Algorithms for Inference - exercises

Exercise 1. Sampling Implicit Curves

In the code box below, the `curve` function defines a vaguely heart-shaped curve. Below, we use rejection sampling to sample points along the boundary of the curve.

```
x
```

```
// takes z = 0 cross section of heart surface to some tolerance
```

```
// see http://mathworld.wolfram.com/HeartSurface.html
```

```
var onCurve = function(x, y) {
```

```
  var x2 = x*x;
```

```
  var term1 = y - Math.pow(x2, 1/3);
```

```
  var crossSection = x2 + term1*term1 - 1;
```

```
  return Math.abs(crossSection) < 0.01;
```

```
};
```

```
var xbounds = [-1, 1];
```

```
var ybounds = [-1, 1.6];
```

```

```

```
var xmu = 0.5 * (xbounds[0] + xbounds[1]);
```

```
var ymu = 0.5 * (ybounds[0] + ybounds[1]);
```

```
var xsigma = 0.5 * (xbounds[1] - xbounds[0]);
```

```
var ysigma = 0.5 * (ybounds[1] - ybounds[0]);



var model = function() {
  var x = gaussian(xmu, xsigma);
  var y = gaussian(ymu, ysigma);
  condition(onCurve(x, y));
  return {x: x, y: y};
};



var post = Infer({method: 'rejection', samples: 1000}, model);
viz.auto(post);
```

run ▼

a)

Try using MCMC with Metropolis-Hastings instead of rejection sampling. You'll notice that it does not fare as well as rejection sampling. Why not? 因为每一条mc链都被限制到了local

b)

Change the *model* to make MH successfully trace the curves. Your solution should result in a graph that clearly traces a heart-shaped figure – though it need not do quite as well as rejection sampling. Why does this work better?

HINT: is there a way you can sample a single (x,y) pair instead of separately sampling x and then y? You might want to check out the distribution DiagCovGaussian() (https://webppl.readthedocs.io/en/master/distributions.html#DiagCovGaussian) in the docs. Note that it expects parameters to be Vectors (https://webppl.readthedocs.io/en/master/functions/tensors.html#Vector) and you can extract elements from vectors with `T.get` (`T` is webppl shorthand for `ad.tensor`: for more information on tensor functions, see adnn docs (https://github.com/dritchie/adnn/blob/master/ad/README.md#available-ad-primitive-functions)).

c)

Now change the the inference *algorithm* (with the original model) to successfully trace the

curves. What parameters worked best? *Why* does this inference algorithm work better than MH?

HINT: you may want to explore HMC! start with the default parameters specified in the HMC docs (https://webppl.readthedocs.io/en/master/inference/methods.html#mcmc) and play with different values. 调不出来

Exercise 2. Properties and pitfalls of Metropolis-Hastings

Consider a very simple function that interpolates between two endpoints.

Suppose one endpoint is fixed at `-10`, but we have uncertainty over the value of the other endpoint and the interpolation weight between them. By conditioning on the resulting value being close to 0, we can infer what the free variables must have been:

```
var interpolate = function(point1, point2, interpolationWeight) {
  return (point1 * interpolationWeight +
          point2 * (1 - interpolationWeight))
}

var model = function(){
  var point1 = -10;
  var point2 = uniform(-100,100);
  var interpolationWeight = uniform(0,1);
  var pointInMiddle = interpolate(point1, point2, interpolationWeight);
  observe(Gaussian({mu: 0, sigma:0.1}), pointInMiddle)
  return {point2, interpolationWeight, pointInMiddle}
}

var posterior = Infer({
```

```
  method: 'MCMC',
  samples: 5000,
  lag: 100,
}, model)

var samples = posterior.samples;
viz(marginalize(posterior, function(x) {return x.pointInMiddle}))

// Store these for future use
editor.put("posterior", posterior)
editor.put("samples", samples)
```

run ▼

By looking at the marginal distribution of `pointInMiddle`, we can see that `Infer()` successfully finds values of `point2` and `interpolationWeight` that satisfy our condition.

a)

Visualize the separate marginal distributions of `point2` and `interpolationWeight`. How would you describe their shapes, compared to the marginal distribution of `pointInMiddle`?

Now visualize the *joint* marginal distribution of point2 and interpolationWeight. What does this tell you about their dependence?

HINT: use the marginalize (http://docs.webppl.org/en/master/functions/other.html#marginalize) helper to elegantly construct these marginal distributions

b)

WebPPL also exposes the list of MCMC samples that the density plots above are built from. This is saved in the `samples` variable. Decrease the number of samples to `50` (and the `lag` to 0) and plot `pointInMiddle` as a function of the sample number. Run this several times to get a feel for the shape of this curve. What do you notice? What property of MCMC are you observing?

HINT: this will require some 'data munging' on the array of samples. Some useful functions will be `map` (http://docs.webppl.org/en/master/functions/arrays.html#map), `_.range()`, and `viz.line` which takes arrays `x` and `y`.

c)

Try re-writing the model to use rejection sampling. Note that you will need to find a way to turn the `observe` statement into a `condition` statement (Hint: See Exercise #1). Is using rejection sampling here a good idea? Why or why not?

d)

Add `verbose: true` to the list of options and run `MH` again. What is the acceptance rate over time (i.e. what proportion of proposals are actually accepted in the chain?). What about the model puts it at this level?

Consider the list of built-in drift kernels here (https://webppl.readthedocs.io/en/master/driftkernels.html?highlight=drift%20kernel#helpers). Which of these would be appropriate to use in your model in place of the current uniform prior from which `point2` is sampled? After using that kernel in your model, what effect do you observe on the acceptence rate, and why?