A combinator library for MCMC sampling

Praveen Narayanan and Chung-chieh Shan
Indiana University

Available on Hackage as mcmc-samplers

Motivation

Reuse and compose inference techniques, just as we wish to do with models. For MCMC sampling, reuse and compose proposal distributions, transition kernels, and operations on sample-streams.

Example: Gaussian Mixture Model

```
data GaussianMixtureState = GMM
  { labels :: [Bool]
  , gaussParams :: ((Double, Double), (Double, Double))
  , bernParam :: Double
  , obs :: [Double] }

gmmProposal :: GaussianMixtureState
            → Proposal GaussianMixtureState
gmmProposal gmm = mixProposals
  [ (updateGaussParams gaussParamsProposal gmm,  1)
  , (updateBernParam   bernParamProposal   gmm,  2)
  , (updateLabels      labelsProposal      gmm, 10) ]

labelsProposal :: [Bool] → Proposal [Bool]
labelsProposal ls = chooseProposal nPoints f
  where f n = updateNth n flipBool ls
        flipBool bn = if bn then bern 0 else bern 1

gmmTarget :: Target GaussianMixtureState
gmmTarget = makeTarget (productDensity
            [ labelsTarget, gaussParamsTarget
            , bernParamTarget, obsTarget ])

gmmMH :: Step GaussianMixtureState
gmmMH = metropolisHastings gmmTarget gmmProposal
```

The random walk

```
run = do
  rng ← createSystemRandom
  let state0 = ...
  walk gmmMH state0 (10^6) rng (every 100 display)
```

Optimized sampler: don't calculate factors that cancel out

```
gmmOpt :: Step GaussianMixtureState
gmmOpt = mixSteps [ (stepGaussParams, 1)
                  , (stepBernParam 2)
                  , (stepLabels, 10) ]

stepLabels :: Step GaussianMixtureState
stepLabels = metropolisHastings (makeTarget dens)
                                labelsProposal
    where dens state = density labelsTarget state
                     * density obsTarget state
```

Example: Independent Metropolis within Gibbs

(Neiswanger, Wang, Zing)

```
imgStep :: EstimatorWeight
        → ([Int] → Proposal x)
        → [Int] → IO x
imgStep mhTarg prop tDot = do
  let unifT = categorical (zip [1..capT] [1..])
      mhProp m = updateNth m (const unifT)
      props = map mhProp [1..capM]
      mhStep = cycleStep metropolisHastings mhTarg props
  tDot' ← mhStep g tDot
  theta_i ← sampleFrom (prop tDot') g
  return theta_i
```

Basic types

```
type Density a = a → Double
type Sample a = Rand → IO a
type Step a = Rand → a → IO a
data Action m x a b
```

Create distributions

```
makeProposal :: Density a
             → Sample a
             → Proposal a

makeTarget :: Density a
           → Target a
```

Standard distributions ⟶

- Compute the density at a point
- Sample from

```
bern :: Double → Proposal Bool
categorical :: [(a, Double)] → Proposal a
beta :: Double → Double → Proposal Double
mvNormal :: [Double] → [[Double]] → Proposal [Double]
mvUniform :: [a] → [a] → Proposal [a]
```

Combinators

```
mixProposals :: [(Proposal a, Double)]
             → Proposal a
```
Relative weights
Mixture density + sampling

```
chooseProposal :: Int
               → (Int → Proposal a)
               → Proposal a
```
Uniform mixture

```
updateNth :: Int
          → (a → Proposal a)
          → [a] → Proposal [a]
```
Conditional proposal (focus)
Change only focused area

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

productDensity :: [d a] → Density a
```

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

```
metropolisHastings :: Kernel a a

type SA a = (a, (Temp, CoolingSchedule))
simulatedAnnealing :: Kernel (SA a) a

display :: Action IO x () ()

collect :: Action IO x [x] [x]

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

every :: Int
      → Action m x a b
      → Action m x (a,Int) b

mixSteps :: [(Step x, Double)]
         → Step x

cycleStep :: Kernel x a
          → Target a
          → [a → Proposal a]
          → Step x
```
List of conditional proposals to apply in a cycle

```
type EstimatorWeight = Target [Int]

nonParametric :: EstimatorWeight

parametric :: EstimatorWeight

semiParametric :: EstimatorWeight

pWalk :: [z]
      → ([z] → Step x)
      → x
      → Int
      → Estimator x
      → Rand → IO [x]
```

Current and future work

- Track dependencies in the probabilistic program for automatic optimizations
- Provide as reusable blocks more sampling methods: reversible jump, HMC
- Code generator for automatic update combinator derivation

Related work

Venture's inference language (Mansinghka et al.), FACTORIE's MCMC infrastructure (McCallum et al.), and BLAISE's SDK graphs (Bonawitz et al.).