Computational Methods

STAT 489-01: Bayesian Methods of Data Analysis *

Spring Semester 2017

Contents

1 Importance Sampling **2**

2 Markov Chain Monte Carlo **2**

2.1 Metropolis and Metropolis-Hastings Algorithms . 3

2.1.1 Metropolis Algorithm 3

2.1.2 Why It Works 4

2.1.3 Metropolis-Hastings Algorithm 5

2.1.4 Dirichlet Example 5

2.2 Gibbs Sampler 12

2.2.1 Dirichlet Example 13

2.3 Using JAGS for MCMC 16

2.4 Example: Hierarchical Normal Model (Gelman 11.6) 19

3 Hamiltonian Monte Carlo **21**

3.1 Maximum Entropy 22

3.2 Maximum Entropy as a Statistical Principle . . . 23

3.3 Hamiltonian Mechanics 24

3.4 Programming HMC with Stan 26

*Copyright 2017, John T. Whelan, and all that

Tuesday 21 March 2017 – Refer to Chapters 10 and 11 of Gelman or Chapter 8 of McElreath or Chapter 7 of Kruschke

We've seen that random samples from the posterior distribution $p(\boldsymbol{\theta}|\mathbf{y},I)$ is useful for estimating expectation values, marginal distributions of parameters, etc. So far we've always been able to draw samples directly from the distribution in question, since it's been standard and simple enough that standard statistical software has had it programmed in. In more complicated situations, such when the distribution is nonstandard or when the parameter space has many dimensions, this is impossible or impractical. There are a number of numerical methods which can effectively draw samples from the desired distribution in these cases.

The distribution $p(\boldsymbol{\theta}|\mathbf{y},I)$ from which we're trying to sample is known as the *target distribution*. Note that we usually don't have to have the functional form of the normalized distribution. If $p(\boldsymbol{\theta}|\mathbf{y},I) = Nq(\boldsymbol{\theta}|\mathbf{y},I)$ for some known $q(\boldsymbol{\theta}|\mathbf{y},I)$, we usually don't need to calculate

$$N(\mathbf{y},I) = \frac{1}{\int q(\boldsymbol{\theta}|\mathbf{y},I)\,d\boldsymbol{\theta}} \tag{0.1}$$

which involves a possibly difficult integral over the parameter space. For instance,

$$\begin{aligned} E[h(\boldsymbol{\theta})|\mathbf{y},I] &= \int h(\boldsymbol{\theta})\, p(\boldsymbol{\theta}|\mathbf{y},I)\, d\boldsymbol{\theta} = \frac{\int h(\boldsymbol{\theta})\, q(\boldsymbol{\theta}|\mathbf{y},I)\, d\boldsymbol{\theta}}{\int q(\boldsymbol{\theta}|\mathbf{y},I)\, d\boldsymbol{\theta}} \\ &\approx \frac{1}{S}\sum_{s=1}^{S} h(\boldsymbol{\theta}_s) \end{aligned} \tag{0.2}$$

where $\{\boldsymbol{\theta}_s\}$ are parameter space points drawn proportional to $q(\boldsymbol{\theta}|\mathbf{y},I)$.

1 Importance Sampling

Before we get into random walk methods, let's consider another case where it is impractical to draw directly from the desired distribution $p(\boldsymbol{\theta}|\mathbf{y},I)$, but we have some "similar" distribution $g(\boldsymbol{\theta})$ from which we can easily draw a sample. In that case, if $\{\boldsymbol{\theta}_s\}$ is such a sample, the average of any function $k(\boldsymbol{\theta})$ can be approximated as

$$\int k(\boldsymbol{\theta})\, g(\boldsymbol{\theta})\, d\boldsymbol{\theta} \approx \frac{1}{S}\sum_{s=1}^{S} k(\boldsymbol{\theta}_s) \tag{1.1}$$

If we want to get the expectation value of $h(\boldsymbol{\theta})$ under $p(\boldsymbol{\theta}|\mathbf{y},I)$, we can construct

$$\begin{aligned} E[h(\boldsymbol{\theta})|\mathbf{y},I] &= \int h(\boldsymbol{\theta})\, p(\boldsymbol{\theta}|\mathbf{y},I)\, d\boldsymbol{\theta} = \frac{\int h(\boldsymbol{\theta})\, q(\boldsymbol{\theta}|\mathbf{y},I)\, d\boldsymbol{\theta}}{\int q(\boldsymbol{\theta}|\mathbf{y},I)\, d\boldsymbol{\theta}} \\ &= \frac{\int h(\boldsymbol{\theta})\, \frac{q(\boldsymbol{\theta}|\mathbf{y},I)}{g(\boldsymbol{\theta})}\, g(\boldsymbol{\theta})\, d\boldsymbol{\theta}}{\int \frac{q(\boldsymbol{\theta}|\mathbf{y},I)}{g(\boldsymbol{\theta})}\, g(\boldsymbol{\theta})\, d\boldsymbol{\theta}} \approx \frac{\frac{1}{S}\sum_{s=1}^{S} h(\boldsymbol{\theta}_s)\, w(\boldsymbol{\theta}_s)}{\frac{1}{S}\sum_{s=1}^{S} w(\boldsymbol{\theta}_s)} \end{aligned} \tag{1.2}$$

where

$$w(\boldsymbol{\theta}) = \frac{q(\boldsymbol{\theta}|\mathbf{y},I)}{g(\boldsymbol{\theta})} \tag{1.3}$$

is a weighting function that corrects for the fact that the sample was drawn from $g(\boldsymbol{\theta})$ rather than $q(\boldsymbol{\theta}|\mathbf{y},I)$.

This looks very powerful and easy, but it can go very wrong if there are regions of the parameter space where $g(\theta)$ is negligible but $q(\boldsymbol{\theta}|\mathbf{y},I)$ is significant. An extreme version would be if there are some regions where $g(\theta) = 0$ but $q(\boldsymbol{\theta}|\mathbf{y},I) \neq 0$. The weighting factor would be infinite at those points, but they would never be drawn. If $g(\theta)$ is finite but small somewhere, those values would be very rare, but when they occurred, their huge weights would take over the estimate, making things very unstable. So in general, it's bad if the weights become too large compared to their typical values, which can happen e.g., if the target distribution $q(\boldsymbol{\theta}|\mathbf{y},I)$ has bigger tails than the sampling distribution $g(\boldsymbol{\theta})$. So this is why we cannot always for example use the Gaussian approximation centered on the MAP point to approximate a distribution, even with importance sampling.

2 Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) methods execute a random walk through parameter space, and are designed to visit points in parameter space in proportion to their probabilities under the target distribution. The "Markov" part means that the probability of reaching a point $\boldsymbol{\theta}^t$ at step t of the chain depends only the previous point $\boldsymbol{\theta}^{t-1}$ and not any earlier points in the chain.

2.1 Metropolis and Metropolis-Hastings Algorithms

2.1.1 Metropolis Algorithm

To carry out an MCMC we need a rule for moving from one point to another in parameter space. Typically there is a proposal distribution $J(\boldsymbol{\theta}'|\boldsymbol{\theta})$ from which a jump is considered, and then a rule for deciding whether to jump to that new point. The Metropolis algorithm requires that the rule be symmetric, so $J(\boldsymbol{\theta}'|\boldsymbol{\theta}) = J(\boldsymbol{\theta}|\boldsymbol{\theta}')$ but doesn't otherwise restrict it.[1] The rule is then as follows: draw a point $\boldsymbol{\theta}^*$ from $J(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{t-1})$ and calculate the ratio

$$r = \frac{p(\boldsymbol{\theta}^*|\mathbf{y}, I)}{p(\boldsymbol{\theta}^{t-1}|\mathbf{y}, I)} = \frac{q(\boldsymbol{\theta}^*)}{q(\boldsymbol{\theta}^{t-1})} \ ; \tag{2.1}$$

If $r \geq 1$, make the jump and let $\boldsymbol{\theta}^t = \boldsymbol{\theta}^*$. If $r < 1$, generate a Uniform$[0, 1]$ random number u; if $u \leq r$, make the jump and $\boldsymbol{\theta}^t = \boldsymbol{\theta}^*$. If $u > r$, don't make the jump and let $\boldsymbol{\theta}^t = \boldsymbol{\theta}^{t-1}$. I.e., make the jump with probability r.

We'll see in detail why it works on Thursday, but as a matter of formalism consider how the rules translate into statements about the probability distributions for the proposed jump position $\boldsymbol{\theta}^*$ and the next value $\boldsymbol{\theta}^t$ in terms of the current value $\boldsymbol{\theta}^{t-1}$. The proposal rule just tells us that

$$p(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{t-1}, \mathbf{y}, I) = J(\boldsymbol{\theta}^*|\boldsymbol{\theta}) \tag{2.2}$$

while the acceptance rule tells us that

$$p(\boldsymbol{\theta}^t|\boldsymbol{\theta}^*, \boldsymbol{\theta}^{t-1}, \mathbf{y}, I) = \begin{cases} \delta(\boldsymbol{\theta}^t - \boldsymbol{\theta}^*) & \text{if } r \geq 1 \\ r\delta(\boldsymbol{\theta}^t - \boldsymbol{\theta}^*) + (1-r)\delta(\boldsymbol{\theta}^t - \boldsymbol{\theta}^{t-1}) & \text{if } r < 1 \end{cases} \tag{2.3}$$

[1] $J(\boldsymbol{\theta}'|\boldsymbol{\theta})$ should be a probability distribution which we can easily draw from for any $\boldsymbol{\theta}$ with $p(\boldsymbol{\theta}|\mathbf{y}, I) > 0$.

where $\delta(\boldsymbol{\theta} - \boldsymbol{\theta}')$ is the Dirac delta function defined so that

$$\int \delta(\boldsymbol{\theta} - \boldsymbol{\theta}')\, f(\boldsymbol{\theta})\, d\boldsymbol{\theta} = f(\boldsymbol{\theta}') \tag{2.4}$$

and r is the ratio defined in (2.1). Note that if $r < 1$, $p(\boldsymbol{\theta}^t|\boldsymbol{\theta}^*, \boldsymbol{\theta}^{t-1}, \mathbf{y}, I)$ is a *mixture distribution*, which is just a linear combination of other probability distributions (in this case degenerate ones). It's basically just a manifestation of the sum rule. If I says that $\boldsymbol{\theta}$ can be drawn from distributions D_1 or D_2, the probability distribution is

$$p(\boldsymbol{\theta}|I) = p(\boldsymbol{\theta}|D_1, I)\ \Pr(D_1|I) + p(\boldsymbol{\theta}|D_2, I)\ \Pr(D_2|I) \tag{2.5}$$

where $p(\boldsymbol{\theta}|D_1, I)$ and $p(\boldsymbol{\theta}|D_2, I)$ are separately normalized probability distributions, and $\Pr(D_1|I) + \Pr(D_2|I) = 1$.

Returning to the distribution (2.3) we can actually combine the two cases by writing

$$p(\boldsymbol{\theta}^t|\boldsymbol{\theta}^*, \boldsymbol{\theta}^{t-1}, \mathbf{y}, I) = \min(1, r)\delta(\boldsymbol{\theta}^t - \boldsymbol{\theta}^*) + [1 - \min(1, r)]\delta(\boldsymbol{\theta}^t - \boldsymbol{\theta}^{t-1}) \tag{2.6}$$

The product rule then gives the joint distribution

$$\begin{aligned} p(\boldsymbol{\theta}^t, \boldsymbol{\theta}^*|\boldsymbol{\theta}^{t-1}, \mathbf{y}, I) = J(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{t-1}) \Big(& \min(1, r)\delta(\boldsymbol{\theta}^t - \boldsymbol{\theta}^*) \\ & + [1 - \min(1, r)]\delta(\boldsymbol{\theta}^t - \boldsymbol{\theta}^{t-1}) \Big) \end{aligned} \tag{2.7}$$

We will show that the target distribution is the stable endpoint of the MCMC by showing that if the marginal sampling distribution for one step is the target distribution, than the marginal sampling distribution for the next step is as well:

$$p(\boldsymbol{\theta}^{t-1}|\text{MCMC}) = p(\boldsymbol{\theta}^{t-1}|\mathbf{y}, I) \quad \text{implies} \quad p(\boldsymbol{\theta}^t|\text{MCMC}) = p(\boldsymbol{\theta}^t|\mathbf{y}, I) \tag{2.8}$$

Thursday 23 March 2017

2.1.2 Why It Works

Now we turn to a demonstration of why the Metropolis algorithm produces, over the long term, samples from the target distribution. As a result of some mathematical theory that's beyond the scope of this course, any reasonable MCMC will settle down into a unique equilibrium, so all we need is to demonstrate that the target distribution is an equilibrium state. What that means is that if we (hypothetically) make a draw from the target distribution, and then take one MCMC step from that point and consider where we ended up, the probability distribution for the new point is also the target distribution. In the notation we've been using, this means that if we assume $p(\boldsymbol{\theta}^{t-1}|\text{MCMC}) = p(\boldsymbol{\theta}^{t-1}|\mathbf{y}, I)$ we can show that $p(\boldsymbol{\theta}^{t}|\text{MCMC}) = p(\boldsymbol{\theta}^{t}|\mathbf{y}, I)$. We'll do this by constructing the joint distribution $p(\boldsymbol{\theta}^{t}, \boldsymbol{\theta}^{t-1}|\text{MCMC})$ for the two points. Note that we're assuming

$$\int p(\boldsymbol{\theta}^{t}, \boldsymbol{\theta}^{t-1}|\text{MCMC}) = p(\boldsymbol{\theta}^{t-1}|\text{MCMC}) = p(\boldsymbol{\theta}^{t-1}|\mathbf{y}, I) \quad (2.9)$$

so if the functional form of $p(\boldsymbol{\theta}^{t}, \boldsymbol{\theta}^{t-1}|\text{MCMC})$ is symmetric under interchange of the two arguments $\boldsymbol{\theta}^{t}$ and $\boldsymbol{\theta}^{t-1}$, it will be the case that the two marginal distributions $p(\boldsymbol{\theta}^{t}|\text{MCMC})$ and $p(\boldsymbol{\theta}^{t-1}|\text{MCMC})$ have the same form, and thus we'll have proved that $p(\boldsymbol{\theta}^{t}|\text{MCMC}) = p(\boldsymbol{\theta}^{t}|\mathbf{y}, I)$.

To construct the joint distribution $p(\boldsymbol{\theta}^{t}, \boldsymbol{\theta}^{t-1}|\text{MCMC})$ we start with the joint distribution

$$\begin{aligned} p(\boldsymbol{\theta}^{t}, \boldsymbol{\theta}^{*}, \boldsymbol{\theta}^{t-1}|\text{MCMC}) &= p(\boldsymbol{\theta}^{t-1}|\mathbf{y}, I) J(\boldsymbol{\theta}^{*}|\boldsymbol{\theta}^{t-1}) \Big(\min(1, r)\delta(\boldsymbol{\theta}^{t} - \boldsymbol{\theta}^{*}) \\ &\quad + \max(1-r, 0)\delta(\boldsymbol{\theta}^{t} - \boldsymbol{\theta}^{t-1}) \Big) \\ &= \delta(\boldsymbol{\theta}^{t} - \boldsymbol{\theta}^{*}) J(\boldsymbol{\theta}^{*}|\boldsymbol{\theta}^{t-1}) \min(p(\boldsymbol{\theta}^{t-1}|\mathbf{y}, I), p(\boldsymbol{\theta}^{*}|\mathbf{y}, I)) \\ &\quad + \delta(\boldsymbol{\theta}^{t} - \boldsymbol{\theta}^{t-1}) J(\boldsymbol{\theta}^{*}|\boldsymbol{\theta}^{t-1}) \max(p(\boldsymbol{\theta}^{t-1}|\mathbf{y}, I) - p(\boldsymbol{\theta}^{*}|\mathbf{y}, I), 0) \end{aligned} \quad (2.10)$$

Next we marginalize over $\boldsymbol{\theta}^{*}$; in the first term, the delta function just sets $\boldsymbol{\theta}^{*}$ to $\boldsymbol{\theta}^{t}$. The second term is more complicated, but the result of the integral (factoring out the $\boldsymbol{\theta}^{*}$-independent delta function) is

$$\begin{aligned} &\int J(\boldsymbol{\theta}^{*}|\boldsymbol{\theta}^{t-1}) \max(p(\boldsymbol{\theta}^{t-1}|\mathbf{y}, I) - p(\boldsymbol{\theta}^{*}|\mathbf{y}, I), 0)\, d\boldsymbol{\theta}^{*} \\ &= \int\limits_{p(\boldsymbol{\theta}^{*}|\mathbf{y},I)<p(\boldsymbol{\theta}^{t-1}|\mathbf{y},I)} J(\boldsymbol{\theta}^{*}|\boldsymbol{\theta}^{t-1}) \left[p(\boldsymbol{\theta}^{t-1}|\mathbf{y}, I) - p(\boldsymbol{\theta}^{*}|\mathbf{y}, I) \right] d\boldsymbol{\theta}^{*} \\ &= p(\boldsymbol{\theta}^{t-1}, \text{reject}|\mathbf{y}, I) \end{aligned} \quad (2.11)$$

I.e., it's the probability of drawing $\boldsymbol{\theta}^{t-1}$ from the target distribution and then rejecting the next jump. So the result of the marginalization is

$$\begin{aligned} p(\boldsymbol{\theta}^{t}, \boldsymbol{\theta}^{t-1}|\text{MCMC}) &= J(\boldsymbol{\theta}^{t}|\boldsymbol{\theta}^{t-1}) \min(p(\boldsymbol{\theta}^{t-1}|\mathbf{y}, I), p(\boldsymbol{\theta}^{t}|\mathbf{y}, I)) \\ &\quad + \delta(\boldsymbol{\theta}^{t} - \boldsymbol{\theta}^{t-1}) p(\boldsymbol{\theta}^{t-1}, \text{reject}|\mathbf{y}, I) \end{aligned} \quad (2.12)$$

The first term is symmetric under interchange $\boldsymbol{\theta}^{t} \longleftrightarrow \boldsymbol{\theta}^{t-1}$ as long as the proposal distribution $J(\boldsymbol{\theta}^{t}|\boldsymbol{\theta}^{t-1})$ is. The second term is also symmetric, because the Dirac delta function is even, and is only non-zero if $\boldsymbol{\theta}^{t} = \boldsymbol{\theta}^{t-1}$. Thus the joint distribution is symmetric and both marginal distributions are the same, i.e., the target distribution.

2.1.3 Metropolis-Hastings Algorithm

The Metropolis algorithm can be extended to situations where the proposal distribution is not symmetric, e.g., distributions which avoid boundaries of parameter space. The modification is to the acceptance rule, which now uses the value of the ratio

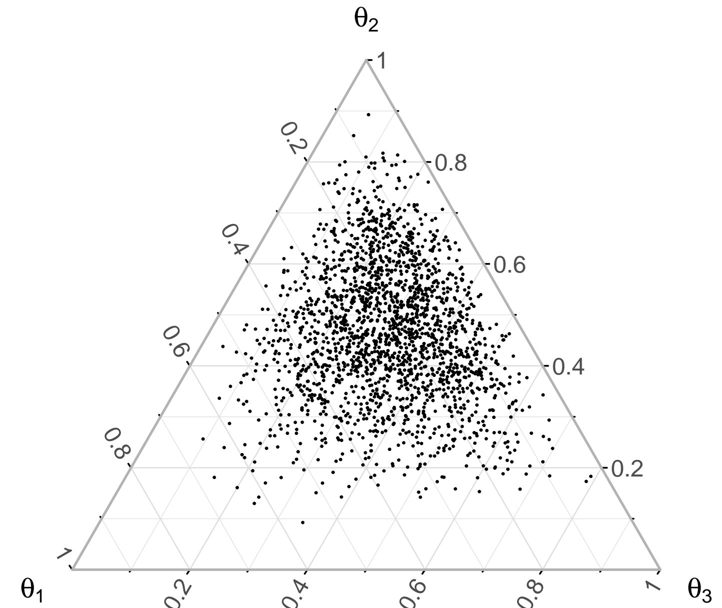
$$r' = \frac{p(\boldsymbol{\theta}^*|\mathbf{y},I)\,J(\boldsymbol{\theta}^{t-1}|\boldsymbol{\theta}^*)}{p(\boldsymbol{\theta}^{t-1}|\mathbf{y},I)\,J(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{t-1})} = \frac{q(\boldsymbol{\theta}^*)\,J(\boldsymbol{\theta}^{t-1}|\boldsymbol{\theta}^*)}{q(\boldsymbol{\theta}^{t-1})\,J(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{t-1})}\ ; \tag{2.13}$$

the rule is as before, i.e., accept the jump with probability $\min(1, r')$.

2.1.4 Dirichlet Example

As a concrete example, suppose the target distribution is a Dirichlet distribution with parameters $(\alpha_1, \alpha_2, \alpha_3) = (3, 6, 4)$. Of course, we can sample from this directly, and plot it on a ternary plot:

```
> alpha = c(3,6,4)
> N = 2000
> set.seed(20170321)
> thetasample = rdirichlet(N,alpha)
> library(ggtern)
> myframe = function(mysample) {
+     ( data.frame(theta1=mysample[,1],
+                  theta2=mysample[,2],
+                  theta3=mysample[,3]) )
+ }
> tickvals = seq(.2,1,.2)
> addmyopts = function(myplot) {
+     ( myplot
+       + geom_point(size=0.01)
+       + tern_limits(breaks=tickvals,labels=tickvals)
+       + xlab(expression(theta[1]))
+       + ylab(expression(theta[2]))
+       + zlab(expression(theta[3]))
+       + theme_light()
+     )
+ }
> thetaframe = myframe(rdirichlet(N,alpha))
> addmyopts(ggtern(thetaframe,aes(theta1,theta2,theta3)))
```



But let's pretend we can't just sample directly from the posterior. We'll define a function that does one step of a Metropolis MCMC by proposing a new point with independent Gaussian changes to θ_1 and θ_2:

$$J(\theta^*|\theta) \propto \exp\left(-\frac{(\theta_1^* - \theta_1)^2}{2\sigma^2} - \frac{(\theta_2^* - \theta_2)^2}{2\sigma^2}\right) \tag{2.14}$$

σ defaults to 0.2 but we leave it as an argument so we can change it later. Note that the proposed point may be out of

bounds (negative θ_1^* or θ_2^*, or $\theta_1^* + \theta_2^* > 1$), but the pdf should be zero outside of the allowed space.

```
> nextpt = function (mytheta,sigma=0.2) {
+     theta1new = rnorm(1,mytheta[1],sigma)
+     theta2new = rnorm(1,mytheta[2],sigma)
+     theta3new = 1 - theta1new - theta2new
+     thetanew = c(theta1new,theta2new,theta3new)
+     r = ( ddirichlet(thetanew,alpha)
+           / ddirichlet(mytheta,alpha) )
+     if ( runif(1) > r ) {
+         thetanew = mytheta
+     }
+     return(thetanew)
+ }
```

We'll start in the middle of the triangle, and update one step at a time:

```
> theta = c(1/3,1/3,1/3)
> theta = nextpt(theta); print(theta)
[1] 0.2233426 0.4357120 0.3409454
```

We've accepted the first jump.

```
> theta = nextpt(theta); print(theta)
[1] 0.2233426 0.4357120 0.3409454
> theta = nextpt(theta); print(theta)
[1] 0.2233426 0.4357120 0.3409454
```

The last two jumps have been rejected.

```
> theta = nextpt(theta); print(theta)
Warning message:
In log(x) : NaNs produced
[1] 0.2233426 0.4357120 0.3409454
```

We tried to jump out of bounds, so it was rejected.

```
> theta = nextpt(theta); print(theta)
[1] 0.3003148 0.4433981 0.2562871
```

We've accepted a jump.

```
> theta = nextpt(theta); print(theta)
[1] 0.3003148 0.4433981 0.2562871
```

We've rejected a jump and stayed at the same point, etc.

Now let's take a chain of 4000 steps:

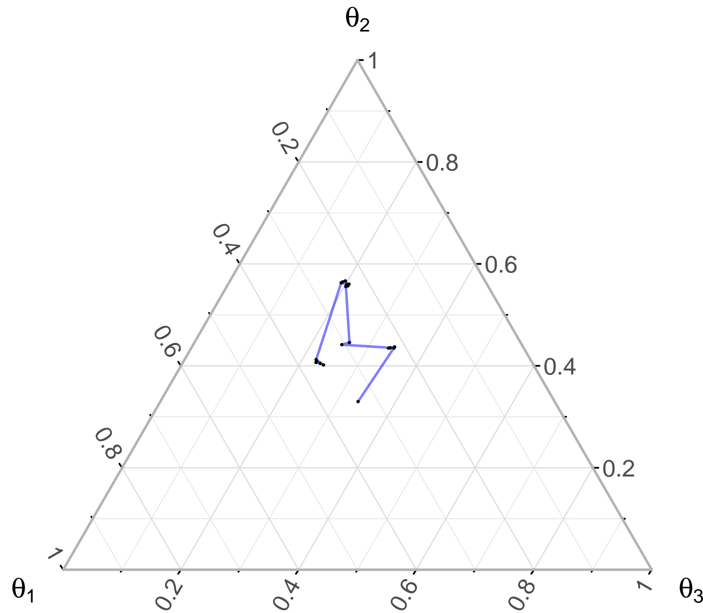
```
> m=4000
> nanchain = matrix(rep(0/0,3*m),ncol=3,byrow=TRUE)
> chain1 = nanchain
> chain1[1,] = c(1/3,1/3,1/3)
> for (i in 2:m) {
+     chain1[i,] = nextpt(chain1[i-1,])
+ }
There were 50 or more warnings (use warnings() to see
 the first 50)
```

The warnings are for more illegal jumps, which we know about. Now we'll plot the chains on a ternary plot. But since the MCMC method allows us to remain at the same point for several steps, we should "jitter" the points slightly so they're plotted next to each other rather than right on top of each other:

```
> myjitter = matrix(runif(3*m,-0.01,0.01),
+                   ncol=3,byrow=TRUE)
> chain1frame = myframe(chain1+myjitter)
```

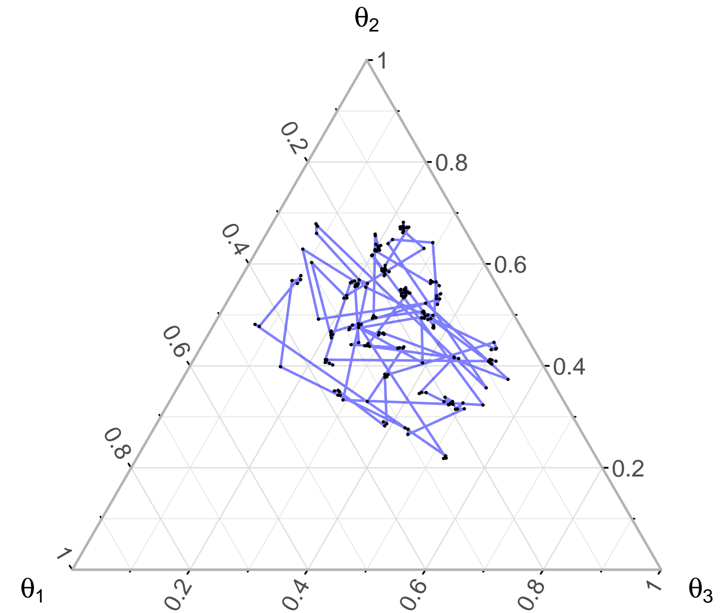
Let's plot the first 20 steps of the chain

```
> addmyopts(ggtern(chain1frame[1:20,],
+                  aes(theta1,theta2,theta3))
+           + geom_path(alpha=0.5,col='blue'))
```



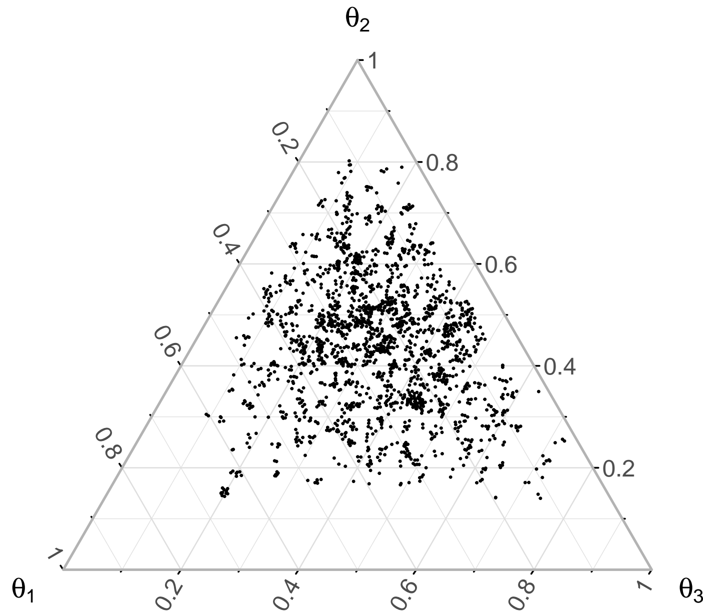
and now the first 200:

```
> addmyopts(ggtern(chain1frame[1:200,],
+                  aes(theta1,theta2,theta3))
+            + geom_path(alpha=0.5,col='blue'))
```



The first part of the chain is of course influenced by where we started it, but it gradually becomes less important, and the chain looks more like a set of correlated draws from the target distribution. It's common practice to discard the first part of the chain, known as "burn-in" (Gelman calls this "warm-up"). Gelman recommends the first half of the chain, which is probably overkill, but let's do that, and plot the result:

```
> addmyopts(ggtern(chain1frame[(m/2+1):m,],
+                  aes(theta1,theta2,theta3)))
```

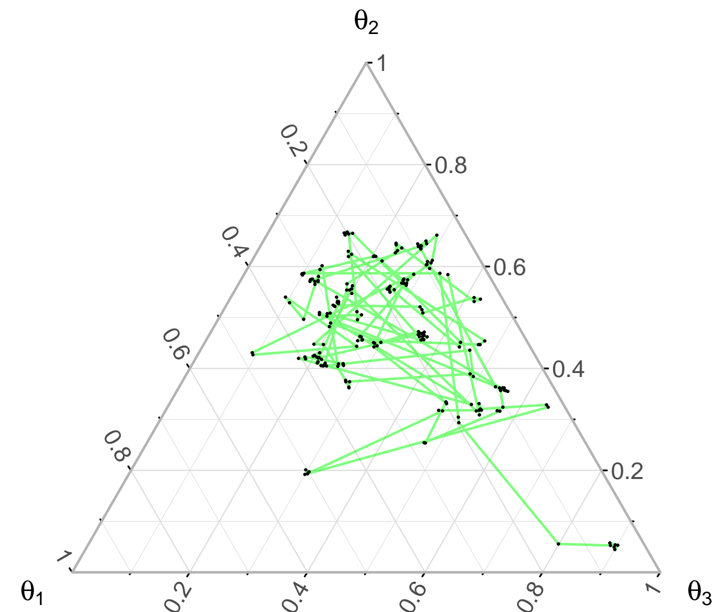


We can see tat this sort of follows the Dirichlet distribution, but it's a bit "clumpier". The correlations show that this does not look like 2000 independent draws from the distribution. Gelman has some more precise effective-sample-size computations, but they're non-trivial, since they involve estimating the autocorrelation function from the data.

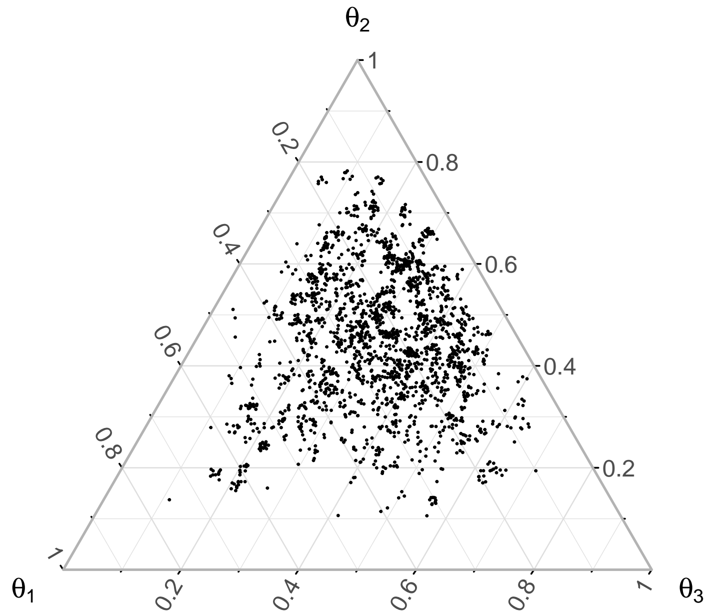
We can start the chains in different parts of the parameter space and see that it makes a difference at the start, but not in the long run:

```
> chain2 = nanchain
> chain2[1,] = c(0.05,0.05,0.9)
> for (i in 2:m) {
+     chain2[i,] = nextpt(chain2[i-1,])
+ }
There were 50 or more warnings (use warnings() to see
 the first 50)
```

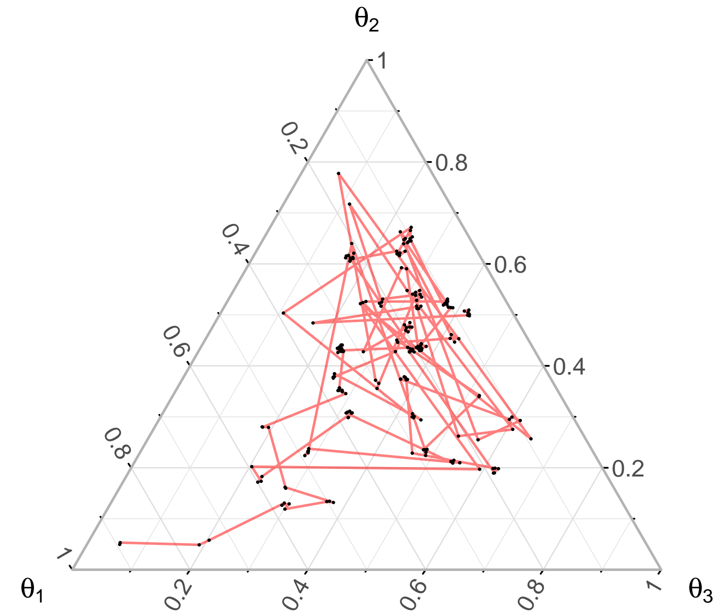
```
> chain2frame = myframe(chain2+myjitter)
> addmyopts(ggtern(chain2frame[1:200,],
+                   aes(theta1,theta2,theta3))
+            + geom_path(alpha=0.5,col='green'))
```



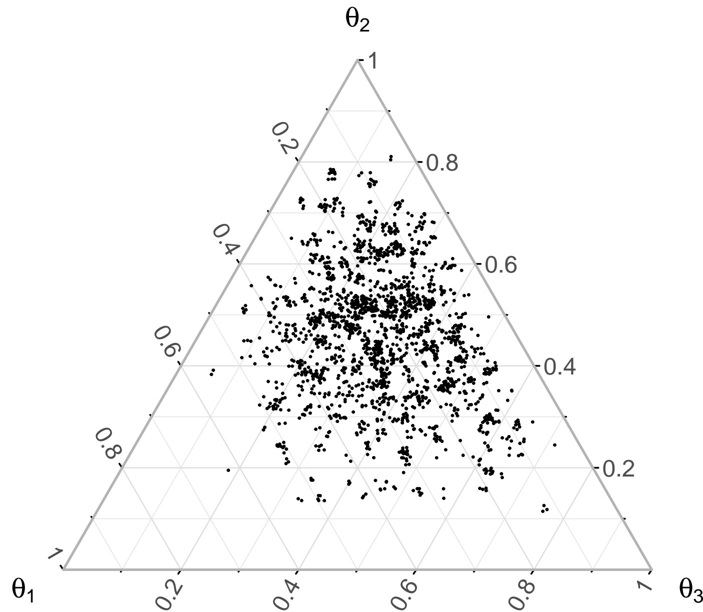

```
> addmyopts(ggtern(chain2frame[(m/2+1):m,],
+                   aes(theta1,theta2,theta3)))
```




```
> chain3 = nanchain
> chain3[1,] = c(0.05,0.05,0.9)
> for (i in 2:m) {
+     chain3[i,] = nextpt(chain3[i-1,])
+ }
There were 50 or more warnings (use warnings() to see
 the first 50)
> chain3frame = myframe(chain3+myjitter)
> addmyopts(ggtern(chain3frame[1:200,],
+                   aes(theta1,theta2,theta3))
+            + geom_path(alpha=0.5,col='red'))
```

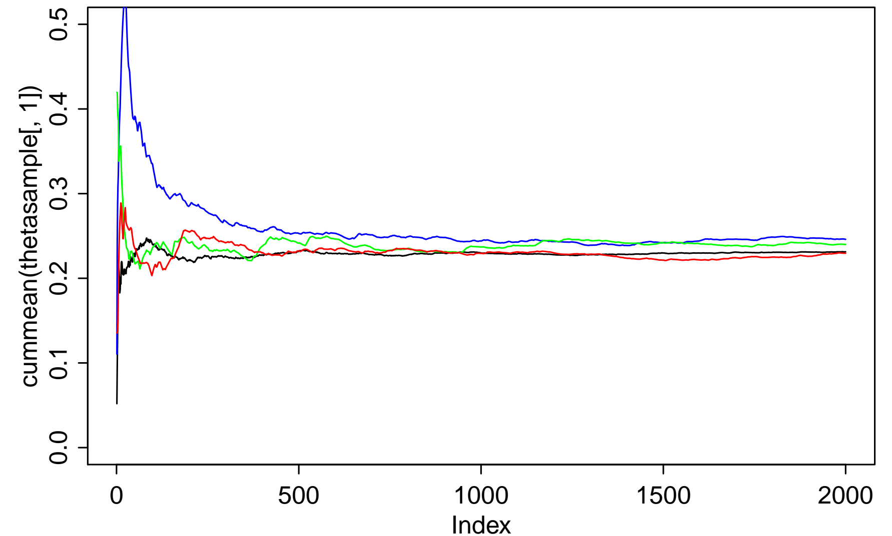



```
> addmyopts(ggtern(chain3frame[(m/2+1):m,],
+                    aes(theta1,theta2,theta3)))
```



To give an example of the sort of things we can calculate with these MCMC results, define a "cumulative mean" function to estimate $E(\theta_1|\mathbf{y}, I)$ from the sample, and compare how it looks for different chains:

```
> cummean = function(x) {return(cumsum(x)/1:length(x))}
> plot(cummean(thetasample[,1]),type='l',ylim=c(0,0.5))
> lines(cummean(chain1frame[(m/2+1):m,1]),col='blue')
> lines(cummean(chain2frame[(m/2+1):m,1]),col='green')
> lines(cummean(chain3frame[(m/2+1):m,1]),col='red')
```

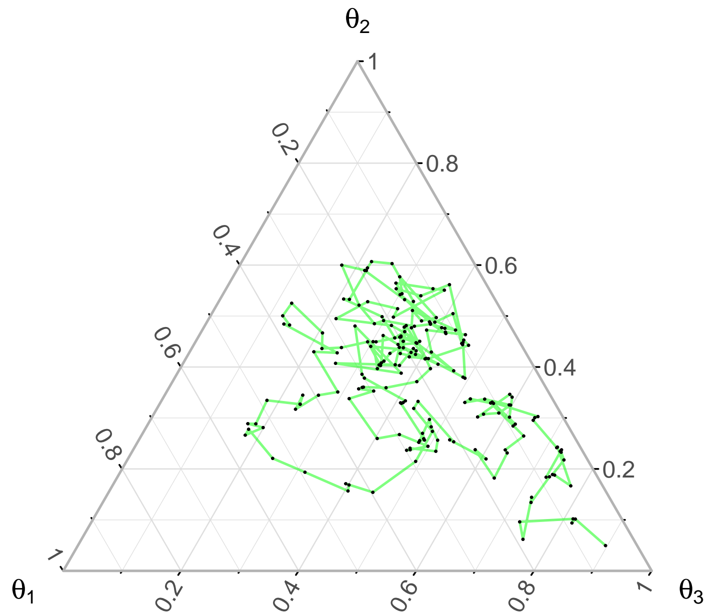


We can see that the MCMC samples take longer to provide an accurate expectation value than the independent sample would.

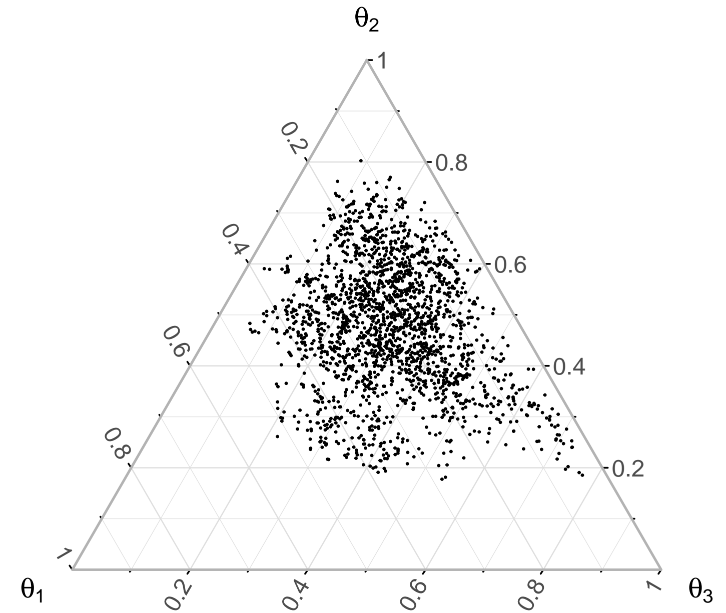
One of the choices we have to make in doing a Metropolis (or Metropolis-Hastings) MCMC is the proposal distribution. It ought to involve jumps which are comparable in size to the features of the distribution. If the jumps are too small, it will take the chains a long time to explore the parameter space, and if they are too large, most of them will be rejected, also leading to slow movement of the chains. For comparison to the results above, suppose we'd used $\sigma = 0.04$ instead of $\sigma = 0.2$:

```
> chain04 = nanchain
> chain04[1,] = c(0.05,0.05,0.9)
> for (i in 2:m) {
+     chain04[i,] = nextpt(chain04[i-1,],0.04)
+ }
There were 44 warnings (use warnings() to see them)
> chain04frame = myframe(chain04+myjitter)
> addmyopts(ggtern(chain04frame[1:200,],
```

```
+                  aes(theta1,theta2,theta3))
+           + geom_path(alpha=0.5,col='green'))
```

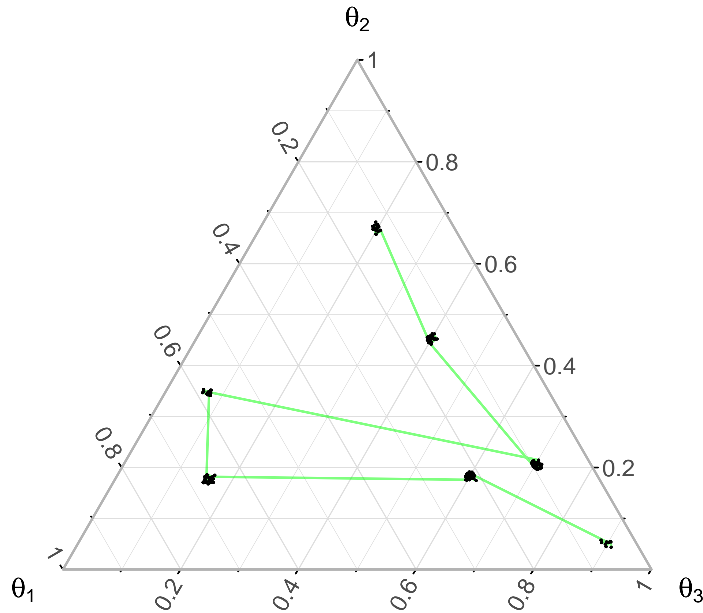



```
> addmyopts(ggtern(chain04frame[(m/2+1):m,],
+                  aes(theta1,theta2,theta3)))
```



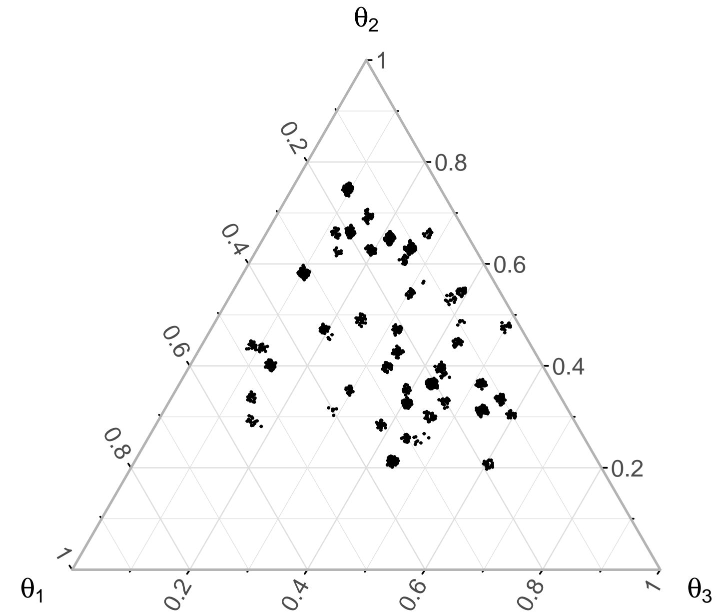
Or if we'd used $\sigma = 1$ instead of $\sigma = 0.2$:

```
> chain10 = nanchain
> chain10[1,] = c(0.05,0.05,0.9)
> for (i in 2:m) {
+     chain10[i,] = nextpt(chain10[i-1,],1)
+ }
There were 50 or more warnings (use warnings() to see
 the first 50)
> chain10frame = myframe(chain10+myjitter)
> addmyopts(ggtern(chain10frame[1:200,],
+                  aes(theta1,theta2,theta3))
+           + geom_path(alpha=0.5,col='green'))
```



Here we see the impact of all the rejected jumps is particularly bad, and the first 200 points represent only 7 unique points in parameter space.

```
> addmyopts(ggtern(chain10frame[(m/2+1):m,],
+                  aes(theta1,theta2,theta3)))
```



Tuesday 28 March 2017

2.2 Gibbs Sampler

Recall the Metropolis-Hastings algorithm, which produces a Markov Chain resulting in samples from a posterior $p(\boldsymbol{\theta}|\mathbf{y},I)$ by proposing jumps drawn from a proposal distribution $J(\boldsymbol{\theta}^*|\boldsymbol{\theta})$. If we've reached a point $\boldsymbol{\theta}^{t-1}$ on the chain, we propose a jump to a new point θ^* drawn from $J(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{t-1})$ and either accept ($\theta^t=\theta^*$) or reject ($\theta^t=\theta^{t-1}$) according to the value of the ratio

$$r=\frac{p(\boldsymbol{\theta}^*|\mathbf{y},I)\,J(\boldsymbol{\theta}^{t-1}|\boldsymbol{\theta}^*)}{p(\boldsymbol{\theta}^{t-1}|\mathbf{y},I)\,J(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{t-1})} \tag{2.15}$$

If $r\geq 1$, we automatically accept the jump; otherwise we accept it with probability r.

The Gibbs sampler is a special example of Metropolis-Hastings, where the proposal distribution is constructed

from the target distribution itself. For each parameter θ_j we construct a vector of the other parameters, $\boldsymbol{\theta}_{-j} = \{\theta_1, \ldots, \theta_{j-1}, \theta_{j+1}, \ldots \theta_m\}$ and consider the conditional distribution

$$p(\theta_j|\boldsymbol{\theta}_{-j}, \mathbf{y}, I) = \frac{p(\boldsymbol{\theta}|\mathbf{y}, I)}{p(\boldsymbol{\theta}_{-j}|\mathbf{y}, I)} \ . \tag{2.16}$$

If the conditional distribution for each individual parameter is simple, we can use that as our proposal distribution. In particular, at each step, we choose[2] one parameter θ_j to update and leave the others constant, so that

$$J(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{t-1}) = p(\theta_j^*|\boldsymbol{\theta}_{-j}^{t-1}, \mathbf{y}, I)\, \delta(\boldsymbol{\theta}_{-j}^* - \boldsymbol{\theta}_{-j}^{t-1}) \tag{2.17}$$

The Metropolis-Hastings ratio is

$$\begin{aligned} r &= \frac{p(\boldsymbol{\theta}^*|\mathbf{y}, I)\, J(\boldsymbol{\theta}^{t-1}|\boldsymbol{\theta}^*)}{p(\boldsymbol{\theta}^{t-1}|\mathbf{y}, I)\, J(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{t-1})} \\ &= \frac{p(\boldsymbol{\theta}^*|\mathbf{y}, I)\, p(\theta_j^{t-1}|\boldsymbol{\theta}_{-j}^*, \mathbf{y}, I)\, \delta(\boldsymbol{\theta}_{-j}^* - \boldsymbol{\theta}_{-j}^{t-1})}{p(\boldsymbol{\theta}^{t-1}|\mathbf{y}, I)\ p(\theta_j^*|\boldsymbol{\theta}_{-j}^{t-1}, \mathbf{y}, I)\, \delta(\boldsymbol{\theta}_{-j}^* - \boldsymbol{\theta}_{-j}^{t-1})} \\ &= \frac{\cancel{p(\boldsymbol{\theta}^*|\mathbf{y}, I)}\, \cancel{p(\boldsymbol{\theta}^{t-1}|\mathbf{y}, I)}\, p(\boldsymbol{\theta}_{-j}^*|\mathbf{y}, I)\, \delta(\boldsymbol{\theta}_{-j}^* - \boldsymbol{\theta}_{-j}^{t-1})}{\cancel{p(\boldsymbol{\theta}^{t-1}|\mathbf{y}, I)}\, p(\boldsymbol{\theta}_{-j}^{t-1}|\mathbf{y}, I)\, \cancel{p(\boldsymbol{\theta}^*|\mathbf{y}, I)}\, \delta(\boldsymbol{\theta}_{-j}^* - \boldsymbol{\theta}_{-j}^{t-1})} = 1 \end{aligned} \tag{2.18}$$

so the jump is always accepted. Note that if we can't actually draw from $p(\theta_j^*|\boldsymbol{\theta}_{-j}^{t-1}, \mathbf{y}, I)$ exactly, we may be able to use an approximation to the marginal distribution, and then calculate the M-H ratio rather than automatically accepting every jump.

[2] A common practice is to group together m steps at a time, and loop through the m parameters in some random order, updating each one.

2.2.1 Dirichlet Example

Let's return to our example of sampling from a Dirichlet posterior:

$$p(\theta_1, \theta_2|\mathbf{y}, I) \propto \theta_1^{\alpha_1}\theta_2^{\alpha_2}(1-\theta_1-\theta_2)^{\alpha_3} \qquad 0 < \theta_1, \theta_2;\ \theta_1 + \theta_2 < 1 \tag{2.19}$$

If we want to use the Gibbs sampler, we need the conditional distributions $p(\theta_1|\theta_2, \mathbf{y}, I)$ and $p(\theta_2|\theta_1, \mathbf{y}, I)$. The marginal distributions of a Dirichlet distribution are beta distributions:

$$p(\theta_1|\mathbf{y}, I) \propto \theta_1^{\alpha_1-1}(1-\theta_1)^{\alpha_2+\alpha_3-1} \qquad 0 < \theta_1 < 1 \tag{2.20a}$$

$$p(\theta_2|\mathbf{y}, I) \propto \theta_2^{\alpha_2-1}(1-\theta_2)^{\alpha_1+\alpha_3-1} \qquad 0 < \theta_2 < 1 \tag{2.20b}$$

so we get the conditional distribution

$$\begin{aligned} p(\theta_1|\theta_2, \mathbf{y}, I) = \frac{p(\theta_1, \theta_2|\mathbf{y}, I)}{p(\theta_2|\mathbf{y}, I)} &\propto \frac{\theta_1^{\alpha_1-1}\cancel{\theta_2^{\alpha_2-1}}(1-\theta_1-\theta_2)^{\alpha_3-1}}{\cancel{\theta_2^{\alpha_2-1}}(1-\theta_2)^{\alpha_1+\alpha_3-1}} \\ &= (1-\theta_2)\left(\frac{\theta_1}{1-\theta_2}\right)^{\alpha_1-1}\left(1-\frac{\theta_1}{1-\theta_2}\right)^{\alpha_3-1} \\ &\propto \left(\frac{\theta_1}{1-\theta_2}\right)^{\alpha_1-1}\left(1-\frac{\theta_1}{1-\theta_2}\right)^{\alpha_3-1} \qquad 0 < \theta_1 < 1-\theta_2 \end{aligned} \tag{2.21}$$

(since the extra factor of $1-\theta_2$ is a "constant" when considering a probability density for θ_1) and likewise

$$p(\theta_2|\theta_1, \mathbf{y}, I) \propto \left(\frac{\theta_2}{1-\theta_1}\right)^{\alpha_2-1}\left(1-\frac{\theta_2}{1-\theta_1}\right)^{\alpha_3-1} \qquad 0 < \theta_2 < 1-\theta_1 \tag{2.22}$$

We can draw easily from the distributions (2.21) and (2.22). For instance, (2.21) tells us that

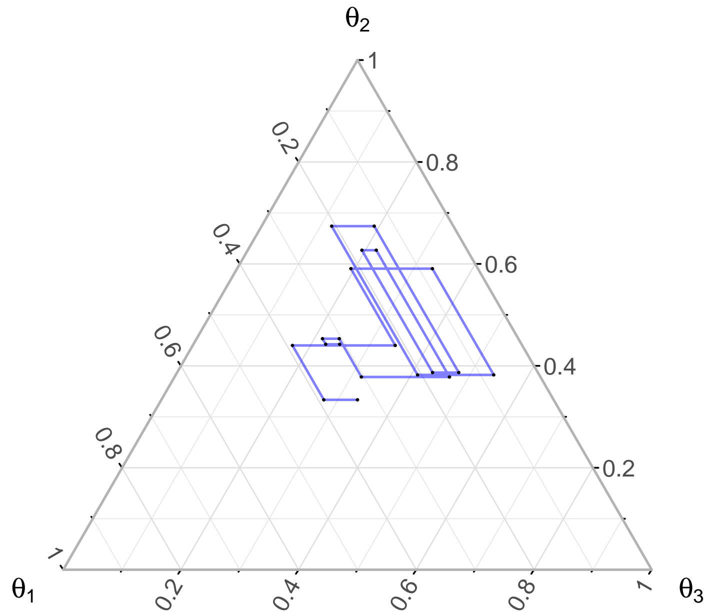
$$\left.\frac{\theta_1}{1-\theta_2}\right|\theta_2 \sim \text{Beta}(\alpha_1, \alpha_3) \tag{2.23}$$

This means, to update θ_1, we draw a Beta(α_1, α_3) random variable and multiply it by $1-\theta_2$. We can code this up in R (we're also keeping track of θ_3, but we just set it to $1-\theta_1-\theta_2$ at each Gibbs step:

```
> alpha = c(3,6,4)
> nextpt1_gibbs = function(mytheta) {
+     theta1new = (1-mytheta[2])*rbeta(1,alpha[1],alpha[3])
+     mytheta[1] = theta1new
+     mytheta[3] = 1 - sum(mytheta[-3])
+     return(mytheta)
+ }
> nextpt2_gibbs = function(mytheta) {
+     theta2new = (1-mytheta[1])*rbeta(1,alpha[2],alpha[3])
+     mytheta[2] = theta2new
+     mytheta[3] = 1 - sum(mytheta[-3])
+     return(mytheta)
+ }
> set.seed(20170328)
> theta = c(1/3,1/3,1/3)
> theta = nextpt1_gibbs(theta); print(theta)
[1] 0.2547338 0.3333333 0.4119328
> theta = nextpt2_gibbs(theta); print(theta)
[1] 0.2547338 0.3682150 0.3770512
> theta = nextpt1_gibbs(theta); print(theta)
[1] 0.3640820 0.3682150 0.2677031
> theta = nextpt2_gibbs(theta); print(theta)
[1] 0.3640820 0.4362305 0.1996875
```

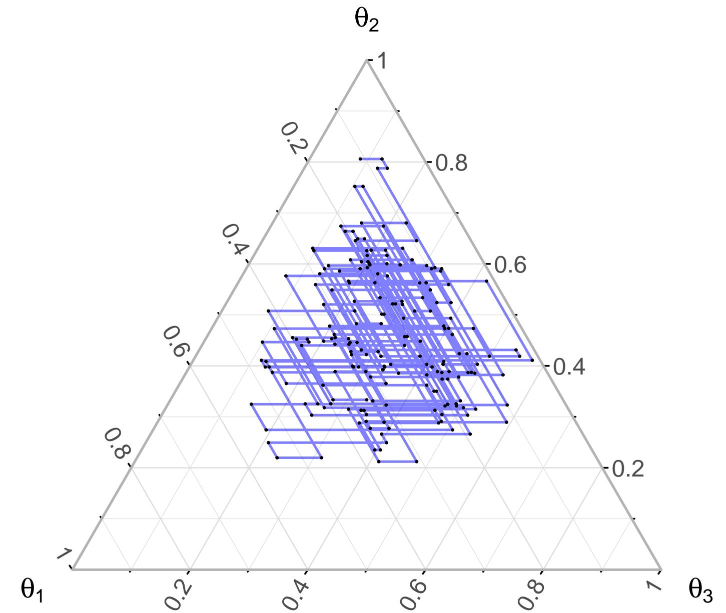
Let's start off a chain, For simplicity, we just alternate the steps in θ_1 and θ_2:

```
> m = 4000
> nanchain = matrix(rep(0/0,3*(m+1)),ncol=3,byrow=TRUE)
> chain1 = nanchain
> chain1[1,] = c(1/3,1/3,1/3)
> for (i in 2*(1:(m/2))) {
+     chain1[i,] = nextpt1_gibbs(chain1[i-1,])
+     chain1[i+1,] = nextpt2_gibbs(chain1[i,])
+ }
> myframe = function(mysample) {
+     ( data.frame(theta1=mysample[,1],
+                  theta2=mysample[,2],
+                  theta3=mysample[,3]) )
+ }
> chain1frame = myframe(chain1)
> library(ggtern)
> tickvals = seq(.2,1,.2)
> addmyopts = function(myplot) {
+     ( myplot
+       + geom_point(size=0.01)
+       + tern_limits(breaks=tickvals,labels=tickvals)
+       + xlab(expression(theta[1]))
+       + ylab(expression(theta[2]))
+       + zlab(expression(theta[3]))
+       + theme_light()
+     )
+ }
> addmyopts(ggtern(chain1frame[1:20,],
+                  aes(theta1,theta2,theta3))
+           + geom_path(alpha=0.5,col='blue'))
```



We can see that the Gibbs-sampler path looks rather different, being made up of steps parallel to the axes.

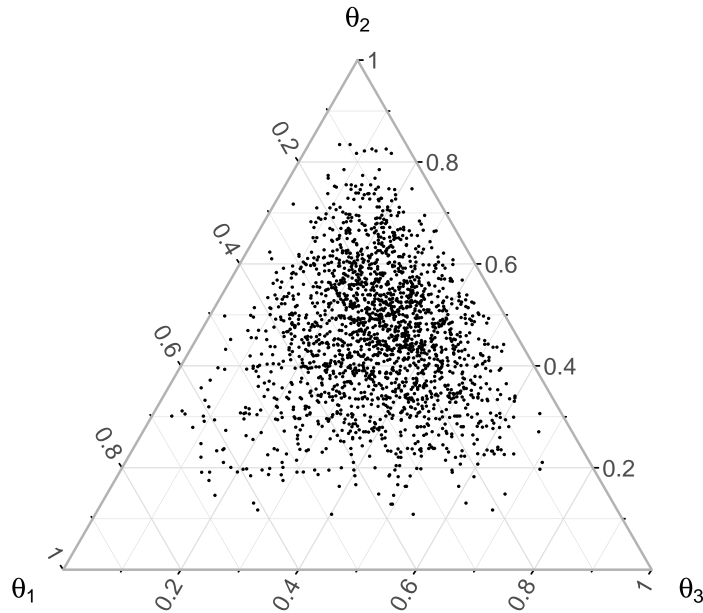
```
> addmyopts(ggtern(chain1frame[1:200,],
+                   aes(theta1,theta2,theta3))
+            + geom_path(alpha=0.5,col='blue'))
```



Within 200 steps, we've already explored the parameter space pretty well

```
> addmyopts(ggtern(chain1frame[(m/2+1):m,],
+                   aes(theta1,theta2,theta3)))
```

This looks like a pretty good sample. Note that no jitter is necessary now because no jumps are rejected.

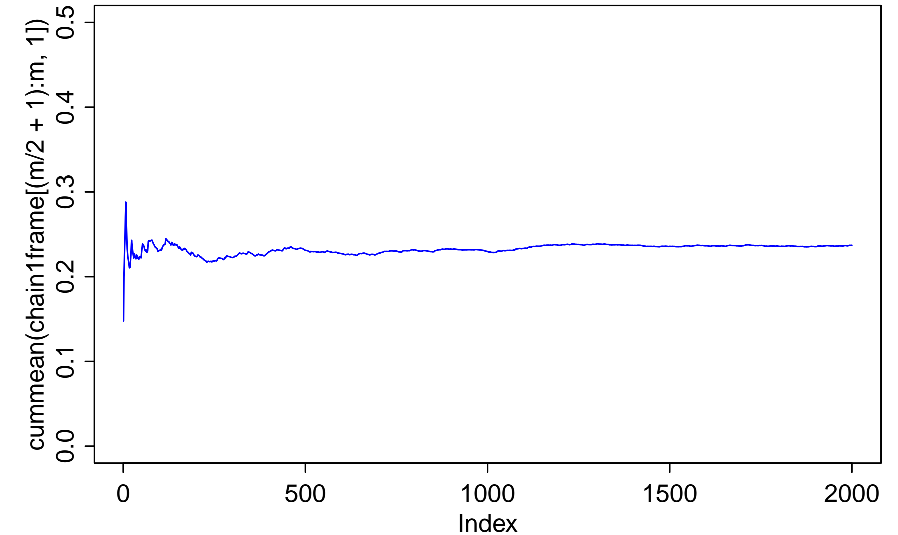


We can see how the means of θ_1, θ_2 and θ_3 estimated from these chains compare to the exact values:

```
> mean(chain1frame[(m/2+1):m,1])
[1] 0.2370543
> alpha[1]/sum(alpha)
[1] 0.2307692
> mean(chain1frame[(m/2+1):m,2])
[1] 0.4571373
> alpha[2]/sum(alpha)
[1] 0.4615385
> mean(chain1frame[(m/2+1):m,3])
[1] 0.3058084
> alpha[3]/sum(alpha)
[1] 0.3076923
```

We can plot the cumulative average, and see that a small sample can be more accurate for the Gibbs sampler than the MCMC we considered before:

```
> cummean = function(x) {return(cumsum(x)/1:length(x))}
> plot(cummean(chain1frame[,1]),type='l',col='blue',
+       ylim=c(0,0.5))
```



Thursday 30 March 2017 – Refer to Chapter 8 of Kruschke

2.3 Using JAGS for MCMC

Rather than coding up your own Gibbs sampler, you can also use libraries available for that purpose. A popular one is JAGS[3], which stands for Just Another Gibbs Sampler. It is a descendant of the BUGS (Bayesian Using Gibbs Sampler) family of programs. JAGS is actually its own language, but there exists an R interface known as `rjags`, which we'll use. To use JAGS, you don't typically specify the posterior directly, but instead specify the whole statistical model, in the form of the prior and

[3]http://mcmc-jags.sourceforge.net/

likelihood, and provide it with some data. The model is specified in a separate file written in JAGS's own language. It differs from R in a few potentially annoying ways:

- The names of some standard distributions are different, as is the order of their parameters; e.g., `dbin(p,n)` rather than `dbinom(n,p)`.
- Assignment must be done with the `<-` operator rather than `=`.
- The commands are not really executed in the order you write them (except inside a loop), and for example, you can't change the value of a variable using `<-`, once you've already set it.
- Not all of the R data structures are available.

It turns out that the Dirichlet distribution doesn't work so smoothly in JAGS, so for a minimal working example, we'll go back to our old standby of Bernoulli trials with a Beta prior. We have to write our model in a file; in practice we'd do that in a text editor, but for demonstration purposes let's write it from within R:

```
> library(rjags)
Loading required package: coda
Linked to JAGS 3.4.0
Loaded modules: basemod,bugs
> modelString = "
+ model {
+   y ~ dbin(theta,n)
+   theta ~ dbeta(1,1)
+   n <- 4
+ }
+ "
> filename = 'model.20170330.txt'
> writeLines( modelString, con=filename )
```

The ~ operator defines the distribution from which a variable is sampled. We've hard-coded $n = 4$ into the model to show that such things can be done, but we could also supply n with the data. We use rjags to initialize the model:

```
> set.seed(20170330)
> jagsModel = jags.model( file=filename, data=list(y=3),
+     n.chains=3 )
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 4

Initializing model

> jagsModel
JAGS model:


model {
  y ~ dbin(theta,n)
  theta ~ dbeta(1,1)
  n <- 4
}

Fully observed variables:
 n y
```

We've specified the number of chains to use; if you leave that out it uses one chain by default. There's another optional option `inits` to tell your chains where to start; if you leave it out as we did, JAGS picks the points for you.

The first thing we need to do is run the model for a while and discard the burn-in:

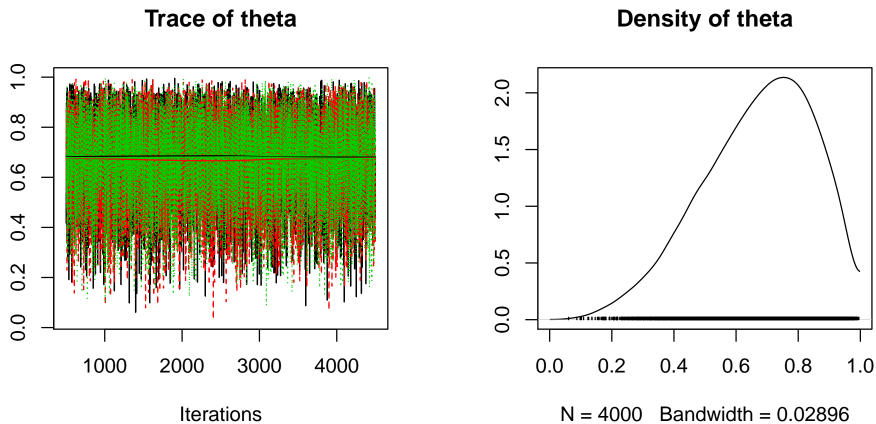
```
> update(jagsModel, n.iter=500)
  |**************************************************| 100%
```

Now we're ready to collect the samples. We could used `jags.samples`, but there's a convenience package called `coda` and we'll use the wrapper from that:

```
> codaSamples = coda.samples(jagsModel,
+      variable.names=c('theta'),n.iter=4000)
  |**************************************************| 100%
```

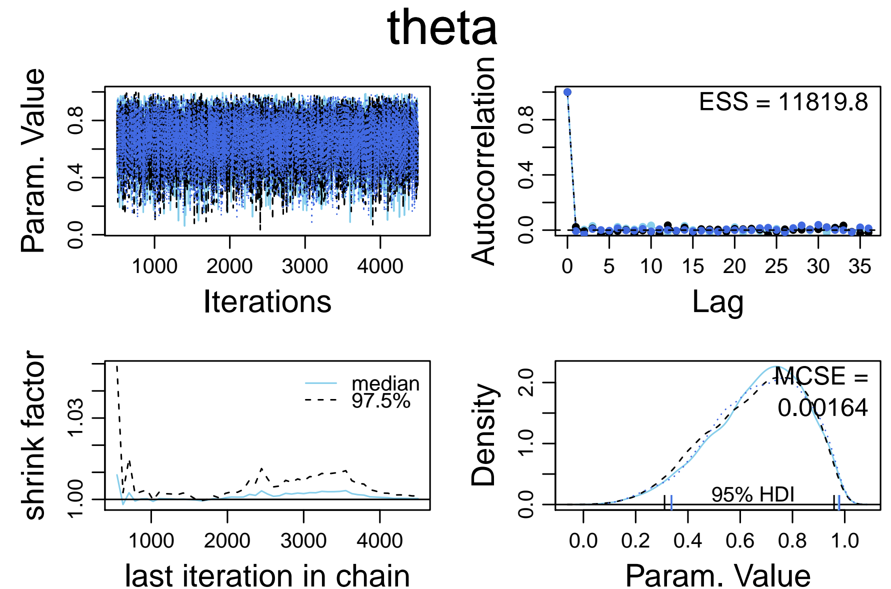
coda has some diagnostic plots which we can make:

```
> plot(codaSamples)
```



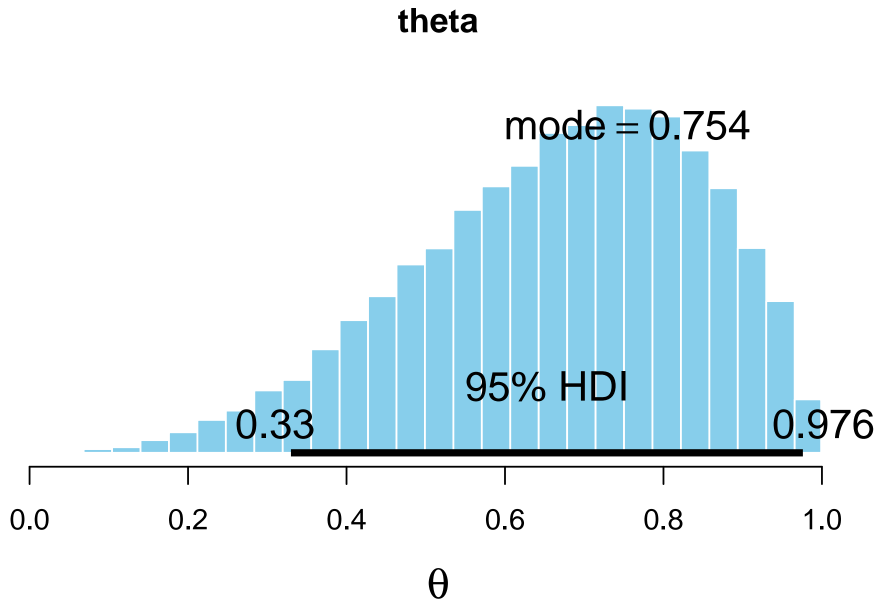
There are more extensive plots available from the software package associated with Kruschke, which we can get from `https://sites.google.com/site/doingbayesiandataanalysis/software-installation/`

```
> source('DBDA2Eprograms/DBDA2E-utilities.R')
> diagMCMC(codaObject = codaSamples, parName = 'theta')
> dev.copy2pdf(file='notes_comp_jags_DBDA2E.pdf',
+               height=4, width=6)
```



Note that some of the diagnostics are trivial here. Since there's only one parameter, we're sampling from the posterior itself, and the samples are actually uncorrelated. DBDA2E also has a nice convenience function for plotting the posterior of a variable.

```
> plotPost(codaSamples[,'theta'],main="theta",
+          xlab = bquote(theta))
           ESS      mean    median      mode
theta 12307.27 0.6673316 0.6887558 0.7543944
      hdiMass    hdiLow   hdiHigh compVal pGtCompVal
theta    0.95 0.3298472 0.9759335      NA         NA
      ROPElow ROPEhigh pLtROPE pInROPE pGtROPE
theta      NA       NA      NA      NA      NA
```



If you want, you can also extract the chains themselves and calculate the usual averages from them:

```
> mean(codaSamples[[1]][,'theta'])
[1] 0.6713674
> mean(codaSamples[[2]][,'theta'])
[1] 0.6618588
> mean(codaSamples[[3]][,'theta'])
[1] 0.6687686
```

2.4 Example: Hierarchical Normal Model (Gelman 11.6)

Consider the model of Section 11.6 in Gelman. In this model the data are $\{y_{ij}\}$ with $j = 1, \ldots, J$ and $i = 1, \ldots, n_j$, which are assumed to be independently drawn from normal distributions $N(\theta_j, \sigma^2)$. I.e., there are J different samples of possibly different sizes $\{n_j\}$, drawn from normal distributions with different means $\{\theta_j\}$ and the same variance σ^2. The prior on the variance is non-informative (uniform in $\ln\sigma$), and the prior on the means is iid normal $N(\mu, \tau^2)$. In the past, we'd have specified values for μ and τ as part of the model, but this is a *hierarchical model* in which μ and τ are hyperparameters with their own prior distribution, which we take to be uniform in μ and $\tau > 0$. (Note that this is a noninformative prior on μ but not on τ, for technical reasons.) Explicitly, the probability distributions describing the model are

$$p(\mathbf{y}|\boldsymbol{\theta}, \sigma, \mu, \tau, I) = p(\mathbf{y}|\boldsymbol{\theta}, \sigma, I) \propto \sigma^{-\sum_{j=1}^{J} n_j} \exp\left(\frac{1}{2\sigma^2}\sum_{j=1}^{J}\sum_{i=1}^{n_j}(y_{ij}-\theta_j)^2\right) \tag{2.24a}$$

$$p(\boldsymbol{\theta}|\mu, \tau, I) \propto \tau^{-J} \exp\left(\frac{1}{2\tau^2}\sum_{j=1}^{J}(\theta_j-\mu)^2\right) \tag{2.24b}$$

$$p(\sigma|\mu, \tau, I) = p(\sigma|I) \propto \sigma^{-1} \tag{2.24c}$$

$$p(\mu|I) = \text{const} \tag{2.24d}$$

$$p(\tau|I) = \text{const} \tag{2.24e}$$

and we're trying to simulate draws from the joint posterior

$$p(\boldsymbol{\theta}, \sigma, \mu, \tau|\mathbf{y}, I) \propto p(\mathbf{y}|\boldsymbol{\theta}, \sigma, I)\, p(\boldsymbol{\theta}|\mu, \tau, I)\, p(\sigma|I)\, p(\mu|I)\, p(\tau|I) \tag{2.25}$$

We can represent the data (given in Table 11.2 of Gelman) in R as a list of lists:

```
> y=list(list(62,60,63,59),
+        list(63,67,71,64,65,66),
+        list(68,66,71,67,68,68),
+        list(56,62,60,61,63,64,63,59))
> J = length(y)
```

```
> n = rep(0/0,J)
> for (j in 1:J) {n[j] = length(y[[j]])}
```

Note that the value y_{ij} is stored as `y[[j]][[i]]` because of the nature of the data structure. In any event, this format doesn't work in JAGS, so we have to "pack" the data into a flat vector (note that the most convenient way to do this stores the indices backwards within each sub-vector):

```
> ypacked = rep(0/0,sum(n))
> for (j in 1:J) {
+     for(i in 1:n[j]) {
+         k = sum(n[1:j]) - (i-1)
+         print(k)
+         ypacked[k] = y[[j]][[i]]
+     }
+ }
[1] 4
[1] 3
[1] 2
[1] 1
[1] 10
[1] 9
[1] 8
[1] 7
[1] 6
[1] 5
[1] 16
[1] 15
[1] 14
[1] 13
[1] 12
[1] 11
[1] 24
[1] 23
[1] 22
[1] 21
[1] 20
[1] 19
[1] 18
[1] 17
```

We can then represent the model in jags in a file which can be found at
http://ccrg.rit.edu/~whelan/courses/2017_1sp_STAT_489/data/ps08_prob2_model.txt
and looks like this:

```
model {
   tau ~ dunif(1e-8,1e8)
   logsigma ~ dunif(-1e8,1e8)
   mu ~ dunif(-1e8,1e8)
   for (j in 1:J) {
       theta[j] ~ dnorm(mu,tau^(-2))
       for (i in 1:n[j]) {
y[sum(n[1:j]) - (i-1)] ~ dnorm(theta[j],exp(-2*logsigma))
       }
   }
}
```

We can initialize the model with the command

```
> jagsModel = jags.model(file='ps08_prob2_model.txt',
+        data=list(y=ypacked,n=n,J=J),n.chains=10)
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 47

Initializing model
```

```
|++++++++++++++++++++++++++++++++++++++++++++++++++| 100%
```

You will explore this model further in problem set 8.

Tuesday 4 April 2017 – refer to Chapter 5 of Sivia or or Chapter 9 of McElreath

3 Hamiltonian Monte Carlo

When we use sampling as a computational method, we're inverting the correspondence associated with the frequentist interpretation of probability. Long-term frequencies for repeated experiments are approximated by the appropriate probability distribution. When we sample, we're simulating repeated experiments and using the frequencies of observations to estimate the underlying probabilities, even if the probability distribution doesn't describe real-world repeatable experiments. The Hamiltonian Monte Carlo (HMC) method is based on another such inversion. In statistical physics, the behavior of a large number of particles has properties which can be deduced from a probability distribution. HMC simulates the motion of a large number of particles, and uses them to estimate probabilities associated with an associated distribution.

The basic idea is this: suppose you have a bunch of particles, each of which has a position vector $\vec{x}$ and a momentum vector $\vec{p} = m\vec{v} = m\frac{d\vec{x}}{dt}$; mostly they move under the influence of some force which gives each one a potential energy $V(\vec{x})$ (e.g., if the box is in a uniform gravitational field with an acceleration of g, the potential energy is $mgh(\vec{x})$ where $h(\vec{x})$ is the height corresponding to position $\vec{x}$), but occasionally they bounce off of each other (or the walls of a box) in a way that leaves the total energy of the particles unchanged. Each particle will have a kinetic energy

$$\frac{1}{2}\sum_{j=1}^{3} mv_j^2 = \frac{1}{2}m\vec{v}\cdot\vec{v} = \frac{\vec{p}\cdot\vec{p}}{2m} \tag{3.1}$$

and a combined potential and kinetic energy

$$E(\vec{x},\vec{p}) = \frac{\vec{p}\cdot\vec{p}}{2m} + V(\vec{x}) \tag{3.2}$$

if the particles are allowed to fly around and bounce off of each other in this energy-conserving way, they will settle into an equilibrium configuration, where the number of particles in a small volume (d^3x) centered on a point $\vec{x}$ with momenta in a small volume (d^3p) centered on $\vec{p}$ will be

$$N(\vec{x},\vec{p}) \propto e^{-\beta E(\vec{x},\vec{p})}\, d^3x\, d^3p \tag{3.3}$$

where β is a constant which turns out to be $\beta = \frac{1}{k_B T}$, where T is the temperature of the gas, and k_B is a physical constant (Boltzmann's constant[4]) which is associated with the definition of the temperature scale. This distribution function

$$f(\vec{x},\vec{p}) = \frac{N(\vec{x},\vec{p})}{d^3x\, d^3p\, N} \propto e^{-\frac{\beta}{2m}\sum_{j=1}^{3} p_j^2} e^{-\beta V(\vec{x})} \tag{3.4}$$

looks like a joint distribution with a Gaussian distribution for the three components of $\vec{p}$ and a distribution proportional to $e^{-\beta V(\vec{x})}$ for the three components of $\vec{x}$. The Hamiltonian Monte Carlo method makes an analogy where the position vector $\vec{x}$ is

[4]This is as good a time as any to break out the following quote from Goodstein's *States of Matter* (Prentice-Hall, 1975): "Ludwig Boltzmann, who spent much of his life studying statistical mechanics, died in 1906, by his own hand. Paul Ehrenfest, carrying on the work, died similarly in 1933. Now it is our turn to study statistical mechanics. Perhaps it will be wise to approach the subject cautiously."

replaced by the parameter space vector $\boldsymbol{\theta}$ and the potential energy $V(\vec{x})$ is replaced by $-\beta^{-1}\ln p(\boldsymbol{\theta}|\mathbf{y},I)$ constructed from the target distribution $p(\boldsymbol{\theta}|\mathbf{y},I)$, where β is chosen for convenience. We also add to the model a new vector of parameters $\boldsymbol{\phi}$, one corresponding to each of the "real" parameters in $\boldsymbol{\theta}$. This vector $\boldsymbol{\phi}$ of "momenta" is the replacement for the momentum vector $\vec{p}$ in the analogy, and its target distribution is taken to be Gaussian.

Before we get into the specifics of the HMC method, it's useful to consider why the equilibrium state of an interacting set of particles has a distribution that goes like $e^{-\beta E}$, because it can be seen as an application of maximum entropy, a method often used to construct distributions for use in Bayesian analysis.

3.1 Maximum Entropy

Suppose that you have N "particles", each of which can be put into one of I "states". If the ith state has N_i particles, the total numer of ways to pick N_1 particles for the first state, N_2 for the second, etc, is

$$\Omega = \frac{N!}{N_1!N_2!\cdots N_I!} \tag{3.5}$$

If the particles can each individually wander around from state to state, the probability of finding a particular configuration $\{N_i\}$ is proportional to the multiplicity Ω associated with that configuration. In thermodynamics, the typical value of N is around 10^{23}, so it's ridiculously unlikely that the system won't end up in a configuration very "close" to the one which maximizes Ω. Because Ω is such a large number, it's conventional to take the logarithm of it to define the *entropy*

$$S = k_B \ln\Omega = k_B\left(\ln N! - \sum_{i=1}^{I}\ln N_i!\right) \tag{3.6}$$

where k_B is Boltzmann's constant again. If we use Stirling's formula $\ln N! \approx N\ln N - N$ we can write

$$\ln\Omega \approx N\ln N - \cancel{N} - \sum_{i=1}^{I} N_i\ln N_i + \cancel{\sum_{i=1}^{I} N_i} = -\sum_{i=1}^{I} N_i \ln\frac{N_i}{N} \tag{3.7}$$

If we write $p_i = \frac{N_i}{N}$ as the fraction of particles in state i (which is also the probability for any randomly chosen particle to be found in state i), the overwhelmingly most likely configuration will be the one which maximizes

$$\frac{S}{Nk_B} = -\sum_{i=1}^{I} p_i\ln p_i \tag{3.8}$$

We can learn interesting things by maximizing this entropy. First, suppose there are no constraints (no conservation laws) and every configuration of the system is accessible. Then we just need to maximize the entropy. Upon reflection, this is not totally unconstrained, since we must have $\sum_{i=1}^{I} p_i = 1$ (or equivalently $\sum_{i=1}^{I} N_i = N$). To do the constrained maximization, we add a Lagrange multiplier α and maximize

$$L = -\sum_{i=1}^{I} p_i\ln p_i - \alpha\left(\sum_{i=1}^{I} p_i - 1\right) \tag{3.9}$$

by solving the equations

$$0 = \frac{\partial L}{\partial p_i} = -\ln p_i - \frac{p_i}{p_i} - \alpha \tag{3.10a}$$

$$0 = \frac{\partial L}{\partial \alpha} = \sum_{i=1}^{I} p_i - 1 \tag{3.10b}$$

The first equation (actually I equations) can be solved for

$$p_i = e^{-1-\alpha} = \text{const} \tag{3.11}$$

and the second equation tells us to choose the constant α so that $\sum_{i=1}^{I} p_i = 1$, i.e., $p_i = \frac{1}{I}$.

Now consider the original problem: each state has an "energy" E_i, and the interactions are such that the only configurations which are accessible are those with $\sum_{i=1}^{I} E_i = E$. To maximize the entropy subject to this additional constraint, we add in another Lagrange multiplier β and maximize

$$L = -\sum_{i=1}^{I} p_i \ln p_i - \alpha \left(\sum_{i=1}^{I} p_i - 1 \right) - \beta \left(\sum_{i=1}^{I} p_i E_i - E \right) \tag{3.12}$$

which means solving the equations

$$0 = \frac{\partial L}{\partial p_i} = -\ln p_i - \frac{p_i}{p_i} - \alpha - \beta E_i \tag{3.13a}$$

$$0 = \frac{\partial L}{\partial \alpha} = \sum_{i=1}^{I} p_i - 1 \tag{3.13b}$$

$$0 = \frac{\partial L}{\partial \beta} = \sum_{i=1}^{I} p_i E_i - E \tag{3.13c}$$

The first equation tells us

$$p_i = e^{-1-\alpha-\beta E_i} \propto e^{-\beta E_i} \tag{3.14}$$

where the last two equations instruct us to choose α and β appropriately so that $\sum_{i=1}^{I} p_i = 1$ and $\sum_{i=1}^{I} p_i E_i = E$. (Note that this only works if E_i is bounded from below, so that there's a maximum value of $e^{-\beta E_i}$.)

3.2 Maximum Entropy as a Statistical Principle

We've now demonstrated the result we asserted from statistical physics, that a system allowed to interact while conserving energy will end up in a distribution where the density of particles in phase space is proportional to $e^{-\beta E(\vec{x},\vec{p})}$. The principle of maximum entropy is also useful as an extension to Bayesian analysis, to construct prior distributions and, for that matter, sampling distributions which make minimal assumptions. For example, we could consider a discrete distribution $p(\mathbf{x}|I)$ for a data vector $\mathbf{x}$ in which each x_j, $j = 1, \ldots, n$, with the constraint

$$\begin{aligned} E\left(\sum_{j=1}^{n} x_j \middle| n, I \right) &= \sum_{\mathbf{x}} p(\mathbf{x}|n, I) \sum_{j=1}^{n} x_j \\ &= \sum_{x_1=0}^{1} \cdots \sum_{x_n=0}^{1} p(\mathbf{x}|n, I) \sum_{j=1}^{n} x_j = \mu \end{aligned} \tag{3.15}$$

for some specified $\mu \in [0, n]$. You can show that the distribution which maximizes

$$-\sum_{\mathbf{x}} p(\mathbf{x}|n, I) \ln p(\mathbf{x}|n, I) \tag{3.16}$$

subject to this constraint (and normalization) is

$$p(\mathbf{x}|n, I) = \left(\frac{\mu}{n} \right)^{\sum_{j=1}^{n} x_j} \left(1 - \frac{\mu}{n} \right)^{n - \sum_{j=1}^{n} x_j} \tag{3.17}$$

One could also consider instead the probability distribution for $y = \sum_{j=1}^{n} x_j$, but we'll quickly see that the distribution which maximizes $-\sum_{y=0}^{n} p(y|n, I) \ln p(y|n, I)$ is not the same as that which maximizes the previously considered entropy $-\sum_{\mathbf{x}} p(\mathbf{x}|n, I) \ln p(\mathbf{x}|n, I)$. It's evident that

$$p(y|n, I) = \sum_{\substack{\mathbf{x} \\ \sum x_i = y}} p(\mathbf{x}|n, I) \tag{3.18}$$

and since the maximum-entropy $p(\mathbf{x}|n, I)$ depends only on $y = \sum_{j=1}^{n} x_j$,

$$p(\mathbf{x}|n, I) = \frac{p(y|n, I)}{m(y|n, I)} \tag{3.19}$$

where

$$m(y|n, I) = \binom{n}{y} = \frac{n!}{y!(n-y)!} \tag{3.20}$$

is the number of distinct $\mathbf{x}$ vectors corresponding to a given y. This can be thought of as a multiplicity of states, and it's also known as the *measure* associated with the discrete space of y values. The entropy can then be written as

$$\begin{aligned} -\sum_{\mathbf{x}} p(\mathbf{x}|n, I) \ln p(\mathbf{x}|n, I) &= -\sum_{y=0}^{n} \sum_{\substack{\mathbf{x} \\ \sum x_i = y}} \frac{p(y|n, I)}{m(y|n, I)} \ln \frac{p(y|n, I)}{m(y|n, I)} \\ &= -\sum_{y=0}^{n} m(y|n, I) \frac{p(y|n, I)}{m(y|n, I)} \ln \frac{p(y|n, I)}{m(y|n, I)} \\ &= -\sum_{y=0}^{n} p(y|n, I) \ln \frac{p(y|n, I)}{m(y|n, I)} \end{aligned} \tag{3.21}$$

and minimizing this subject to $\sum_{y=0}^{n} p(y|n, I) = 1$ and $\sum_{y=0}^{n} p(y|n, I) y = \mu$ gives a binomial distribution with parameters n and μ/n, consistent with the previous result.

So the general form of the entropy for a discrete distribution is

$$-\sum_{\mathbf{x}} p(\mathbf{x}|I) \ln \frac{p(\mathbf{x}|I)}{m(\mathbf{x}|I)} \tag{3.22}$$

The nice thing is that we only need to know the measure $m(\mathbf{x}|I)$ up to a constant since

$$\begin{aligned} -\sum_{\mathbf{x}} p(\mathbf{x}|I) \ln \frac{p(\mathbf{x}|I)}{cm(\mathbf{x}|I)} &= -\sum_{\mathbf{x}} p(\mathbf{x}|I) \ln \frac{p(\mathbf{x}|I)}{m(\mathbf{x}|I)} \\ &\quad + \sum_{\mathbf{x}} p(\mathbf{x}|I) \ln c \\ &= -\sum_{\mathbf{x}} p(\mathbf{x}|I) \ln \frac{p(\mathbf{x}|I)}{m(\mathbf{x}|I)} + \ln c \end{aligned} \tag{3.23}$$

and the maximum entropy calculation is unaffected by adding a constant to the entropy.

The nice thing about the form (3.22) is that it extends naturally to the situation where $\mathbf{x}$ is continuous, as

$$-\int p(\mathbf{x}|I) \ln \frac{p(\mathbf{x}|I)}{m(\mathbf{x}|I)} \, d\mathbf{x} \tag{3.24}$$

where the measure $m(\mathbf{x}|I)$ is a density in $\mathbf{x}$ and therefore transforms like $p(\mathbf{x}|I)$ under changes of variables:

$$m(\mathbf{y}|I) \, d\mathbf{y} = m(\mathbf{x}|I) \, d\mathbf{x} \tag{3.25}$$

so

$$m(\mathbf{y}|I) = m(\mathbf{x}|I) \left| \det \left\{ \frac{\partial y_i}{\partial x_j} \right\} \right|^{-1} \tag{3.26}$$

You can derive a lot of nice results, e.g., if $m(x|I)$ is constant and we impose the constrants $E(x|I) = \mu$ and $\mathrm{Var}(x|I) = \sigma^2$, the maximum entropy distribution is $N(\mu, \sigma^2)$.

Thursday 6 April 2017 – refer to Chapter 12 of Gelman

3.3 Hamiltonian Mechanics

Recall the observation from last class that a group of particles moving in a potential $V(\mathbf{x})$ and interacting in a way that con-

serves energy will end up with a distribution function of the form

$$f(\vec{x},\vec{p}) \propto e^{-\frac{\beta}{2m}\sum_{j=1}^{3} p_j^2} e^{-\beta V(\vec{x})} \tag{3.27}$$

which we will use to simulate a target distribution $p(\boldsymbol{\theta}|\mathbf{y},I)$ by making the analogy $\vec{x} \leftrightarrow \boldsymbol{\theta}$, $\vec{p} \leftrightarrow \boldsymbol{\phi}$, $V(\vec{x}) \leftrightarrow -\beta^{-1}\ln p(\boldsymbol{\theta}|\mathbf{y},I)$. We'll carry out a simulation procedure where, in one stage of the procedure, $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are allowed to evolve analogous to $\vec{x}$ and $\vec{p}$. Now, Newton's second law $\vec{F} = m\vec{a} = \frac{d^2\vec{x}}{dt^2}$ tells us how the potential influences the evolution, since the force is $\vec{F} = -\vec{\nabla}V$, or

$$m\frac{d^2x_j}{dt^2} = -\frac{\partial V}{\partial x_j} \tag{3.28}$$

But since the definition of momentum is

$$\vec{p} = m\vec{v} = m\frac{d\vec{x}}{dt} \tag{3.29}$$

it's more convenient to write things in terms of a set of coupled first-order differential equations[5]

$$\frac{dp_j}{dt} = -\frac{\partial V}{\partial x_j} = -\frac{\partial H}{\partial x_j} \tag{3.30a}$$

$$\frac{dx_j}{dt} = \frac{1}{m}p_j = \frac{\partial H}{\partial p_j} \tag{3.30b}$$

The Hamiltonian Monte Carlo procedure is a random walk through parameter space, just like a Markov Chain Monte Carlo, except the proposed jump, instead of being drawn from a specified jump distribution, is performed in a combined parameter-and-momentum way using the joint distribution

$$p(\boldsymbol{\theta},\boldsymbol{\phi}|\mathbf{y},I) = p(\boldsymbol{\theta}|\mathbf{y},I)\,p(\boldsymbol{\phi}|I) \tag{3.31}$$

[5]The last form, in terms of the total energy, also known as the Hamiltonian, $H = \frac{1}{2m}\sum_{j=1}^{3} p_j^2 + V(\vec{x})$, is not important for our purposes, but it's the basis of the field of Hamiltonian mechanics, which is very powerful for dealing with situations where the system is described in terms of more complicated coördinates than the standard Cartesian $(x_1,x_2,x_3) = (x,y,z)$.

analogous to (3.27):

1. The chain is at some position $\boldsymbol{\theta}^{t-1}$. Draw a momentum $\boldsymbol{\phi}^{t-1}$ from a normal distribution $N(\mathbf{0},\mathbf{M})$. (The matrix $\mathbf{M}$, which is generally diagonal, but need not have identical diagonal elements, is the equivalent of the quantity m/β in the physical analogy.)
2. Simulate evolution for some time under the equations

$$\beta\frac{d\boldsymbol{\theta}}{dt} = \mathbf{M}^{-1}\boldsymbol{\phi} \tag{3.32a}$$

$$\beta\frac{d\boldsymbol{\phi}}{dt} = \frac{\partial}{\partial\boldsymbol{\theta}}\ln p(\boldsymbol{\theta}|\mathbf{y},I) \tag{3.32b}$$

   which are analogous to the equations of motion (3.30). Call the point you arrive at $(\boldsymbol{\theta}^*,\boldsymbol{\phi}^*)$.
3. If the simulation of continuous evolution were perfect, energy would be conserved, and the joint distribution (3.31) would have the same value at $(\boldsymbol{\theta}^*,\boldsymbol{\phi}^*)$ as it did at $(\boldsymbol{\theta}^{t-1},\boldsymbol{\phi}^{t-1})$. But since it's imperfect (evolving differential equations is somewhat expensive, so we don't want to blow our computing budget on the evolution), we will construct a Metropolis ratio

$$r = \frac{p(\boldsymbol{\theta}^*,\boldsymbol{\phi}^*|\mathbf{y},I)}{p(\boldsymbol{\theta}^{t-1},\boldsymbol{\phi}^{t-1}|\mathbf{y},I)} = \frac{p(\boldsymbol{\theta}^*|\mathbf{y},I)p(\boldsymbol{\phi}^*|I)}{p(\boldsymbol{\theta}^{t-1}|\mathbf{y},I)p(\boldsymbol{\phi}^{t-1}|I)} \tag{3.33}$$

   and accept or reject the new point based on that.
4. We then go back to step 1, at our point $\boldsymbol{\theta}^t$, and re-draw a new momentum (this is supposed to represent the interactions).

The actual way that the "evolution" is usually done is to break the finite change in $\Delta t/\beta$ into L steps of size ϵ, and "leapfrog" by updating the parameters and momenta in turn as follows

(writing $\boldsymbol{\theta}^{t-1}$ as $\boldsymbol{\theta}(0)$ and $\boldsymbol{\theta}^*$ as $\boldsymbol{\theta}(\epsilon L)$ and likewise for $\boldsymbol{\phi}$:

$$\boldsymbol{\phi}(\epsilon/2) = \boldsymbol{\phi}(0) + \frac{\epsilon}{2}\frac{\partial}{\partial\boldsymbol{\theta}} \ln p(\boldsymbol{\theta}|\mathbf{y}, I)\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}(0)} \tag{3.34a}$$

$$\boldsymbol{\theta}(\epsilon) = \boldsymbol{\theta}(0) + \epsilon\,\mathbf{M}^{-1}\boldsymbol{\phi}(\epsilon/2) \tag{3.34b}$$

$$\boldsymbol{\phi}(3\epsilon/2) = \boldsymbol{\phi}(\epsilon/2) + \epsilon\frac{\partial}{\partial\boldsymbol{\theta}} \ln p(\boldsymbol{\theta}|\mathbf{y}, I)\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}(\epsilon)} \tag{3.34c}$$

$$\boldsymbol{\theta}(2\epsilon) = \boldsymbol{\theta}(\epsilon) + \epsilon\,\mathbf{M}^{-1}\boldsymbol{\phi}(3\epsilon/2) \tag{3.34d}$$

$$\vdots$$

$$\boldsymbol{\theta}(L\epsilon) = \boldsymbol{\theta}([L-1]\epsilon) + \epsilon\,\mathbf{M}^{-1}\boldsymbol{\phi}([2L-1]\epsilon/2) \tag{3.34e}$$

$$\boldsymbol{\phi}(L\epsilon) = \boldsymbol{\phi}([2L-1]\epsilon/2) + \frac{\epsilon}{2}\frac{\partial}{\partial\boldsymbol{\theta}} \ln p(\boldsymbol{\theta}|\mathbf{y}, I)\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}(L)} \tag{3.34f}$$

Tuesday 11 April 2017 – refer to Chapter 14 of Kruschke, Section 12.6 and/or Appendix C of Gelman, or Section 8.3 of McElreath

3.4 Programming HMC with Stan

Since coding up Hamiltonian Monte Carlo by hand is even more difficult than for other types of MCMC, we'll use a library to do it for us: the software package Stan.[6] Like JAGS, it's got its own programming language, which is based on a hybrid of BUGS and C++, but there is an RStan interface which we'll use:

```
> library(rstan)
Loading required package: ggplot2
```

[6]It's written Stan and not STAN because it's named after Stanisław Ulam, just as the Gibbs Sampler is named after Physicist Josiah Gibbs. It is now also associated with the backronym "Sampling through adaptive neighborhoods".

```
Loading required package: StanHeaders
rstan (Version 2.14.1, packaged: 2016-12-28 14:55:41 UTC, G
For execution on a local, multicore CPU with excess RAM we
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
```

Since I'm running this on a multicore laptop which isn't doing anything else at the moment, I'll follow the suggestion;

```
> rstan_options(auto_write = TRUE)
> options(mc.cores = parallel::detectCores())
```

To give a minimal working example of the language, I'll go back to our old example of a single draw from a binomial, where the parameter θ has a uniform prior:

```
> modelString = "
+ data {
+   int<lower=0>n ;
+   int y ;
+ }
+ parameters {
+   real<lower=0,upper=1> theta ;
+ }
+ model {
+   y ~ binomial(n,theta) ;
+   theta ~ beta(1,1) ;
+ }
+ "
```

We have to process this model with RStan; the nice thing is that unlike with JAGS, we don't have to write it out to disk (although it is often easier to put he model in a separate file and edit that); we can provide the string containing the model:

```
> stanDSO = stan_model(model_code=modelString)
starting worker pid=18630 on localhost:11849 at 13:37:44.66
```

It's actually compiled the model using C++, into something called a dynamic shared object (DSO) which can be used without further compilation. There were some scary-looking compilation warnings, which don't appear to make a difference. Now we have a command to actually do the sampling:

```
> stanFit = sampling(object=stanDSO, data=list(n=4,y=3),
+                     chains=3, warmup=500, iter=4000, seed=20170401)
starting worker pid=18856 on localhost:11849 at 13:39:05.538
starting worker pid=18865 on localhost:11849 at 13:39:05.709
starting worker pid=18874 on localhost:11849 at 13:39:05.889

SAMPLING FOR MODEL 'b03824edadb60a7db4ccc69b7b478bfa' NOW (CHAIN 1).

Chain 1, Iteration:    1 / 4000 [  0%]  (Warmup)
Chain 1, Iteration:  400 / 4000 [ 10%]  (Warmup)
Chain 1, Iteration:  501 / 4000 [ 12%]  (Sampling)
Chain 1, Iteration:  900 / 4000 [ 22%]  (Sampling)
Chain 1, Iteration: 1300 / 4000 [ 32%]  (Sampling)
Chain 1, Iteration: 1700 / 4000 [ 42%]  (Sampling)
Chain 1, Iteration: 2100 / 4000 [ 52%]  (Sampling)
Chain 1, Iteration: 2500 / 4000 [ 62%]  (Sampling)
Chain 1, Iteration: 2900 / 4000 [ 72%]  (Sampling)
Chain 1, Iteration: 3300 / 4000 [ 82%]  (Sampling)
Chain 1, Iteration: 3700 / 4000 [ 92%]  (Sampling)
Chain 1, Iteration: 4000 / 4000 [100%]  (Sampling)
 Elapsed Time: 0.003924 seconds (Warm-up)
               0.026157 seconds (Sampling)
               0.030081 seconds (Total)


SAMPLING FOR MODEL 'b03824edadb60a7db4ccc69b7b478bfa' NOW (CHAIN 2).

Chain 2, Iteration:    1 / 4000 [  0%]  (Warmup)
Chain 2, Iteration:  400 / 4000 [ 10%]  (Warmup)
Chain 2, Iteration:  501 / 4000 [ 12%]  (Sampling)
Chain 2, Iteration:  900 / 4000 [ 22%]  (Sampling)
Chain 2, Iteration: 1300 / 4000 [ 32%]  (Sampling)
Chain 2, Iteration: 1700 / 4000 [ 42%]  (Sampling)
Chain 2, Iteration: 2100 / 4000 [ 52%]  (Sampling)
Chain 2, Iteration: 2500 / 4000 [ 62%]  (Sampling)
Chain 2, Iteration: 2900 / 4000 [ 72%]  (Sampling)
Chain 2, Iteration: 3300 / 4000 [ 82%]  (Sampling)
Chain 2, Iteration: 3700 / 4000 [ 92%]  (Sampling)
Chain 2, Iteration: 4000 / 4000 [100%]  (Sampling)
 Elapsed Time: 0.004705 seconds (Warm-up)
               0.026137 seconds (Sampling)
               0.030842 seconds (Total)


SAMPLING FOR MODEL 'b03824edadb60a7db4ccc69b7b478bfa' NOW (

Chain 3, Iteration:    1 / 4000 [  0%]  (Warmup)
Chain 3, Iteration:  400 / 4000 [ 10%]  (Warmup)
Chain 3, Iteration:  501 / 4000 [ 12%]  (Sampling)
Chain 3, Iteration:  900 / 4000 [ 22%]  (Sampling)
Chain 3, Iteration: 1300 / 4000 [ 32%]  (Sampling)
Chain 3, Iteration: 1700 / 4000 [ 42%]  (Sampling)
Chain 3, Iteration: 2100 / 4000 [ 52%]  (Sampling)
Chain 3, Iteration: 2500 / 4000 [ 62%]  (Sampling)
Chain 3, Iteration: 2900 / 4000 [ 72%]  (Sampling)
Chain 3, Iteration: 3300 / 4000 [ 82%]  (Sampling)
Chain 3, Iteration: 3700 / 4000 [ 92%]  (Sampling)
Chain 3, Iteration: 4000 / 4000 [100%]  (Sampling)
 Elapsed Time: 0.004161 seconds (Warm-up)
               0.026565 seconds (Sampling)
               0.030726 seconds (Total)
```

```
summary(stanFit)$summary
hist(as.array(stanFit)[,1,'theta'],breaks=20,
  probability = TRUE)
theta = seq(0,1,length.out = 100)
lines(theta,dbeta(theta,4,2),col='blue')
hierString = "
data {
  int<lower=0> ntot; // total number of observations
  int<lower=0> J; // number of samples
  vector[ntot] y; // observational data
  int n[J]; // number of observations in a sample
}
parameters {
  real mu; // hyperparameter mean for means
  real<lower=0> tau; // hyperparameter sd for means
  real theta[J]; // mean for each sample
  real logsigma; // log-sd for data
}
model {
  int pos;
  theta ~ normal(mu,tau);
  // Suggestion from sec 15.2 of Stan manual
  // 'Ragged data structures'
  pos = 1;
  for (j in 1:J) {
    segment(y, pos, n[j])
       ~ normal(theta[j], exp(logsigma));
    pos = pos + n[j];
  }
}
"
hierDSO = stan_model(model_code=hierString)
y=list(list(62,60,63,59),
       list(63,67,71,64,65,66),
       list(68,66,71,67,68,68),
       list(56,62,60,61,63,64,63,59))
J = length(y)
n = rep(0/0,J)
for (j in 1:J) {n[j] = length(y[[j]])}
unlist(y)
hierFit = sampling(object=hierDSO,
  data=list(y=unlist(y),n=n,ntot=sum(n),J=J),
  chains=3, warmup=500, iter=4000, seed=201704102)
summary(hierFit)$summary
pairs(hierFit)
```