p(x) \propto 0.3 \exp(-0.2x^2) + 0.7 \exp(-0.2(x-10)^2)$$

A gaussian might work as a proposal

What is the optimal mean? Standard deviation?

We would like to easily experiment with different proposals on a sampler

A second look

$$p(x) \propto 0.3 \exp(-0.2x^2) + 0.7 \exp(-0.2(x-10)^2)$$

A mixture of gaussians might work even better

We would like to easily mix various proposals

Quick info

Proposal and target combinators

Kernel combinators

Making the random walk

Future work

Quick info

Proposal and target combinators

Kernel combinators

Making the random walk

Future work

The library is written in Haskell

It extends Indiana's probabilistic programming system

Consists of three modules:

Distributions, Actions, Kernels

Quick info

Proposal and target combinators

Kernel combinators

Making the random walk

Future work

Targets vs proposals

Targets have a density

```haskell
type Density a = a → Probability
```

```haskell
data Target a = T (Density a)
```

Proposals have a density and can be sampled from

```haskell
type Sample a = Rand → IO a
```

```haskell
data Proposal a =
       P (Density a) (Sample a)
```

Proposals

What proposals are already defined?

uniform, normal,

categorical

```
uniform [0,2,4] [3,5,7]

normal [0,1,4,7] (diag [2,2,2,2])

categorical
["pizza", "sushi", "curry"]
[0.7, 0.9, 0.3]
```

How do we create our own proposals?

Mix existing proposals

```
proposalMix ::
 [Proposal a] → [Double] → Proposal a
```

```
gMix = let g1 = normal [0,0] (diag [1,1])
           g2 = normal [5,5] (diag [2,2])
       in proposalMix [g1, g2] [0.3, 0.7]
```

Increase dimensions of existing proposals

Use `updateNth`, `updateBlock`

```
updateNth :: Int
          → ([a] → Proposal [a])
```

```
condProposal =
updateNth
4
(λy → normal y [[1]])
```

Update component a_4

for a point $(a_1, a_2, a_3, a_4, \ldots, a_n)$

```
updateBlock :: Int
            → Int
            → ([a] → Proposal [a])
            → ([a] → Proposal [a])
```

```
condProposal =
updateBlock
1
2
(λy → normal y (diag [1,1])
```

Update components a_1, a_2
for a point $(a_1, a_2, \ldots, a_n)$

Construct our own

using `makeProposal`

```
customProposal =
let density = ...
    sample = ...
in makeProposal density sample
```

Targets

How do we create targets?

Use `makeTarget`

```
t1 = makeTarget $
     λ[x] → 0.3 * exp (-0.2*x*x)
```

Mix existing targets, using `targetMix`

Quick info

Proposal and target combinators

Kernel combinators

Making the random walk

Future work

What is a kernel?

A function that, when given a target and conditional proposal, performs MCMC steps

```
type Step x = Rand → x → IO x

type Kernel x a =
  Target a →
  (a → Proposal a) →
  Step x
```

What kernels are already defined?

metropolisHastings,
simulatedAnnealing

How do we define our own steps?

Mix some steps

```
mixSteps :: [Step x]
         → [Double]
         → Step x
```

```haskell
mhMix = let mh = metropolisHastings
            mh1 = mh target proposal1
            mh2 = mh target proposal2
        in mixSteps [mh1, mh2] [0.7, 0.3]
```

Cycle through a kernel

```
cycleKernel :: Kernel x a
            → Target a
            → [a → Proposal a]
            → Step x
```

```
mhCycle =
cycleKernel
metropolisHastings
target
[p1, p2, p3]
```

Quick info

Proposal and target combinators

Kernel combinators

Making the random walk

Future work

```
walk :: Step x
     → x
     → Int
     → Rand
     → Action x a
     → IO a
```

Start state

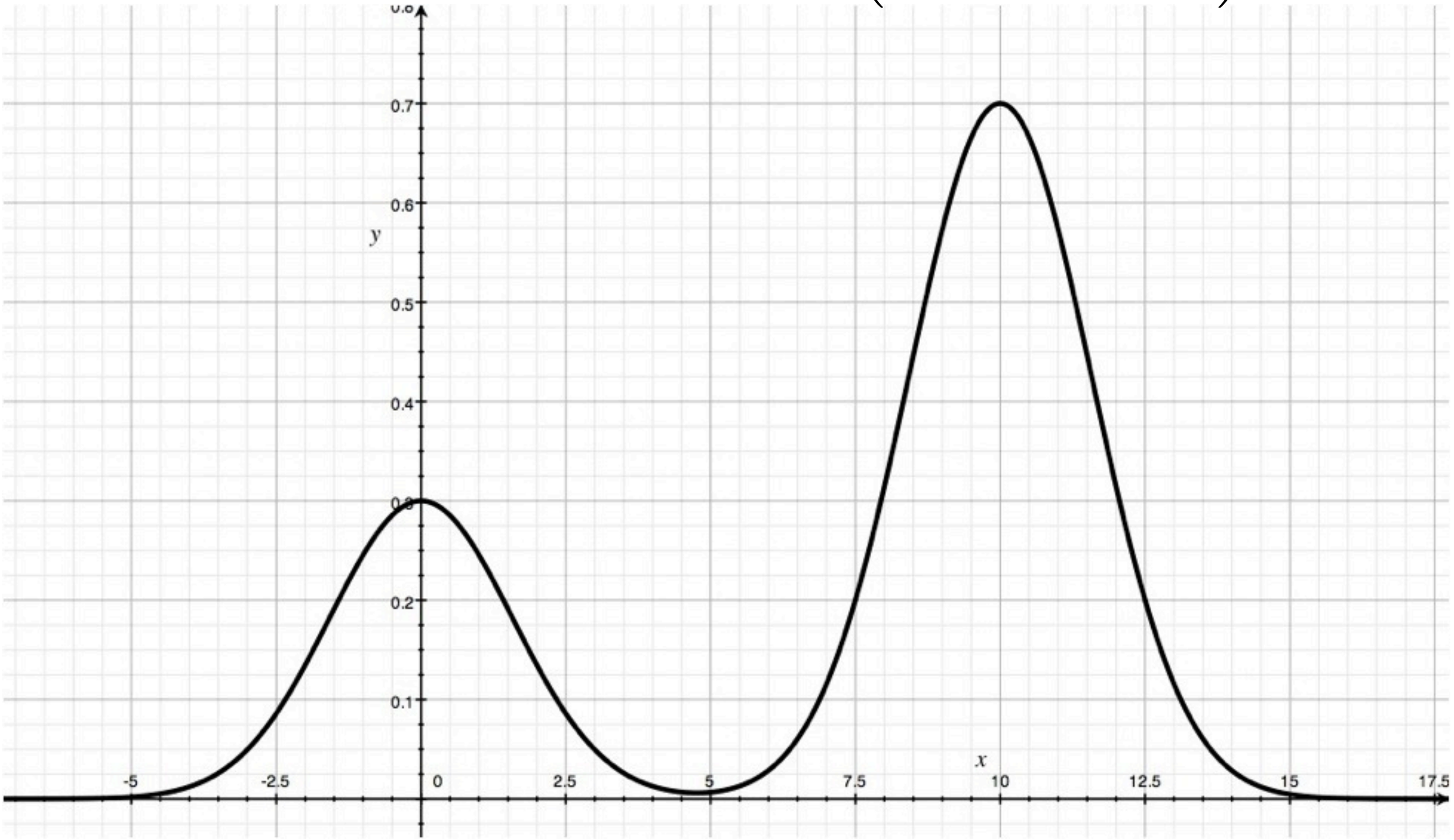
Number of steps

Action to take at each step

A third look

$$p(x) \propto 0.3 \exp(-0.2x^2) + 0.7 \exp(-0.2(x-10)^2)$$

$$0.3\exp\left(-0.2x^2\right) \;+\; 0.7\exp\left(-0.2\left(x-10\right)^2\right)$$



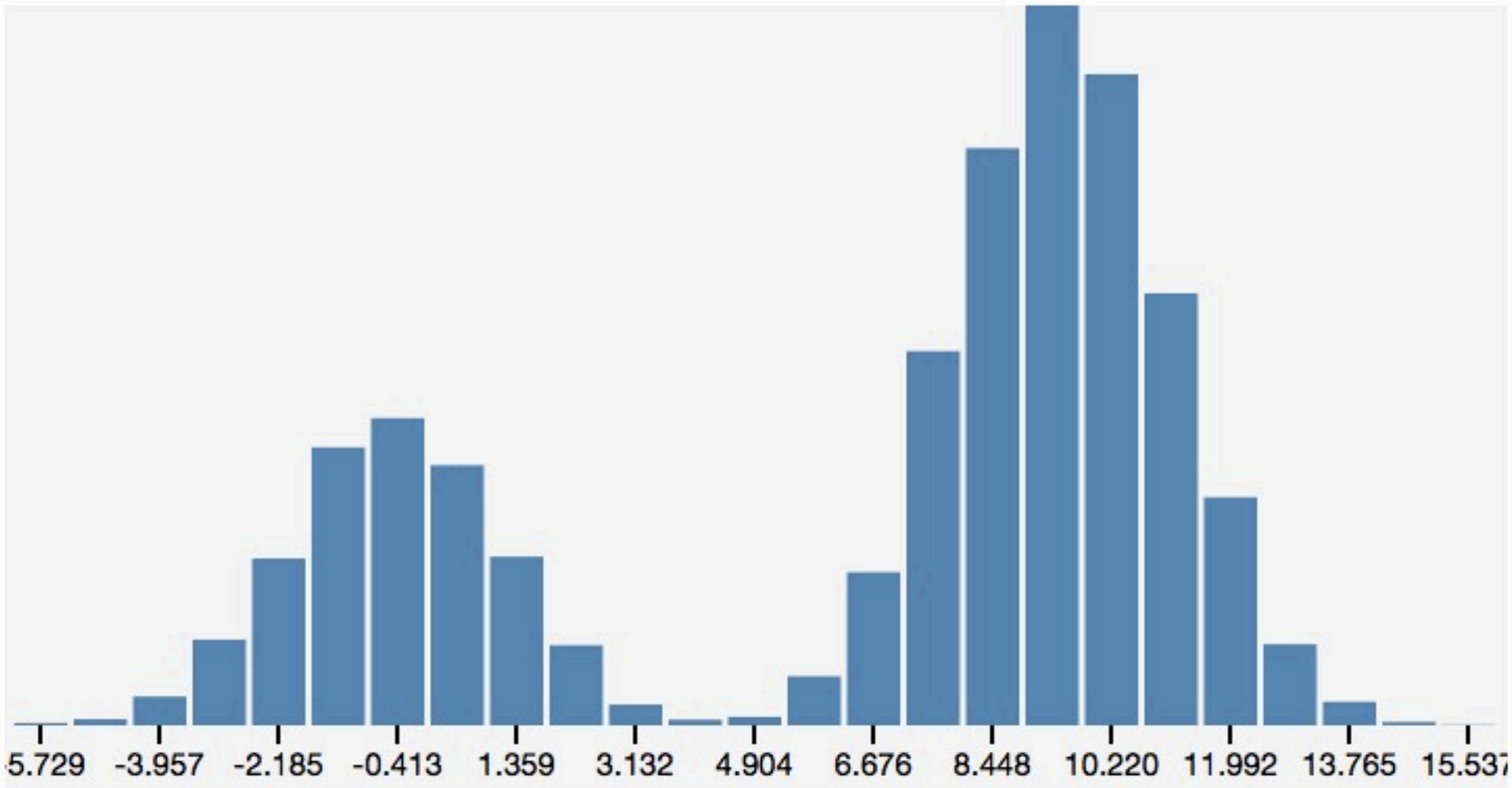
```haskell
mhTest :: IO ()
mhTest = do
    g ← MWC.createSystemRandom
    let (t,p) = (... , ...)
        mh = metropolisHastings t p
        act = every 100 (batchViz vizMH 50)
    walk mh [0] (10^6) g act
```

Number of samples plotted: 10000

x

Min: -5.7291215850716135

Max: 16.422778145776434



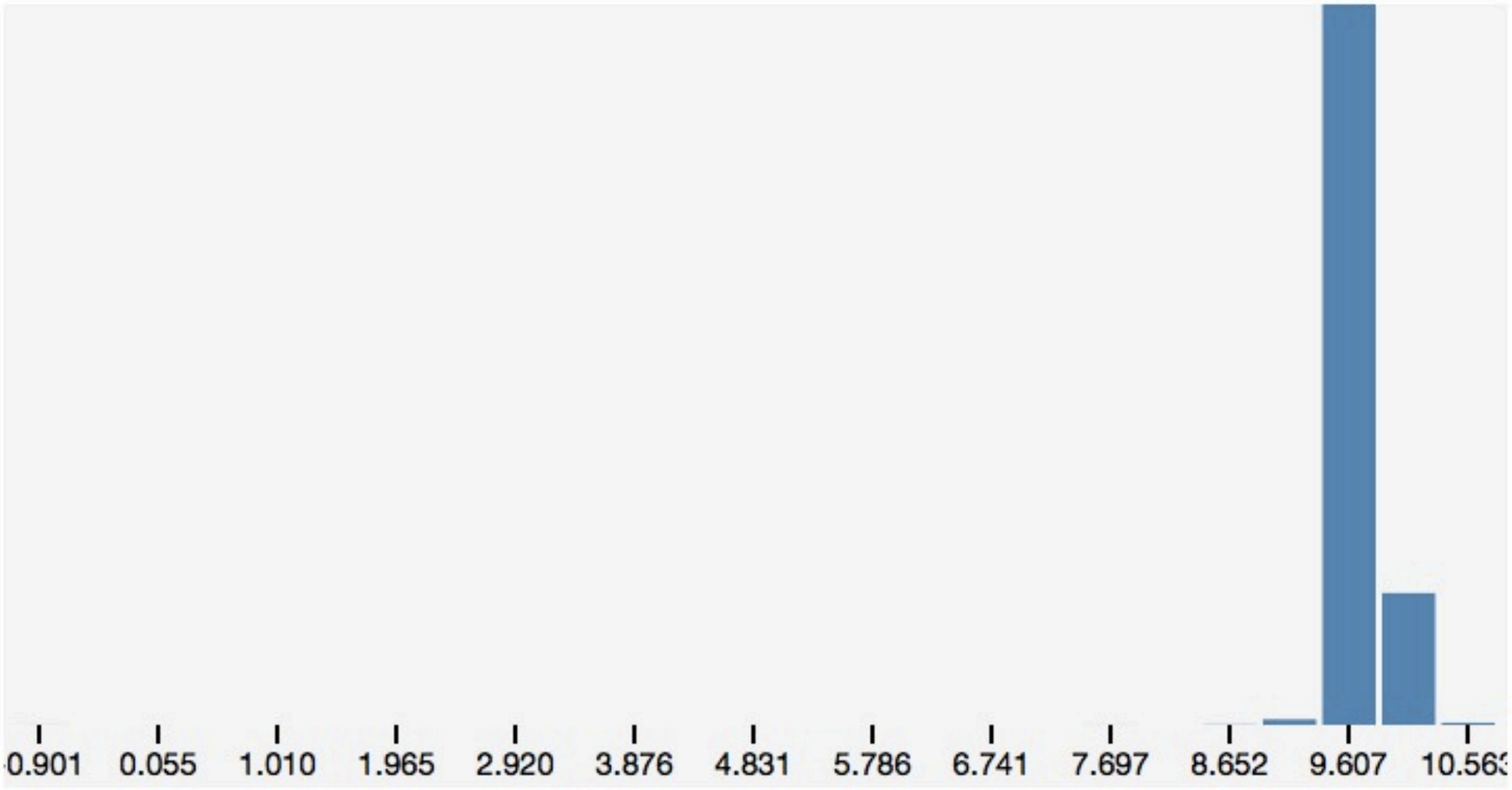
```haskell
saTest :: IO ()
saTest = do
    g ← MWC.createSystemRandom
    let (t,p) = (... , ...)
        sa = simulatedAnnealing t p
        coolSch temp = temp * (1 - 1e-3)
        x0 = ([0], 1, coolSch)
        act = every 100 (batchViz vizSA 50)
  walk sa x0 (10^6) g act
```

Number of samples plotted: 9889

x

Min: -0.9005223518480108

Max: 11.040135975148 39



```haskell
blockMH :: Step [Double]
blockMH = let target = normal [0,1,4,7] (diag [2,2,2,2])
              mh4D = metropolisHastings target
              mh1 = mh4D $ updateBlock 1 2
                           (λy → normal y (diag [1,1]))
          mh2 = mh4D $ updateBlock 3 3 (λy → normal y [[1]])
          mh3 = mh4D $ updateNth 4 (λy → normal y [[5]])
      in mixSteps [mh1, mh2, mh3] [0.5, 0.4, 0.7]
```

```haskell
blockTest :: IO ()
blockTest = do
  g ← MWC.createSystemRandom
  let a = batchPrint printMH 50
  walk blockMH [0,0,0,0] (10^6) g a
```

[0.907,-1.067,2.128,5.558],
[0.907,-1.067,2.128,6.584],
[1.336,-1.198,2.128,6.584],
[1.336,-1.198,2.128,6.584],
[-0.857,-1.729,2.128,6.584],
[-0.857,-1.729,2.660,6.584],
[-0.857,-1.729,2.660,6.584],
[-0.857,-1.729,2.660,8.325],
[-0.857,-1.729,2.660,8.325],
[-0.857,-1.729,2.660,8.325],
[0.598,-2.241,2.660,8.325],
[0.598,-2.241,2.660,8.325],
[0.598,-2.241,2.660,8.325],
[0.598,-2.241,2.660,7.699],
[0.598,-2.241,2.660,7.699], ...

Quick info

Proposal and target combinators

Kernel combinators

Making the random walk

Future work

Kernel combinators for changing the accept ratio

More sampling methods

Approximate MH, Stochastic EM, Reversible jump MCMC

Thank you