Gibbs Sampling, More Bayesian Topics

Mark M. Fredrickson (mfredric@umich.edu)

Computational Methods in Statistics and Data Science (Stats 406)

Review

- **Bayesian statistics** (as compared to **frequentist statistics**):
  - Treats **parameters as random variables** and **samples as fixed values**
  - Captures beliefs about parameters with a **prior distribution** $p(\theta)$.
  - Models sample with **conditional** $f(x \mid \theta)$ **and marginal** $f(x)$ **likelihoods**
  - Combines using **Bayes' Rule** to form a **posterior distribution**:

$$\pi(\theta \mid x) = \frac{f(x \mid \theta)p(\theta)}{f(x)}$$

- Inference is performed using **integration on posterior** (or other properties of π):

$$\mathrm{E}\left(g(\theta) \mid x\right)$$

Review, cont.

- For **simple problems** (e.g., **conjugate** priors and likelihoods), we can **draw directly from posterior**
- More complicated problems require **Markov Chain Monte Carlo**
  - A **Markov Chain** is a **stochastic process** with $X(t) \mid X(t-1), X(t-2), \ldots = X(t) \mid X(t-1)$
  - If we have algorithms that are **stationary** and **ergodic**, we can use those to draw from posterior.
  - **Metropolis-Hastings** (regular, independent, symmetric)
- Practical concerns:
  - **Burn in** to drop non-stationary portion of chain
  - **Rejection rate** trades off exploration for time to get a sufficient sample size
  - We frequently have to **tune parameters** to get samplers

Non-Conjugate Prior

Return to binomial example

Last time we consider a sample which we **modeled as binomial**:

$$f(x \mid \theta) = \binom{30}{x} \theta^x (1-\theta)^{30-x}$$

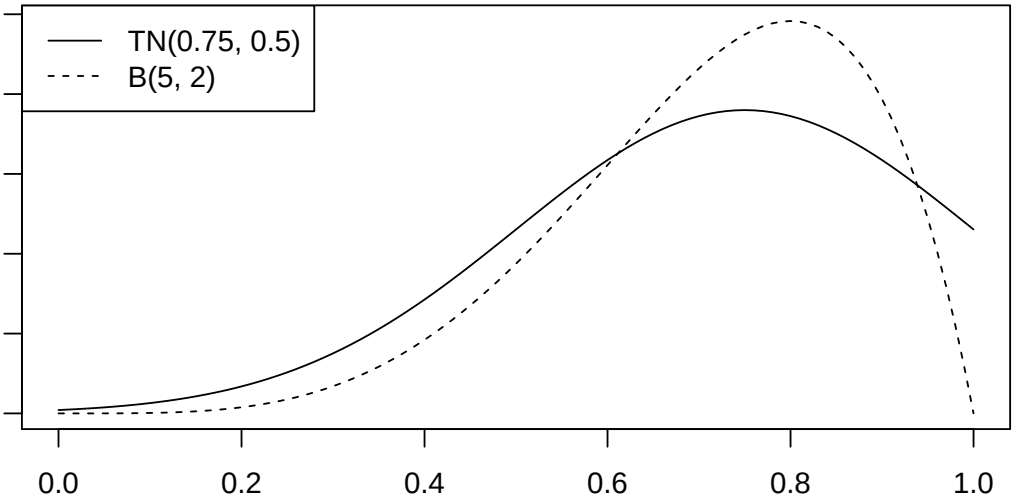
We saw that with a **beta prior** for θ, the **posterior was also beta** (conjugate).

What about other priors? Suppose instead described out beliefs using a **truncated Normal**?:

$$p(\theta) \propto \phi((\theta - \mu)/\sigma)$$

where ϕ is PDF of $N(0, 1)$. (Notice use of **proportionality**.)

Truncated Normal Prior



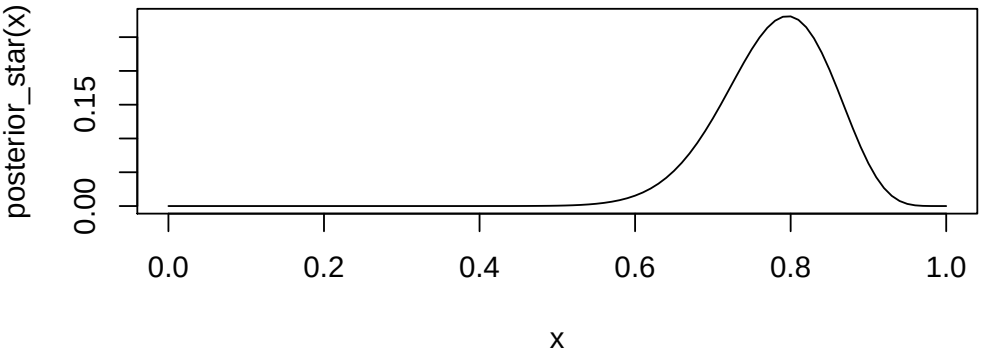
Posterior after seeing $X = 24$

What is the posterior distribution (proportional to)? Dropping **all terms not involving** θ,

$$\pi(\theta \mid x) \propto \theta^x (1-\theta)^x \exp\left\{ -\frac{(\theta - 0.75)^2}{0.5} \right\}$$

```
> posterior_star <- function(theta) {
+   dbinom(24, 30, prob = theta) * dnorm(theta, 0.75, 0.25)
+ }
```

Unnormalized Posterior



Drawing from posterior with Metropolis-Hastings

Due the complexity of the posterior, let's use MCMC to draw from π.

We need a **candidate distribution**, let's a **random walk** (bounded by [0,1]):

$$\theta(t)^* \sim U\left[a(\theta(t-1)), b(\theta(t-1))\right], a(x) = \max(0, x-\delta), b(x) = \min(x+\delta, 1))$$

so θ^* has density $1/(b(\theta(t-1)) - a(\theta(t-1)))$.

Recall, we will need to compute:

$$\begin{aligned}
\frac{\pi^*(\theta^* \mid x)}{\pi^*(\theta(t-1) \mid x)} \frac{g(\theta(t-1) \mid \theta^*)}{g(\theta^* \mid \theta(t-1)} &= \frac{\pi^*(\theta^* \mid x)}{\pi^*(\theta(t-1) \mid x)} \frac{\left[\frac{1}{b(\theta^*) - a(\theta^*)}\right]}{\left[\frac{1}{b(\theta(t-1)) - a(\theta(t-1))}\right]} \\
&= \frac{\pi^*(\theta^* \mid x)}{\pi^*(\theta(t-1) \mid x)} \frac{b(\theta(t-1)) - a(\theta(t-1))}{b(\theta^*) - a(\theta^*)}
\end{aligned}$$

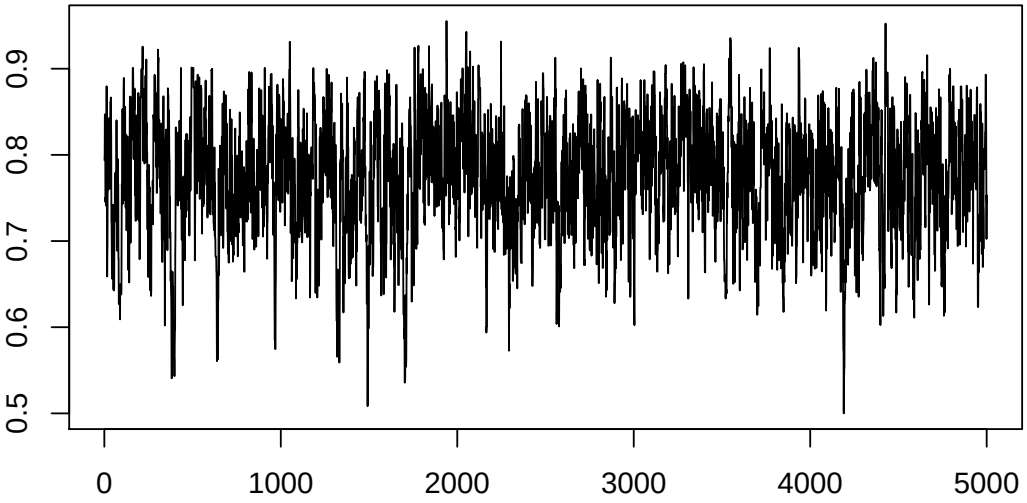
```
> findDeltas <- function(point, delta) {
+     c(max(0, point - delta), # must be positive
+       min(1, point + delta)) # must be less than 1
+ }
> drawCandidate <- function(previous, delta) {
+     ab <- findDeltas(previous, delta)
+     runif(1, ab[1], ab[2])
+ }
```

```
> nextDraw <- function(previous, delta = 0.1) {
+     candidate <- drawCandidate(previous, delta)
+
+     post_ratio <- posterior_star(candidate) / posterior_star(previous)
+     cand_ratio <- diff(findDeltas(previous, delta)) /
+                   diff(findDeltas(candidate, delta))
+
+     u <- runif(1)
+     if (u <= post_ratio * cand_ratio) {
+         return(candidate)
+     } else {
+         return(previous)
+     }
+ }
```

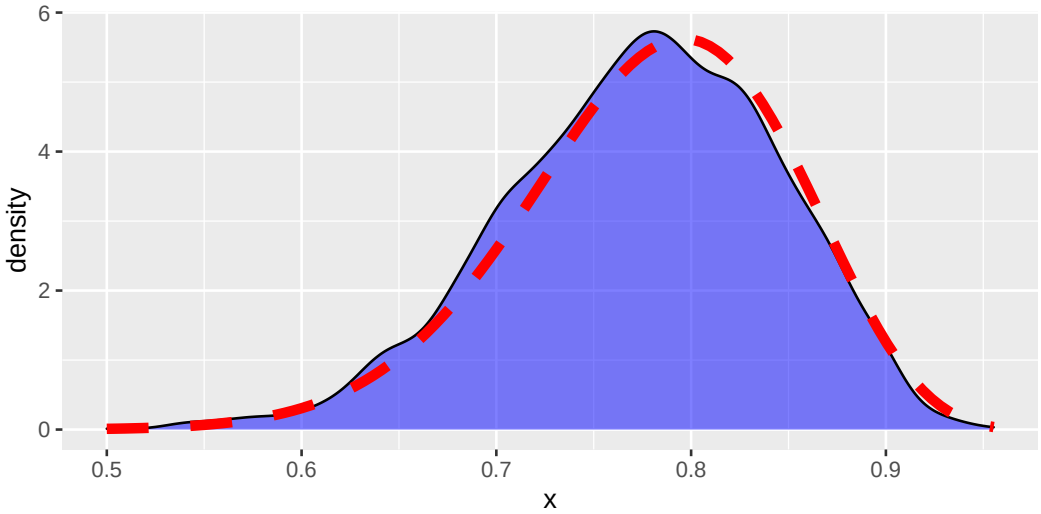
```
> B <- 10000 ; chain <- numeric(B) ; chain[1] <- 0.75
> for(i in 2:B) {
+     chain[i] <- nextDraw(chain[i - 1], 0.1)
+ }
> half_chain <- chain[5001:B] # drop burn in
> mean(half_chain > 0.75) # probability of successful teaching

[1] 0.6544
```

Chain (5000 to 10000)



Density



Multivariate Sampling

Multidimensional Posterior

So far, we've focused on **univariate** random number generation.

Goal: sample from **multivariate posterior**

$$\pi(\theta_1, \theta_2, \ldots, \theta_k \,|\, x)$$

- μ and σ^2 from a Normal distribution
- $p_1, \ldots, p_k$ where $\sum_{i=1}^{k} p_k = 1$ in a multinomial distribution
- β_0, β_1 and σ^2 from the model:

$$Y = \beta_0 + \beta_1 x + \epsilon, \quad \epsilon \sim N(0, \sigma^2)$$

- Mixture distributions with $\sigma^2 > 0, \theta \in \{0, 1\}$:

$$\theta\, N(0, \sigma^2) + (1 - \theta)\, \text{Laplace}(0, \sigma^2)$$

Metropolis-Hastings

Everything we've discussed for univariate sampling, also works for multivariate sampling.

We need to find a **multivariate candidate distribution**.

Example: Generating zero-mean bivariate Normals with variance-covariance matrix:

$$\begin{pmatrix} 1 & 0.8 \\ 0.8 & 1 \end{pmatrix}$$

As a candidate, we'll use a **random walk** with $\epsilon = N(0, 0.25I_2)$.

Setup

```
> library(mvtnorm) # includes the multivariate normal density
> B <- 5000
> chain <- matrix(0, nrow = 2, ncol = B)
> S1 <- matrix(c(1, 0.8, 0.8, 1), ncol = 2)
> S2 <- matrix(c(0.25, 0, 0, 0.25), ncol = 2)
```

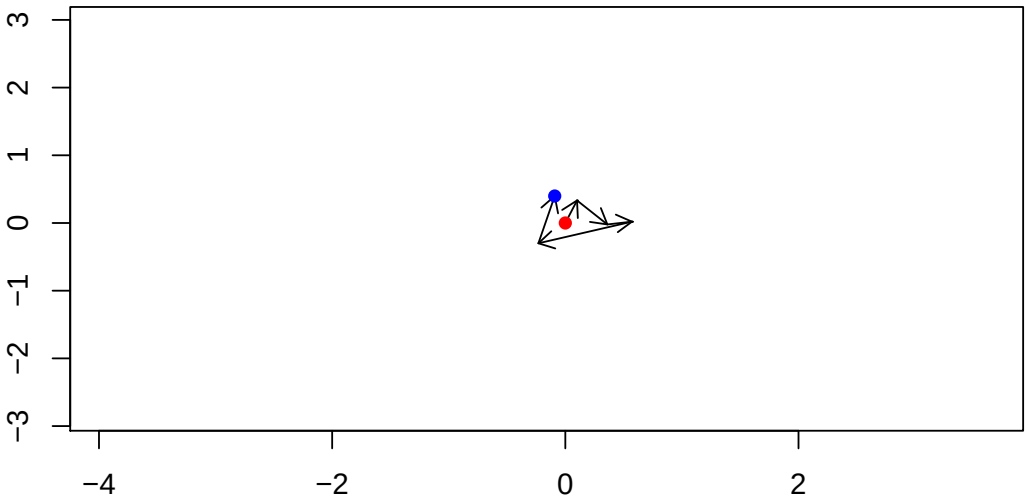
Computing Ratio

```
> ratio <- function(candidate, previous) {
+
+     ## evaluate pi(theta*) / pi(theta(t - 1))
+     a <- dmvnorm(candidate, mean = c(0, 0), sigma = S1) /
+          dmvnorm(previous, mean = c(0, 0), sigma = S1)
+
+     ## evaluate the candidate density
+     b <- dmvnorm(previous, mean = candidate, sigma = S2) /
+          dmvnorm(candidate, mean = previous, sigma = S2)
+
+     return(a * b)
+ }
```

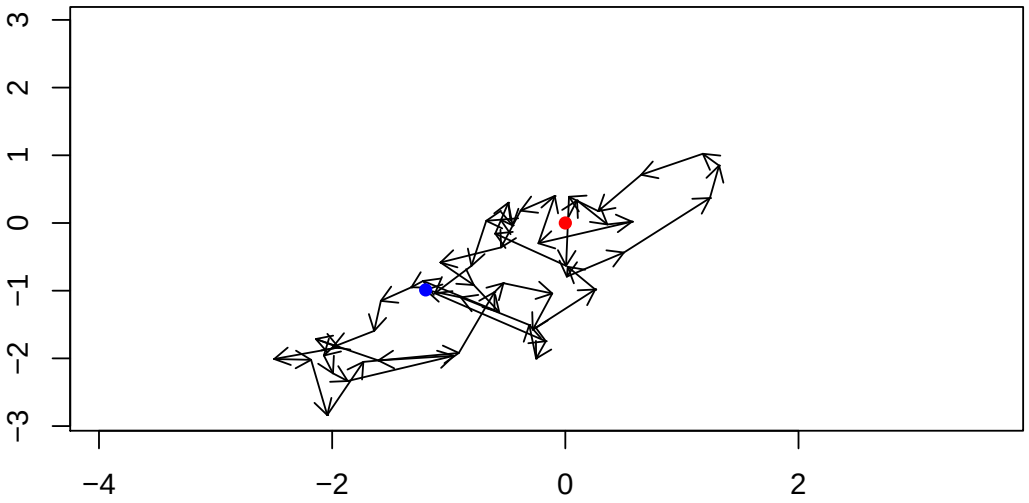
Making the Chain

```
> for (i in 2:B) {
+   ## generate 2, independent Normals at the previous point
+   candidate <- rnorm(2, mean = chain[, i - 1],
+                         sd = sqrt(0.25))
+
+   r <- ratio(candidate, chain[, i - 1])
+   if (runif(1) <= r) {
+       chain[, i] <- candidate
+   } else {
+       chain[, i] <- chain[, i - 1]
+   }
+ }
```

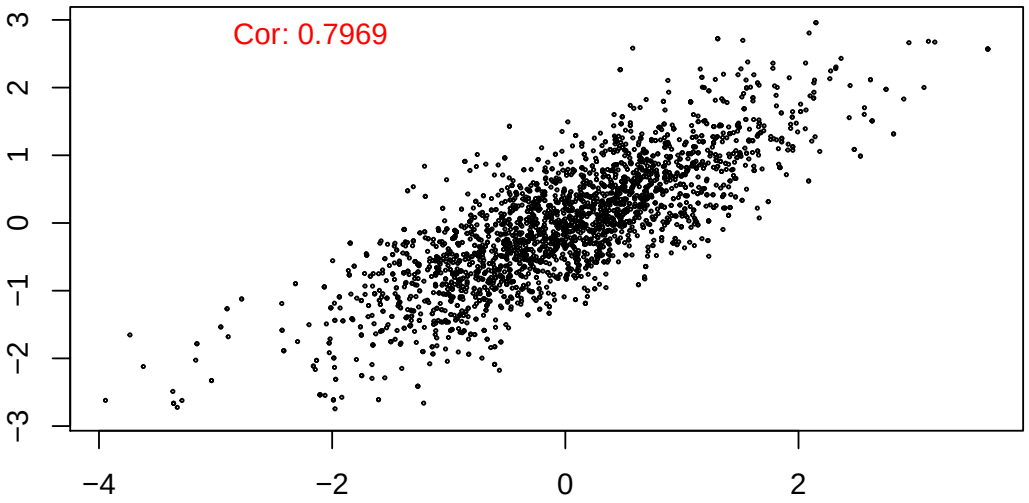
Chain: 1:10



Chain: 1:100



Chain: 2000:5000



Gibbs sampling

It may be difficult to find effective multivariate candidates.

Define the **full conditional posterior** for parameter θ_i:

$$\pi(\theta_i \,|\, \theta_1, \theta_2, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_d, x) = \pi_i(\theta_i)$$

Gibbs sampler: Step through the θ_i drawing from π_i conditional on the **rest of the parameters**.

Special case of Metropolis-Hastings that

- draws θ_i^* directly from π_i (which is always accepted)
- sets $\theta_j(t) = \theta_j(t-1)$ for all $j \neq i$.

Gibbs for Multivariate Normal

If $(\theta_1, \theta_2, \ldots)$ are MVN, all **conditional distributions** are also MVN.

Two dimensional case:

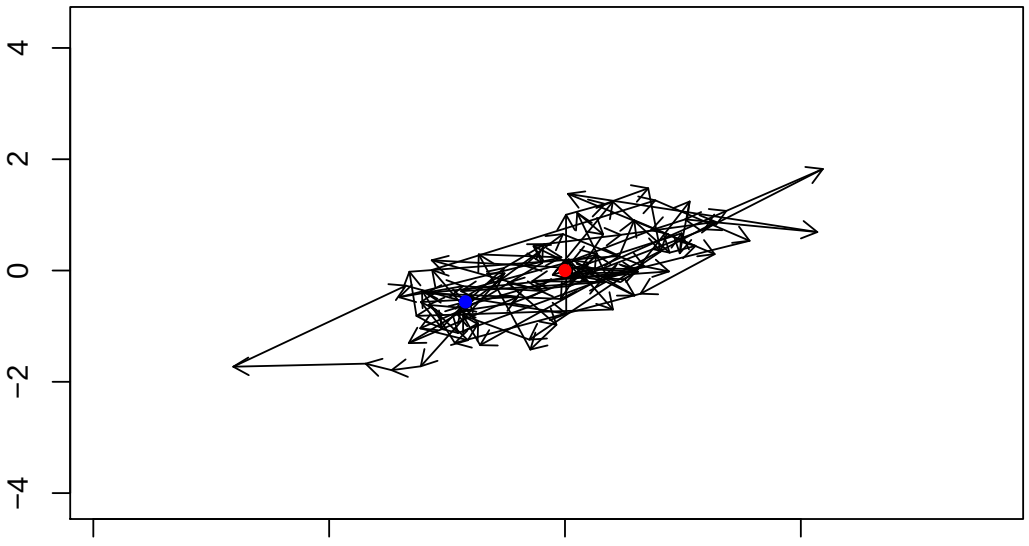
$$\theta_1 \,|\, \theta_2 = t \sim N\left(\mu_1 + \frac{\text{Cov}(\theta_1, \theta_2)}{\text{Var}(\theta_2)}(t - \mu_2), \text{Var}(\theta_1) - \frac{\text{Cov}(\theta_1, \theta_2)^2}{\text{Var}(\theta_2)}\right)$$

$$\theta_2 \,|\, \theta_1 = t \sim N\left(\mu_2 + \frac{\text{Cov}(\theta_2, \theta_1)}{\text{Var}(\theta_1)}(t - \mu_1), \text{Var}(\theta_2) - \frac{\text{Cov}(\theta_2, \theta_1)^2}{\text{Var}(\theta_1)}\right)$$

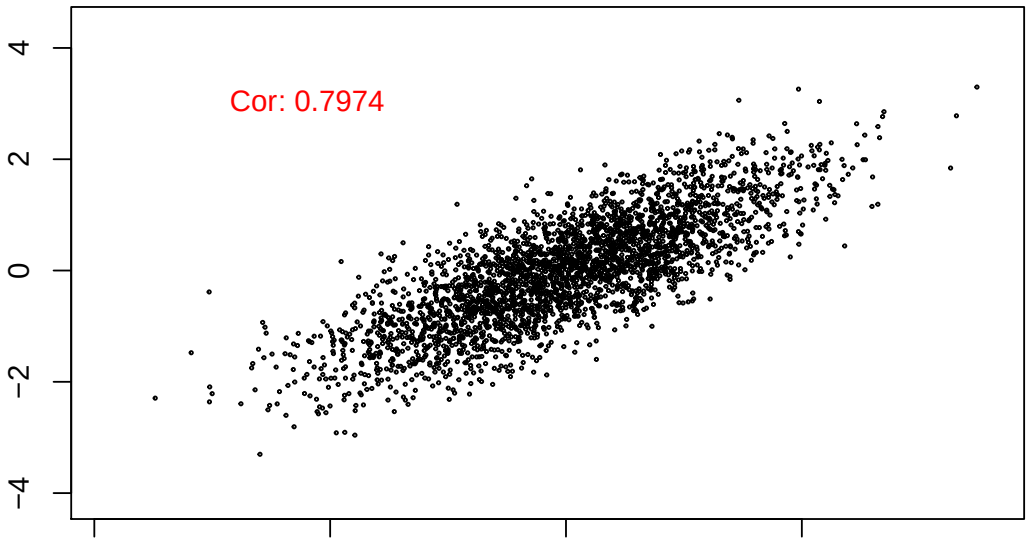
In our example, $\mu_1 = \mu_2 = 0$, $\text{Var}(\theta_1) = \text{Var}(\theta_2) = 1$, and $\text{Cov}(\theta_1, \theta_2) = 0.8$.

```
> pi_i <- function(t) {
+   rnorm(1, mean = 0.8 * t, sd = sqrt(1 - 0.8^2))
+ }
> chain_gibbs <- matrix(0, ncol = B, nrow = 2)
> for (i in 2:B) {
+     chain_gibbs[1, i] <- pi_i(chain_gibbs[2, i - 1])
+     chain_gibbs[2, i] <- pi_i(chain_gibbs[1, i])
+ }
```

Gibbs: 1:100



Gibbs: 2000:5000



Remarks

- Must be able to draw from the full conditional posterior directly.
- Often use conjugate prior distributions to make this possible.
- Gibbs sampling is frequently the choice for **hierarchical models** in which pieces are built on top of each other.

Hierarchical Model: Normal mean

Suppose we are willing to assume that

$$X_i \sim N(\mu, \sigma^2), \quad i = 1, \ldots, n$$

We have two parameters, we'll attach two convenient priors (assumed independent):

$$\mu \sim N(\theta, \tau^2), \quad \sigma^2 \sim \text{Inv. Gamma}(a, b)$$

The Inverse Gamma distribution has density:

$$f(\sigma^2) = \frac{b^a}{(\sigma^2)^{a+1}} \frac{\exp(-b/\sigma^2)}{\Gamma(a)}$$

Full conditional posteriors: useful tricks

Before we find the FCPs for this model, let's learn two useful tricks.

The full conditional posterior for any parameter is **proportional to the joint posterior**. Writing θ_{-i} for all other parameters,

$$\pi(\theta_i \mid \theta_{-i}, x, \ldots) = \frac{\pi(\theta_i, \theta_{-i} \mid x, \ldots)}{\pi(\theta_{-i} \mid x, \ldots)} \propto \pi(\theta_i, \theta_{-i} \mid x, \ldots)$$

Remember that addition in exponents is mutiplicative:

$$g(a \mid b) = \exp(a + b) = \exp(a)\exp(b) \propto \exp(a)$$

Joint Posterior Distribution

$$
\begin{aligned}
\pi(\mu, \sigma^2 \mid \theta, \tau^2, a, b, \mathbf{x}) &\propto f(\mathbf{x} \mid \mu, \sigma^2) p(\mu, \sigma^2) \\
&\propto \exp\left\{ -\frac{\sum_{i=1}^{n} (x_i - \mu)^2}{2\sigma^2} \right\} \frac{1}{(\sigma^2)^{a+1}} \exp\left\{ -b/\sigma^2 \right\} \exp\left\{ -\frac{(\mu - \theta)^2}{2\tau^2} \right\} \\
&= \exp\left\{ -\frac{\tau^2 \sum_{i=1}^{n} (x_i - \mu)^2 + \sigma^2 (\mu - \theta)^2}{2\sigma^2 \tau^2} \right\} \frac{1}{(\sigma^2)^{a+1}} \exp\left\{ -b/\sigma^2 \right\}
\end{aligned}
$$

Full Conditional for μ

$$\pi(\mu \mid \sigma^2, \theta, \tau^2, a, b, x) \propto \exp\left\{-\frac{\tau^2 \sum_{i=1}^n (x-\mu)^2 + \sigma^2(\mu-\theta)^2}{2\sigma^2\tau^2}\right\} \frac{1}{(\sigma^2)^{a+1}} \exp\left\{-b/\sigma^2\right\}$$

$$\propto \exp\left\{-\frac{\tau^2 \sum_{i=1}^n (x_i-\mu)^2 + \sigma^2(\mu-\theta)^2}{2\sigma^2\tau^2}\right\}$$

$$= \exp\left\{-\frac{\tau^2\left(\left[\sum_{i=1}^n x_i^2\right] - 2n\bar{x}\mu + n\mu^2\right) + \sigma^2\mu^2 - 2\sigma^2\mu\theta + \sigma^2\theta^2}{2\sigma^2\tau^2}\right\}$$

$$\propto \exp\left\{-\frac{(\tau^2+n\sigma^2)\mu^2 - 2(\sigma^2\mu\theta - \tau^2 n\bar{x})\mu}{2\sigma^2\tau^2}\right\}$$

$$\propto \exp\left\{-\frac{\left(\mu - (\sigma^2\mu\theta - \tau^2 n\bar{x})/(\tau^2+n\sigma^2)\right)^2}{2\sigma^2\tau^2/(\tau^2+n\sigma^2)}\right\}$$

Full Conditionals

After working through $\pi(\mu, \sigma^2 \mid x)$, we find:

$$\mu \mid \sigma^2, x \sim N\left(\frac{\sigma^2\theta + n\tau^2\bar{x}}{\sigma^2 + n\tau^2}, \frac{\sigma^2\tau^2}{\sigma^2 + n\tau^2}\right)$$

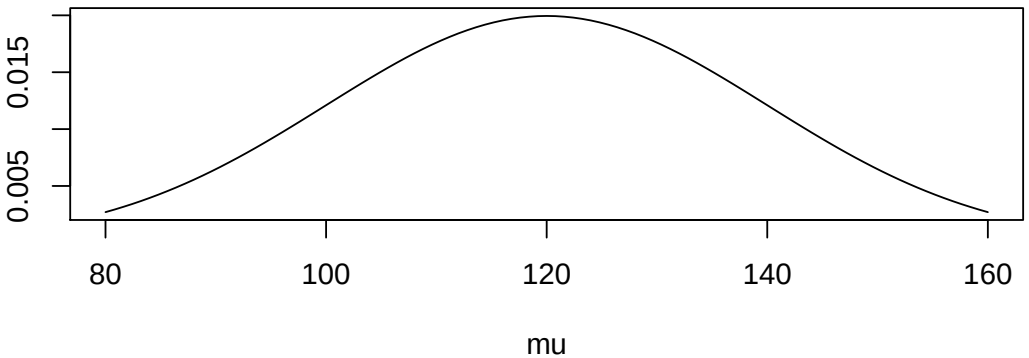
$$\sigma^2 \mid \mu, x \sim \text{Inv. Gamma}\left(\frac{n}{2} + a, \frac{1}{2}\sum_{i=1}^{n}(x-\mu)^2 + b\right)$$

Prior for μ

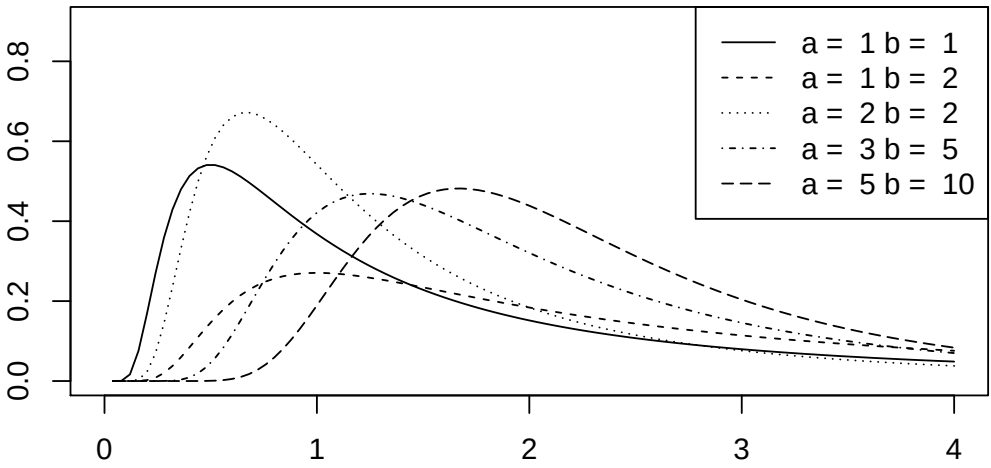
We'll work on the **systolic blood pressure** data again.

If we start from the idea that a **healthy systolic BP** is about **120**, so $\theta = 120$.

We might not be very certain about that, so let's provide a **wide variance**: $\tau = 20$.



Inverse Gamma



Prior for σ^2, setup

For my **subjective beliefs**, I think $a = 2$, $b = 5$ best captures my position.

```
> param_a <- 3
> param_b <- 5
> param_theta = 120
> param_tau2 = 20^2
> n <- dim(combined)[1] # the BP data
```

Gibbs for BP: μ step

$$\mu \,|\, \sigma^2, x \sim N\left(\frac{\sigma^2\theta + n\tau^2\bar{x}}{\sigma^2 + n\tau^2}, \frac{\sigma^2\tau^2}{\sigma^2 + n\tau^2}\right)$$

```
> xsum <- sum(combined$sys_mean)
> gibbs_mu <- function(s2) {
+   denom <- s2 + n * param_tau2
+   rnorm(1, mean = (s2 * param_theta + param_tau2 * xsum) / denom,
+         sd = sqrt(s2 * param_tau2 / denom))
+ }
```

Gibbs for BP: σ^2 step

$$\sigma^2 \,|\, \mu, x \sim \text{Inv. Gamma}\left(\frac{n}{2} + a, \frac{1}{2}\sum_{i=1}^{n}(x-\mu)^2 + b\right)$$

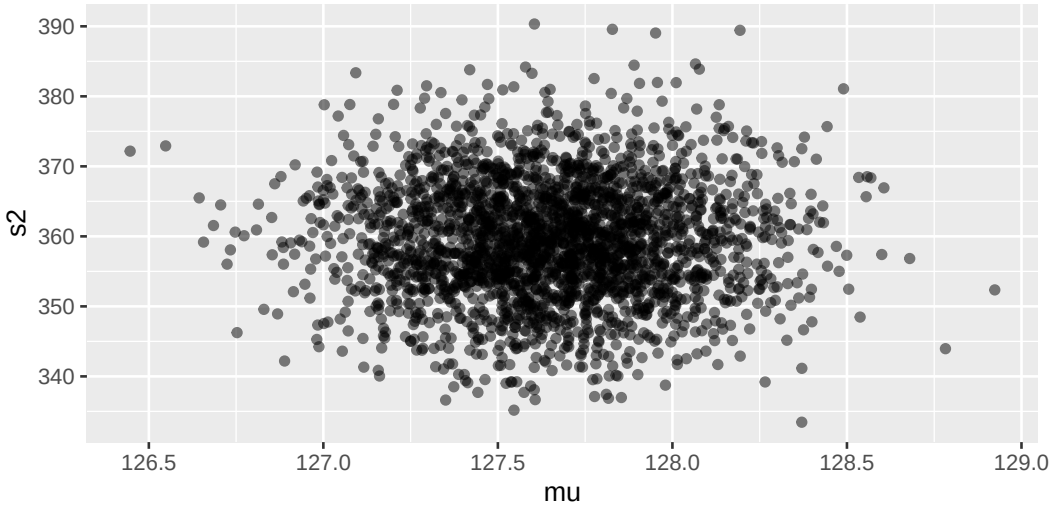
```
> shape_1 <- n/2 + param_a # doesn't depend on mu
> gibbs_s2 <- function(mu) {
+   shape_2 <- (0.5) * sum((combined$sys_mean - mu)^2) + param_b
+   g <- rgamma(1, shape_1, shape_2)
+   1 / g # recall: inverse gamma
+ }
```

Creating the chain

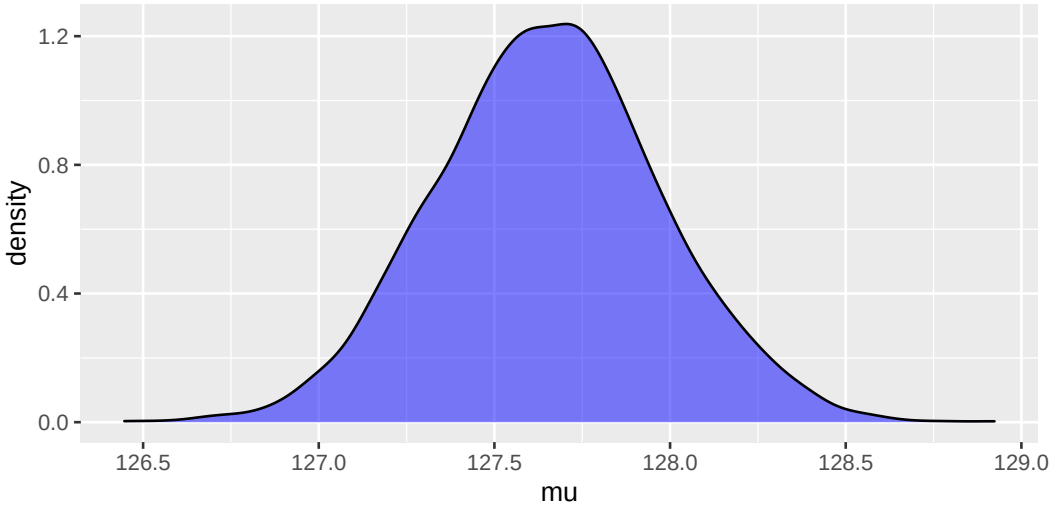
```
> gibbs_mu_sigma2 <- matrix(0, nrow = 2, ncol = B)
> ## draw from the prior to set up the chain
> gibbs_mu_sigma2[1, 1] <- rnorm(1, param_theta, sqrt(param_tau2))
> gibbs_mu_sigma2[2, 1] <- 1 / rgamma(1, param_a, param_b)

> for (i in 2:B) {
+   chain[1, i] <- gibbs_mu(chain[2, i - 1])
+   chain[2, i] <- gibbs_s2(chain[1, i])
+ }
```

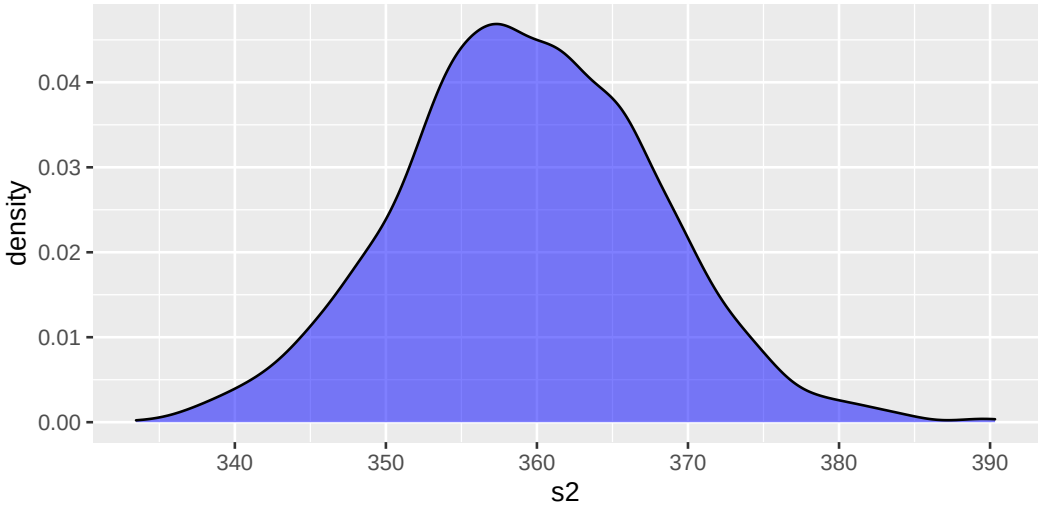
Joint distribution (discarding first 1/2 chain)



Marginal: μ



Marginal: σ^2



How much did we decrease our uncertainty?

For μ:

```
> 1 - var(chain[1, -(1:500)]) / 20^2 # recall, tau^2  = 20^2
```

```
[1] 0.9997
```

We've greatly increased our certainty on the mean.

For σ^2:

```
> prior_var <- param_b^2  / ((param_a - 1)^2  * (param_a - 2))
> 1 - var(chain[2, -(1:500)]) / prior_var
```

```
[1] -10.05
```

But now we are much less certain about the variance (perhaps we were over confident before!).

Inference

```
> # discard the first 500 draws
> stationary <- chain[, -(1:500)]
```

Bayes estimators (with regard to squared loss/MSE) for the parameters:

```
> # mu
> mean(stationary[1,])
```

```
[1] 127.7
```

```
> # sigma^2
> mean(stationary[2,])
```

```
[1] 359.2
```

More inference

95% credible intervals for the parameters (intervals that contain 95% of the mass for marginal posterior distributions:

```
> # mu
> quantile(stationary[1, ], c(0.025, 0.975))

 2.5% 97.5%
127.0 128.3

> # sigma^2
> quantile(stationary[2, ], c(0.025, 0.975))

 2.5% 97.5%
343.0 375.3
```

Gibbs Sampling Summary

- While MH can do multivariate, candidate distributions can be hard to find or inefficient
- Gibbs sampling draws parameters one at a time (much easier!)
- Downside: need to be able to draw from **full conditional posteriors**
- Algorithm for B MCMC steps
  1. Draw θ_1 conditional on $\theta_2, \theta_3, \ldots, \theta_k)$
  2. Draw θ_2 conditional on $\theta_1, \theta_3, \ldots, \theta_k)$
  3. ...
  4. Draw θ_k conditional on $\theta_1, \theta_2, \ldots, \theta_{k-1})$
- Most of the work is getting the FPCs.

Bayesian Computation with Stan

Stan: A Bayesian Programming Language

- A programming language for specifying Bayesian models
- A library R, Python, and other languages
- A collection of advanced Bayesian algorithms

Example: BP Model

$$
\begin{aligned}
X_i &\sim N(\mu, \sigma^2) \\
\mu &\sim N(120, 400) \\
\sigma^2 &\sim \text{Inv. Gamma}(2, 5)
\end{aligned}
$$

Data and Parameters

```
data {
  int<lower=0> n;
  real y[n];
}

parameters {
  real mu;
  real<lower=0> sigmasq;
}
```

Model

```
model {
  mu ~ normal(120, 20);
  sigmasq ~ inv_gamma(3, 5);
  for (i in 1:n)
      y[i] ~ normal(mu, sqrt(sigmasq));
}
```

Running the `bp.stan`

The `data`, `parameters`, and `model` sections are put into `bp.stan`.

```
> library(rstan)
> bp_stan <- stan("bp.stan",
+                 data = list(n = dim(combined)[1],
+                 y = combined$sys_mean))
```

After compiling `bp.stan` into its own executable, we see

```
SAMPLING FOR MODEL 'bp' NOW (CHAIN 1).
Chain 1:
Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
...
```

Results

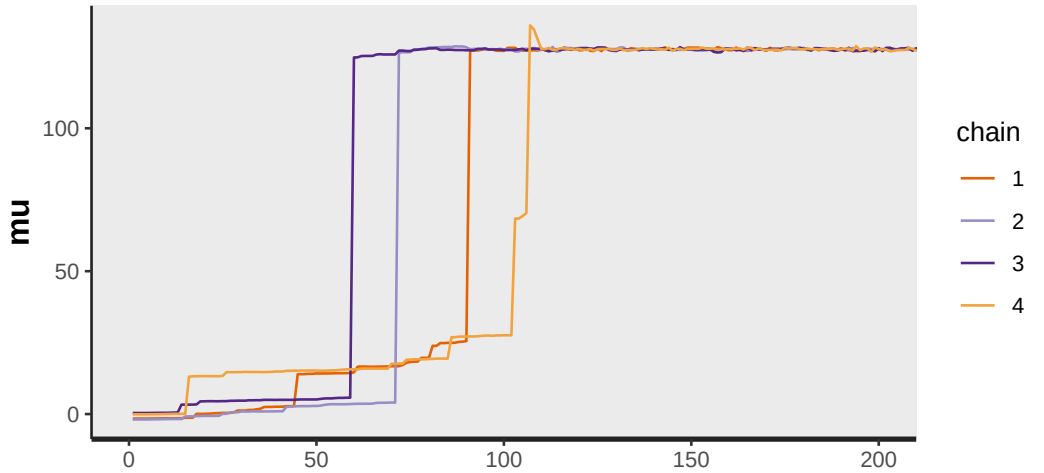
```
> print(bp_stan)

Inference for Stan model: bp.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

           mean se_mean   sd     2.5%      25%      50%
mu         127.6    0.01 0.31    127.0    127.4    127.6
sigmasq    358.6    0.15 8.37    342.4    353.0    358.4
lp__    -12243.4    0.02 0.96 -12246.0 -12243.8 -12243.1
            75%    97.5% n_eff Rhat
mu         127.8    128.2  3644    1
sigmasq    364.1    375.4  3086    1
lp__    -12242.7 -12242.5  2055    1
```

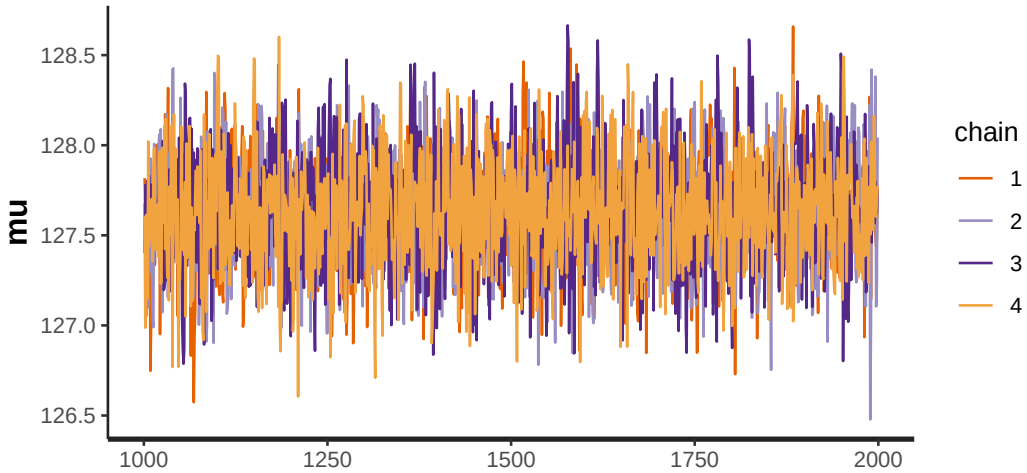
Plotting chains, warmup period

```
> traceplot(bp_stan, pars = "mu", inc_warmup = TRUE, window = c(0, 200))
```



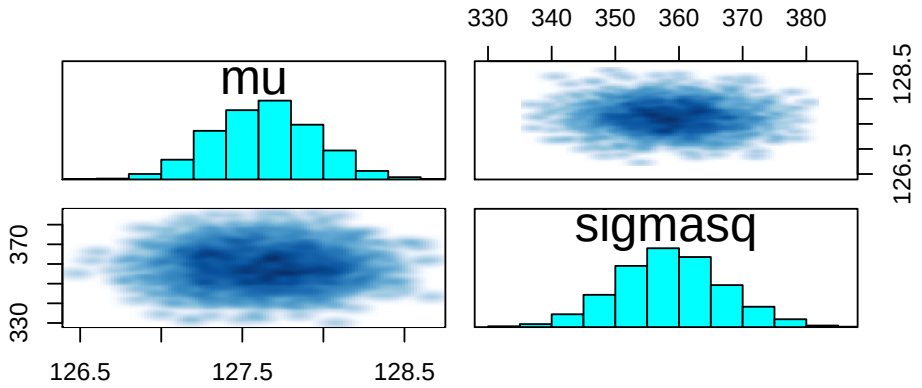
Plotting chains, convergence

```
> traceplot(bp_stan, "mu")
```



Joint Posterior

```
> pairs(bp_stan, pars = c("mu", "sigmasq"))
```



A more involved model

We previously investigated the **effect of aspirin** on BP, comparing subjects who took to those who did not.

$$
\begin{aligned}
\mu_1 &\sim N(120, 20) \\
\delta &\sim N(0, 10) \\
\sigma^2 &\sim \text{Inv. Gamma}(3, 5) \\
X_i &\sim N(\mu_1 + I(T_i = 0)\delta, \sigma^2)
\end{aligned}
$$

Data and Parameters

```
data {
  int<lower=1> n;
  int<lower=1> m;
  real takers[n];
  real nontakers[m];
}

parameters {
  real mu_1;
  real delta;
  real<lower=0> sigmasq;
}
```

Transformed Parameters and Model

```
transformed parameters {
  real mu_2 = mu_1 + delta;
}

model {
  mu_1      ~ normal(120, 20);
  delta     ~ normal(0, 10);
  sigmasq   ~ inv_gamma(3, 5);
  takers    ~ normal(mu_1, sqrt(sigmasq));
  nontakers ~ normal(mu_2, sqrt(sigmasq));
}
```

Running the model

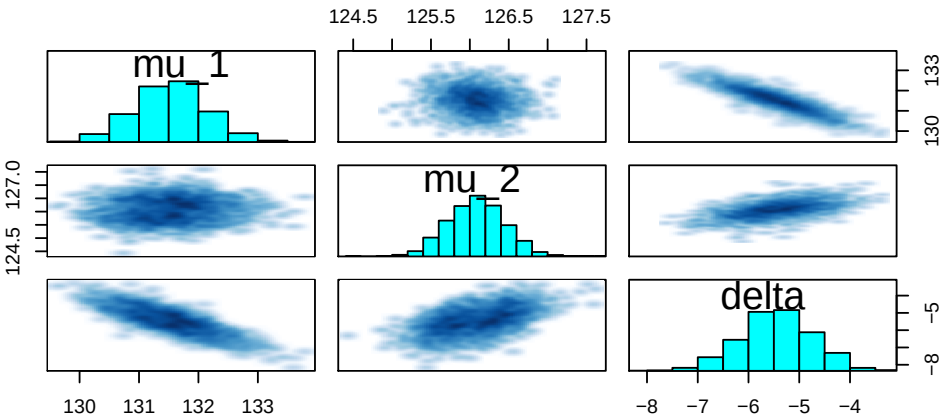
```
> aspirin_effect <- stan("aspirin_effect.stan",
+   data = with(combined,
+               list(
+                   n = sum(taking_aspirin),
+                   m = sum(!taking_aspirin),
+                   takers = sys_mean[taking_aspirin],
+                   nontakers = sys_mean[!taking_aspirin])))
```

Results

```
Inference for Stan model: aspirin_effect.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

            mean se_mean   sd      2.5%      25%       50%
mu_1      131.53    0.02 0.59    130.37    131.1    131.54
delta      -5.45    0.02 0.70     -6.86     -5.9     -5.44
sigmasq   352.93    0.16 8.24    337.06    347.3    352.75
mu_2      126.08    0.01 0.37    125.39    125.8    126.08
lp__   -12213.71    0.03 1.21 -12216.65 -12214.3 -12213.41
             75%     97.5% n_eff Rhat
mu_1      131.92    132.72  1531    1
delta      -4.97     -4.13  1645    1
sigmasq   358.32    369.39  2676    1
```

Posterior Distribution



Using the posterior distribution

```
> samples <- as.array(aspirin_effect)
> dimnames(samples)

$iterations
NULL

$chains
[1] "chain:1" "chain:2" "chain:3" "chain:4"

$parameters
[1] "mu_1"    "delta"   "sigmasq" "mu_2"    "lp__"
```

```
> ## probability that the effect size is at least 5
> mean(abs(samples[,, "delta"]) >= 5)

[1] 0.7362
```

Installing Stan

In theory, installing Stan should be as easy as:

```
> install.packages("rstan")
```

In practice, I've had **mixed success** (yes on my linux machine, no on my OS X machine).

We won't use Stan in class, but feel free to use on your final project.

Convergence

Example: $N(0, 1)$

Recall this example from last class:

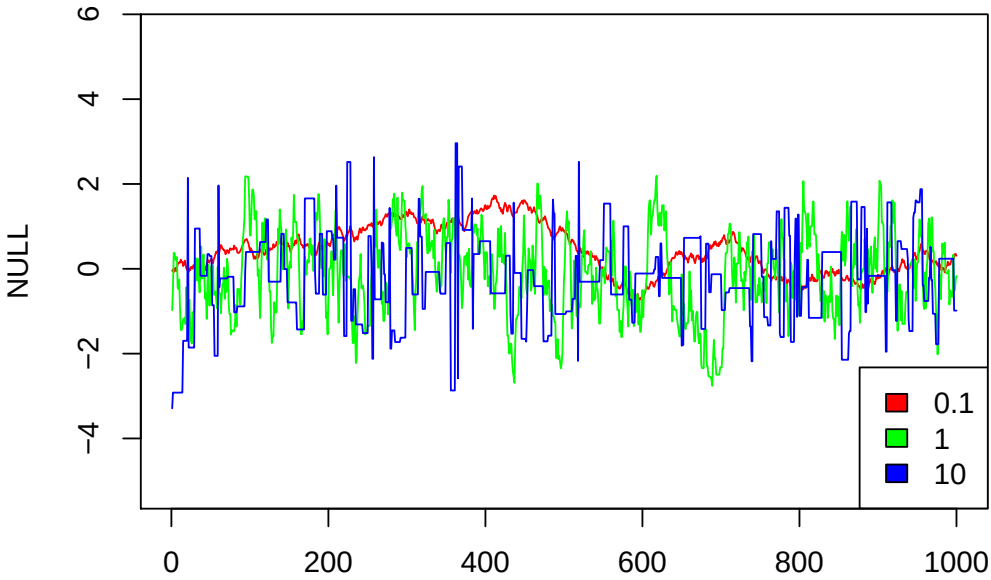
Generating draws from $N(0, 1)$ using Metroplis-Hastings with

$$\theta^* = \theta(t-1) + U(-\delta, \delta)$$

as a proposal.

```
> unif_chain <- function(delta, B = 5000) {
+   chain <- numeric(B); chain[1] <- runif(1, -delta, delta)  ; rejects <-
+   for (i in 2:B) {
+     candidate <- chain[i - 1] + runif(1, -delta, delta)
+     ratio <- dnorm(candidate) / dnorm(chain[i - 1])
+     if (runif(1) <= ratio) {
+         chain[i] <- candidate
+     } else {
+         chain[i] <- chain[i - 1]
+         rejects <- rejects + 1
+     }
+   }
+   list(reject_rate = rejects / B, chain = chain)
+ }
```

```
> n01_chain_0.1 <- unif_chain(0.1)
> n01_chain_1 <- unif_chain(1)
> n01_chain_10 <- unif_chain(10)
```


NULL
6
4
2
0
−2
−4
0
200
400
600
800
1000
0.1
1
10

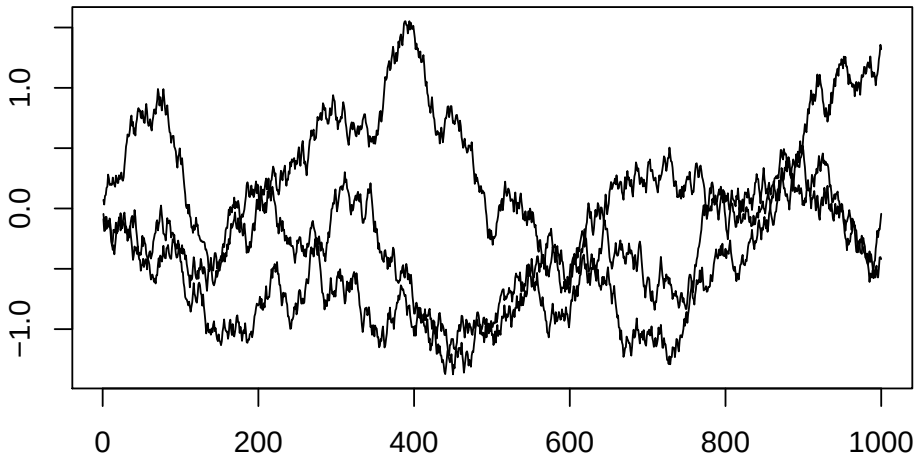
Convergence

We can see that for $\delta = 0.1$, the **chain did not converge**.

This conclusion was made easier **generating multiple chains**.

To check convergence for a single δ, we will need to generate several chains. Luckily, we can do this in **parallel.**

```
> library(parallel)
> RNGkind("L'Ecuyer-CMRG") # parallel safe PRNG
> chains_0.1 <- mclapply(rep(0.1, 3), unif_chain, mc.cores = 3, B = 1000)
```


1.0
0.0
−1.0
0
200
400
600
800
1000

Gelman-Rubin Method

Suppose we have k Markov Chains from 1 up to n:

$$X_1(0), X_1(2), \ldots, X_1(n)$$
$$\vdots$$
$$X_k(0), X_k(1), \ldots, X_k(n)$$

Let ψ be a function of n arguments. We can compute:

$$\psi_{1n} = \psi_n(X_1(0), \ldots, X_1(n))$$
$$\vdots$$
$$\psi_{kn} = \psi_n(X_k(0), \ldots, X_k(n))$$

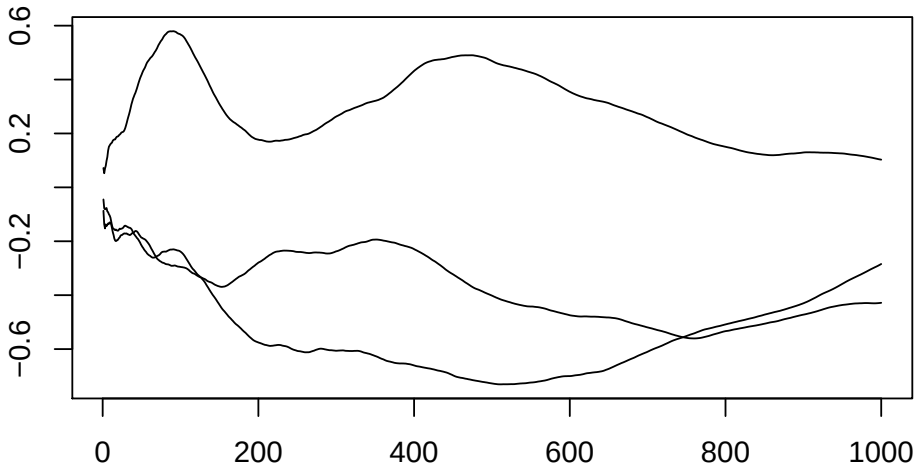
Observation: If the chains are converging to the same distribution, then **all** ψ_{in} **should converge** as well.

Example: Mean

One common choice for ψ_n is the **mean of the chain up** n:

```
> psi <- function(chain) {
+   n <- length(chain)
+   cumsum(chain) / (1:n) # cumulative mean
+ }

> psi_1 <- psi(chains_0.1[[1]]$chain)
> psi_2 <- psi(chains_0.1[[2]]$chain)
> psi_3 <- psi(chains_0.1[[3]]$chain)
```


0.6
0.2
−0.2
−0.6
0
200
400
600
800
1000

Within and Between Variances

Define the following:

$$B = \frac{n}{k-1} \sum_{i=1}^{l} (\bar{\psi}_{i\cdot} - \bar{\psi}_{\cdot\cdot})^2$$

$$W = \frac{1}{nk} \sum_{i=1}^{k} \sum_{t=1}^{n} (\psi_{it} - \bar{\psi}_{i\cdot})^2$$

where the "dot" notation means taking the mean over that index.

B measures how much **each chain's average varies about the overall average**.

W measures the **within chain variance**.

Variance of ψ

An unbiased estimate of the variance of ψ (with respect to the true posterior) is

$$V = \frac{n-1}{n} W + \frac{1}{n} B$$

However, if the chains are **overdispersed** (i.e., tending to cover more of the sample space than the posterior suggests) V **overestimates** the true variance.

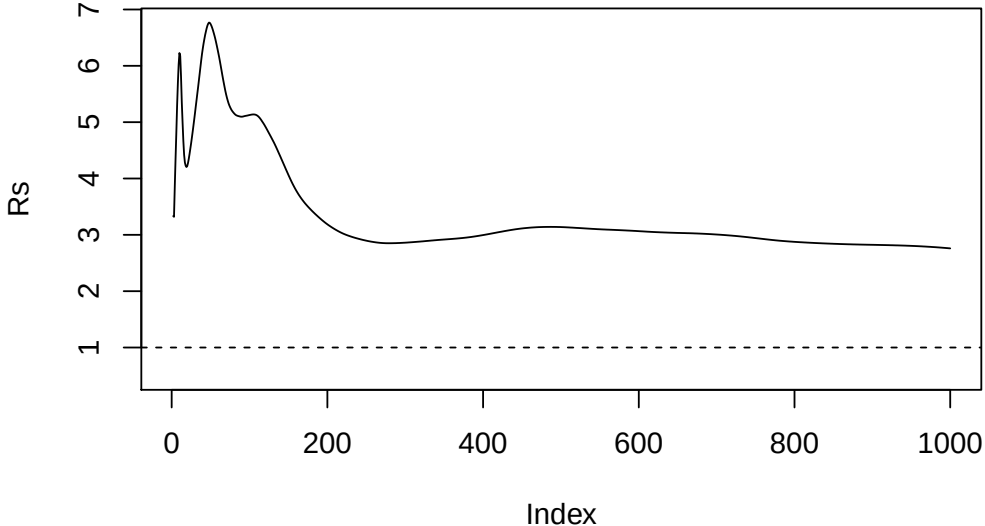
Conversely, W (on its own) **underestimates** the variance, but converges to the true variance.

This suggests:

$$\hat{R} = \sqrt{\frac{V}{W}} \approx 1 \iff \text{the chains have converged}$$

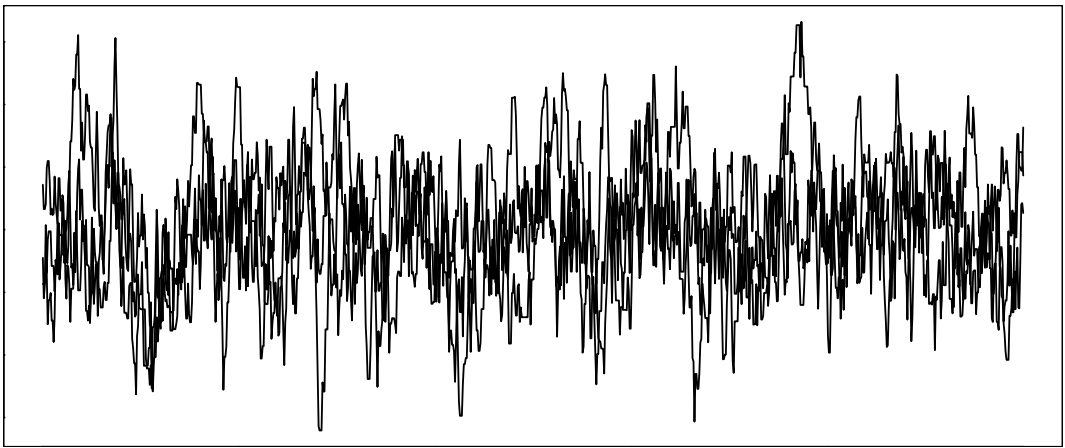
Computing $\hat{R}$

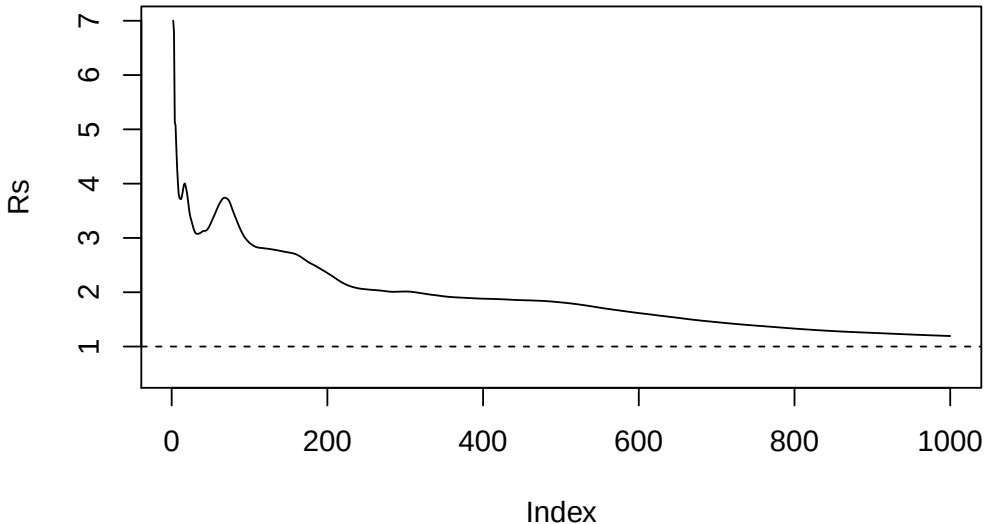
```
> Rs <- numeric(1000)
> for (t in 1:1000) {
+     p1 <- psi_1[1:t] ; p2 <- psi_2[1:t] ; p3 <- psi_3[1:t]
+     psi_bars <- c(mean(p1), mean(p2), mean(p3))
+     psi_bar_bar <- mean(c(p1, p2, p3))
+
+     B <- (t / 2) * sum((psi_bars - psi_bar_bar)^2)
+     W = (1 / (2 * t))*(sum((p1 - psi_bars[1])^2) +
+                        sum((p2 - psi_bars[2])^2) +
+                        sum((p2 - psi_bars[2])^2))
+    Rs[t] <- sqrt(((t - 1) / t * W + 1/ t * B) / W)
+ }
```


Rs
1
2
3
4
5
6
7
0
200
400
600
800
1000
Index

Repeat with better mixing

```
> chains_1 <- mclapply(rep(1, 3), unif_chain, mc.cores = 3, B = 1000)
```




Rs
1
2
3
4
5
6
7
0
200
400
600
800
1000
Index

Computing the $\hat{R}$ values can be somewhat tedious. Luckily, **Stan will compute them for us**:

```
> summary(bp_stan)$summary[, c("mean", "sd", "n_eff", "Rhat")]
```

```
             mean     sd n_eff   Rhat
mu          127.6 0.3149  3644 1.0006
sigmasq     358.6 8.3742  3086 0.9998
lp__    -12243.4 0.9593  2055 1.0019
```