Non-Conjugate Models and Grid Approximations

Patrick Lam

Outline

Outline

The Binomial Example

The Binomial Example

Suppose we have vector of data on voter turnout for a random sample of n voters in the 2004 US Presidential election.

The Binomial Example

Suppose we have vector of data on voter turnout for a random sample of n voters in the 2004 US Presidential election.

We can model the voter turnout with a binomial model.

The Binomial Example

Suppose we have vector of data on voter turnout for a random sample of n voters in the 2004 US Presidential election.

We can model the voter turnout with a binomial model.

$$Y \sim \text{Binomial}(n, \pi)$$

The Binomial Example

Suppose we have vector of data on voter turnout for a random sample of n voters in the 2004 US Presidential election.

We can model the voter turnout with a binomial model.

$$Y \sim \mathrm{Binomial}(n, \pi)$$

Quantity of interest: π (voter turnout)

The Binomial Example

Suppose we have vector of data on voter turnout for a random sample of n voters in the 2004 US Presidential election.

We can model the voter turnout with a binomial model.

$$Y \sim \mathrm{Binomial}(n, \pi)$$

Quantity of interest: π (voter turnout)

Assumptions:

The Binomial Example

Suppose we have vector of data on voter turnout for a random sample of n voters in the 2004 US Presidential election.

We can model the voter turnout with a binomial model.

$$Y \sim \text{Binomial}(n, \pi)$$

Quantity of interest: π (voter turnout)

Assumptions:

- Each voter's decision to vote follows the Bernoulli distribution.

The Binomial Example

Suppose we have vector of data on voter turnout for a random sample of n voters in the 2004 US Presidential election.

We can model the voter turnout with a binomial model.

$$Y \sim \text{Binomial}(n, \pi)$$

Quantity of interest: π (voter turnout)

Assumptions:

- Each voter's decision to vote follows the Bernoulli distribution.
- Each voter has the same probability of voting.

The Binomial Example

Suppose we have vector of data on voter turnout for a random sample of n voters in the 2004 US Presidential election.

We can model the voter turnout with a binomial model.

$$Y \sim \mathrm{Binomial}(n, \pi)$$

Quantity of interest: π (voter turnout)

Assumptions:

- Each voter's decision to vote follows the Bernoulli distribution.
- Each voter has the same probability of voting. (unrealistic)

The Binomial Example

Suppose we have vector of data on voter turnout for a random sample of n voters in the 2004 US Presidential election.

We can model the voter turnout with a binomial model.

$$Y \sim \mathrm{Binomial}(n, \pi)$$

Quantity of interest: π (voter turnout)

Assumptions:

- Each voter's decision to vote follows the Bernoulli distribution.
- Each voter has the same probability of voting. (unrealistic)
- Each voter's decision to vote is independent.

The Binomial Example

Suppose we have vector of data on voter turnout for a random sample of n voters in the 2004 US Presidential election.

We can model the voter turnout with a binomial model.

$$Y \sim \mathrm{Binomial}(n, \pi)$$

Quantity of interest: π (voter turnout)

Assumptions:

- Each voter's decision to vote follows the Bernoulli distribution.
- Each voter has the same probability of voting. (unrealistic)
- Each voter's decision to vote is independent. (unrealistic)

Non-Conjugate Priors

Non-Conjugate Priors

Suppose we decide not to use the conjugate beta prior, but rather a non-conjugate prior.

Non-Conjugate Priors

Suppose we decide not to use the conjugate beta prior, but rather a non-conjugate prior. The posterior will no longer be in a convenient distributional form.

Non-Conjugate Priors

Suppose we decide not to use the conjugate beta prior, but rather a non-conjugate prior. The posterior will no longer be in a convenient distributional form.

For example, the triangle prior we saw in class:

Non-Conjugate Priors

Suppose we decide not to use the conjugate beta prior, but rather a non-conjugate prior. The posterior will no longer be in a convenient distributional form.

For example, the triangle prior we saw in class:

$$p(\pi) = \begin{cases} 8\pi & \text{if } 0 \leq \pi < 0.25 \\ \frac{8}{3} - \frac{8}{3}\pi & \text{if } 0.25 \leq \pi \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

Non-Conjugate Priors

Suppose we decide not to use the conjugate beta prior, but rather a non-conjugate prior. The posterior will no longer be in a convenient distributional form.

For example, the triangle prior we saw in class:

$$p(\pi) = \begin{cases} 8\pi & \text{if } 0 \leq \pi < 0.25 \\ \frac{8}{3} - \frac{8}{3}\pi & \text{if } 0.25 \leq \pi \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

```
> triangle.prior <- function(x) {
+     if (x >= 0 && x < 0.25)
+         8 * x
+     else if (x >= 0.25 && x <= 1)
+         8/3 - 8 * x/3
+     else 0
+ }
```

Non-Conjugate Priors

Suppose we decide not to use the conjugate beta prior, but rather a non-conjugate prior. The posterior will no longer be in a convenient distributional form.

For example, the triangle prior we saw in class:

$$p(\pi) = \begin{cases} 8\pi & \text{if } 0 \leq \pi < 0.25 \\ \frac{8}{3} - \frac{8}{3}\pi & \text{if } 0.25 \leq \pi \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

```
> triangle.prior <- function(x) {
+     if (x >= 0 && x < 0.25)
+         8 * x
+     else if (x >= 0.25 && x <= 1)
+         8/3 - 8 * x/3
+     else 0
+ }
```

To find the posterior, we need to use grid approximation methods.

Grid Approximation Method

Grid Approximation Method

We can find the unnormalized posterior simply multiplying the binomial likelihood and the non-conjugate prior.

Grid Approximation Method

We can find the unnormalized posterior simply multiplying the binomial likelihood and the non-conjugate prior.

```
> posterior.function <- function(theta, n, y) {
+     (theta^y) * (1 - theta)^(n - y) * triangle.prior(theta)
+ }
```

Grid Approximation Method

We can find the unnormalized posterior simply multiplying the binomial likelihood and the non-conjugate prior.

```
> posterior.function <- function(theta, n, y) {
+     (theta^y) * (1 - theta)^(n - y) * triangle.prior(theta)
+ }
```

To find the normalized posterior, we need to estimate the normalizing constant, which I denote as c.

Grid Approximation Method

We can find the unnormalized posterior simply multiplying the binomial likelihood and the non-conjugate prior.

```
> posterior.function <- function(theta, n, y) {
+     (theta^y) * (1 - theta)^(n - y) * triangle.prior(theta)
+ }
```

To find the normalized posterior, we need to estimate the normalizing constant, which I denote as c.

c is simply the area under the unnormalized posterior.

Grid Approximation Method

We can find the unnormalized posterior simply multiplying the binomial likelihood and the non-conjugate prior.

```
> posterior.function <- function(theta, n, y) {
+     (theta^y) * (1 - theta)^(n - y) * triangle.prior(theta)
+ }
```

To find the normalized posterior, we need to estimate the normalizing constant, which I denote as c.

c is simply the area under the unnormalized posterior.

Once we find c, we can simply divide all the ordinates (density values) of the unnormalized posterior by c to get the ordinates of the normalized posterior.

Grid Approximation Method

We can find the unnormalized posterior simply multiplying the binomial likelihood and the non-conjugate prior.

```
> posterior.function <- function(theta, n, y) {
+     (theta^y) * (1 - theta)^(n - y) * triangle.prior(theta)
+ }
```

To find the normalized posterior, we need to estimate the normalizing constant, which I denote as c.

c is simply the area under the unnormalized posterior.

Once we find c, we can simply divide all the ordinates (density values) of the unnormalized posterior by c to get the ordinates of the normalized posterior.

There are two ways to get the normalized posterior in R.

Univariate Grid Approximation Method (harder way)

Univariate Grid Approximation Method (harder way)

1. Divide the area under the unnormalized posterior into m grids, each of width k, starting from π_0, which is a value of π that is at the far left of the parameter space.

Univariate Grid Approximation Method (harder way)

1. Divide the area under the unnormalized posterior into m grids, each of width k, starting from π_0, which is a value of π that is at the far left of the parameter space.

```
> m <- 100
> grid.points <- seq(from = 0, to = 1, length.out = m)
```

Univariate Grid Approximation Method (harder way)

1. Divide the area under the unnormalized posterior into m grids, each of width k, starting from π_0, which is a value of π that is at the far left of the parameter space.

```
> m <- 100
> grid.points <- seq(from = 0, to = 1, length.out = m)
```

2. Evaluate each grid point $(\pi_0 + k), (\pi_0 + 2k), \ldots, (\pi_0 + mk)$ at the unnormalized posterior: $p(y|\pi_0 + ik)p(\pi_0 + ik)$.

Univariate Grid Approximation Method (harder way)

1. Divide the area under the unnormalized posterior into m grids, each of width k, starting from π_0, which is a value of π that is at the far left of the parameter space.

```
> m <- 100
> grid.points <- seq(from = 0, to = 1, length.out = m)
```

2. Evaluate each grid point $(\pi_0 + k), (\pi_0 + 2k), \ldots, (\pi_0 + mk)$ at the unnormalized posterior: $p(y|\pi_0 + ik)p(\pi_0 + ik)$.

```
> unnormal.post.ord <- posterior.function(theta = grid.points,
+     n = 500, y = 285)
```

3. Estimate c (area under the unnormalized posterior) as the sum of the areas of rectangles of width k and height at the unnormalized posterior ordinates:

3. Estimate c (area under the unnormalized posterior) as the sum of the areas of rectangles of width k and height at the unnormalized posterior ordinates:

$$c \approx \sum_{i=1}^{m} k \; p(y|\pi_0 + ik)p(\pi_0 + ik)$$

3. Estimate c (area under the unnormalized posterior) as the sum of the areas of rectangles of width k and height at the unnormalized posterior ordinates:

$$c \approx \sum_{i=1}^{m} k\ p(y|\pi_0 + ik)p(\pi_0 + ik)$$

```
> k <- 1/m
> normal.constant <- sum(k * unnormal.post.ord)
```

3. Estimate c (area under the unnormalized posterior) as the sum of the areas of rectangles of width k and height at the unnormalized posterior ordinates:

$$c \approx \sum_{i=1}^{m} k \; p(y|\pi_0 + ik)p(\pi_0 + ik)$$

```
> k <- 1/m
> normal.constant <- sum(k * unnormal.post.ord)
```

4. Divide the unnormalized posterior ordinates by c to get normalized posterior ordinates.

3. Estimate c (area under the unnormalized posterior) as the sum of the areas of rectangles of width k and height at the unnormalized posterior ordinates:

$$c \approx \sum_{i=1}^{m} k \; p(y|\pi_0 + ik)p(\pi_0 + ik)$$

```
> k <- 1/m
> normal.constant <- sum(k * unnormal.post.ord)
```

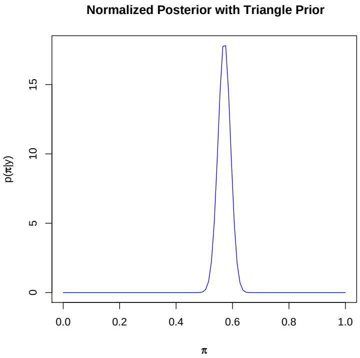
4. Divide the unnormalized posterior ordinates by c to get normalized posterior ordinates.

```
> post.ord <- unnormal.post.ord/normal.constant
```

We can use the normalized posterior ordinates to plot our normalized posterior.

We can use the normalized posterior ordinates to plot our normalized posterior.

```
> plot(x = grid.points, y = post.ord, type = "l", col = "blue",
+     xlab = expression(pi), ylab = expression(paste("p(", pi,
+         "|y)")), main = "Normalized Posterior with Triangle Prior")
```



We can also sample from our normalized posterior.

We can also sample from our normalized posterior.

```
> set.seed(12345)
> posterior.triangle.1 <- sample(grid.points, size = 10000, replace = T,
+     prob = post.ord)
```

We can also sample from our normalized posterior.

```
> set.seed(12345)
> posterior.triangle.1 <- sample(grid.points, size = 10000, replace = T,
+     prob = post.ord)
```

Note that the normalized posterior ordinates are not probabilities, but rather the heights of the density.

We can also sample from our normalized posterior.

```
> set.seed(12345)
> posterior.triangle.1 <- sample(grid.points, size = 10000, replace = T,
+     prob = post.ord)
```

Note that the normalized posterior ordinates are not probabilities, but rather the heights of the density. However, the `sample()` function in R will normalize the ordinates into probabilities.

Univariate Grid Approximation Method (easier way)

Univariate Grid Approximation Method (easier way)

Since the `sample()` function in R will automatically normalize unnormalized probability values, we do not need steps 3 and 4.

Univariate Grid Approximation Method (easier way)

Since the `sample()` function in R will automatically normalize unnormalized probability values, we do not need steps 3 and 4.

1. Divide the unnormalized posterior area into m grids.

Univariate Grid Approximation Method (easier way)

Since the `sample()` function in R will automatically normalize unnormalized probability values, we do not need steps 3 and 4.

1. Divide the unnormalized posterior area into m grids.

```
> m <- 100
> grid.points <- seq(from = 0, to = 1, length.out = m)
```

Univariate Grid Approximation Method (easier way)

Since the `sample()` function in R will automatically normalize unnormalized probability values, we do not need steps 3 and 4.

1. Divide the unnormalized posterior area into m grids.

```
> m <- 100
> grid.points <- seq(from = 0, to = 1, length.out = m)
```

2. Evaluate each grid point at the unnormalized posterior.

Univariate Grid Approximation Method (easier way)

Since the `sample()` function in R will automatically normalize unnormalized probability values, we do not need steps 3 and 4.

1. Divide the unnormalized posterior area into m grids.

```
> m <- 100
> grid.points <- seq(from = 0, to = 1, length.out = m)
```

2. Evaluate each grid point at the unnormalized posterior.

```
> unnormal.post.ord <- posterior.function(theta = grid.points,
+     n = 500, y = 285)
```

Univariate Grid Approximation Method (easier way)

Since the `sample()` function in R will automatically normalize unnormalized probability values, we do not need steps 3 and 4.

1. Divide the unnormalized posterior area into *m* grids.

```
> m <- 100
> grid.points <- seq(from = 0, to = 1, length.out = m)
```

2. Evaluate each grid point at the unnormalized posterior.

```
> unnormal.post.ord <- posterior.function(theta = grid.points,
+     n = 500, y = 285)
```

3. Sample grid points with unnormalized posterior ordinates as the probabilities (letting R normalize them for us).

Univariate Grid Approximation Method (easier way)

Since the `sample()` function in R will automatically normalize unnormalized probability values, we do not need steps 3 and 4.

1. Divide the unnormalized posterior area into m grids.

```
> m <- 100
> grid.points <- seq(from = 0, to = 1, length.out = m)
```

2. Evaluate each grid point at the unnormalized posterior.

```
> unnormal.post.ord <- posterior.function(theta = grid.points,
+     n = 500, y = 285)
```

3. Sample grid points with unnormalized posterior ordinates as the probabilities (letting R normalize them for us).

```
> set.seed(12345)
> posterior.triangle.2 <- sample(grid.points, size = 10000, replace = T,
+     prob = unnormal.post.ord)
> all.equal(posterior.triangle.1, posterior.triangle.2)

[1] TRUE
```

Outline

The Binomial Model with a Non-Conjugate Prior

Bayesian Regression with Grid Approximations

Poisson Regression

Poisson Regression

Suppose we had some data $\mathbf{y}$ that was the number of events in a given period of time.

Poisson Regression

Suppose we had some data $\mathbf{y}$ that was the number of events in a given period of time. We also have a covariate $\mathbf{x}$.

Poisson Regression

Suppose we had some data $\mathbf{y}$ that was the number of events in a given period of time. We also have a covariate $\mathbf{x}$. We can do a Bayesian Poisson regression to find the posterior for an intercept α and a slope β.

Poisson Regression

Suppose we had some data **y** that was the number of events in a given period of time. We also have a covariate **x**. We can do a Bayesian Poisson regression to find the posterior for an intercept α and a slope β.

Let's use a couple of Normal priors on α and β.

$$\text{Posterior}(\alpha, \beta) \quad \propto \quad \prod_{i=1}^{n} \text{Poisson}(\lambda_i) \times \text{Normal}(\mu_\alpha, \sigma_\alpha^2) \times \text{Normal}(\mu_\beta, \sigma_\beta^2)$$

Poisson Regression

Suppose we had some data $\mathbf{y}$ that was the number of events in a given period of time. We also have a covariate $\mathbf{x}$. We can do a Bayesian Poisson regression to find the posterior for an intercept α and a slope β.

Let's use a couple of Normal priors on α and β.

$$
\begin{aligned}
\mathrm{Posterior}(\alpha, \beta) &\propto \prod_{i=1}^{n} \mathrm{Poisson}(\lambda_i) \times \mathrm{Normal}(\mu_\alpha, \sigma_\alpha^2) \times \mathrm{Normal}(\mu_\beta, \sigma_\beta^2) \\
p(\alpha, \beta | \mathbf{y}, \mathbf{x}) &\propto \prod_{i=1}^{n} \frac{e^{-\lambda_i} \lambda_i^{y_i}}{y_i!} \times \frac{1}{\sqrt{2\pi\sigma_\alpha^2}} \exp\left(-\frac{(\alpha - \mu_\alpha)^2}{2\sigma_\alpha^2}\right) \\
&\quad \times \frac{1}{\sqrt{2\pi\sigma_\beta^2}} \exp\left(-\frac{(\beta - \mu_\beta)^2}{2\sigma_\beta^2}\right)
\end{aligned}
$$

Our covariate x_i affects the mean of the Poisson likelihood λ_i.

Our covariate x_i affects the mean of the Poisson likelihood λ_i.

Our linear predictor $\alpha + \beta x_i$ can predict negative values, but our Poisson mean λ_i is always positive, so we can use the **link function** to reparameterize λ_i so that it can take on negative values.

Our covariate x_i affects the mean of the Poisson likelihood λ_i.

Our linear predictor $\alpha + \beta x_i$ can predict negative values, but our Poisson mean λ_i is always positive, so we can use the **link function** to reparameterize λ_i so that it can take on negative values.

For the Poisson, the link function is the natural log function:

$$\ln(\lambda_i) = \alpha + \beta x_i$$

Our covariate x_i affects the mean of the Poisson likelihood λ_i.

Our linear predictor $\alpha + \beta x_i$ can predict negative values, but our Poisson mean λ_i is always positive, so we can use the **link function** to reparameterize λ_i so that it can take on negative values.

For the Poisson, the link function is the natural log function:

$$\ln(\lambda_i) = \alpha + \beta x_i$$

However, typically we use the **inverse link function** to reparameterize $\alpha + \beta x_i$ so that it can only take on positive values.

Our covariate x_i affects the mean of the Poisson likelihood λ_i.

Our linear predictor $\alpha + \beta x_i$ can predict negative values, but our Poisson mean λ_i is always positive, so we can use the **link function** to reparameterize λ_i so that it can take on negative values.

For the Poisson, the link function is the natural log function:

$$\ln(\lambda_i) = \alpha + \beta x_i$$

However, typically we use the **inverse link function** to reparameterize $\alpha + \beta x_i$ so that it can only take on positive values.

$$\lambda_i = \exp(\alpha + \beta x_i)$$

Common Link (and Inverse Link) Functions

Common Link (and Inverse Link) Functions

Poisson:

- Link: $\ln(\lambda_i) = \alpha + \beta x_i$
- Inverse Link: $\lambda_i = \exp(\alpha + \beta x_i)$

Common Link (and Inverse Link) Functions

Poisson:

- Link: $\ln(\lambda_i) = \alpha + \beta x_i$
- Inverse Link: $\lambda_i = \exp(\alpha + \beta x_i)$

Logit:

- Link: $\ln\left(\frac{\pi_i}{1-\pi_i}\right) = \alpha + \beta x_i$
- Inverse Link: $\pi_i = \frac{e^{\alpha+\beta x_i}}{1+e^{\alpha+\beta x_i}} = \frac{1}{1+e^{-(\alpha+\beta x_i)}}$

Common Link (and Inverse Link) Functions

Poisson:

- Link: $\ln(\lambda_i) = \alpha + \beta x_i$
- Inverse Link: $\lambda_i = \exp(\alpha + \beta x_i)$

Logit:

- Link: $\ln\left(\frac{\pi_i}{1-\pi_i}\right) = \alpha + \beta x_i$
- Inverse Link: $\pi_i = \frac{e^{\alpha+\beta x_i}}{1+e^{\alpha+\beta x_i}} = \frac{1}{1+e^{-(\alpha+\beta x_i)}}$

Probit:

- Link: $\Phi^{-1}(\pi_i) = \alpha + \beta x_i$
- Inverse Link: $\pi_i = \Phi(\alpha + \beta x_i)$

Back to our Poisson Model

Back to our Poisson Model

So our posterior is

$$p(\alpha, \beta | \mathbf{y}, \mathbf{x}) \propto \prod_{i=1}^{n} \frac{e^{-\lambda_i} \lambda_i^{y_i}}{y_i!} \times \frac{1}{\sqrt{2\pi\sigma_\alpha^2}} \exp\left(-\frac{(\alpha - \mu_\alpha)^2}{2\sigma_\alpha^2}\right) \times \frac{1}{\sqrt{2\pi\sigma_\beta^2}} \exp\left(-\frac{(\beta - \mu_\beta)^2}{2\sigma_\beta^2}\right)$$

Back to our Poisson Model

So our posterior is

$$p(\alpha, \beta | \mathbf{y}, \mathbf{x}) \propto \prod_{i=1}^{n} \frac{e^{-\lambda_i} \lambda_i^{y_i}}{y_i!} \times \frac{1}{\sqrt{2\pi\sigma_\alpha^2}} \exp\left(-\frac{(\alpha - \mu_\alpha)^2}{2\sigma_\alpha^2}\right) \times \frac{1}{\sqrt{2\pi\sigma_\beta^2}} \exp\left(-\frac{(\beta - \mu_\beta)^2}{2\sigma_\beta^2}\right)$$

Using our inverse link function, we get

$$p(\alpha, \beta | \mathbf{y}, \mathbf{x}) \propto \prod_{i=1}^{n} \frac{e^{-\exp(\alpha + \beta x_i)} (\exp(\alpha + \beta x_i))^{y_i}}{y_i!} \times \frac{1}{\sqrt{2\pi\sigma_\alpha^2}} \exp\left(-\frac{(\alpha - \mu_\alpha)^2}{2\sigma_\alpha^2}\right) \times \frac{1}{\sqrt{2\pi\sigma_\beta^2}} \exp\left(-\frac{(\beta - \mu_\beta)^2}{2\sigma_\beta^2}\right)$$

Computationally, we like to work with the log posterior (we can add small numbers rather than multiplying) and then exponentiate.

Computationally, we like to work with the log posterior (we can add small numbers rather than multiplying) and then exponentiate.

Our log posterior is

$$
\begin{aligned}
\ln p(\alpha, \beta | \mathbf{y}, \mathbf{x}) \quad \propto \quad & \sum_{i=1}^{n} \ln \left(\frac{e^{-\exp(\alpha+\beta x_i)} (\exp(\alpha + \beta x_i))^{y_i}}{y_i!} \right) \\
& + \ln \left(\frac{1}{\sqrt{2\pi\sigma_\alpha^2}} \exp \left(-\frac{(\alpha - \mu_\alpha)^2}{2\sigma_\alpha^2} \right) \right) \\
& + \ln \left(\frac{1}{\sqrt{2\pi\sigma_\beta^2}} \exp \left(-\frac{(\beta - \mu_\beta)^2}{2\sigma_\beta^2} \right) \right)
\end{aligned}
$$

Computationally, we like to work with the log posterior (we can add small numbers rather than multiplying) and then exponentiate.

Our log posterior is

$$
\begin{aligned}
\ln p(\alpha, \beta | \mathbf{y}, \mathbf{x}) \quad \propto \quad & \sum_{i=1}^{n} \ln \left(\frac{e^{-\exp(\alpha+\beta x_i)} (\exp(\alpha + \beta x_i))^{y_i}}{y_i!} \right) \\
& + \ln \left(\frac{1}{\sqrt{2\pi\sigma_\alpha^2}} \exp \left(- \frac{(\alpha - \mu_\alpha)^2}{2\sigma_\alpha^2} \right) \right) \\
& + \ln \left(\frac{1}{\sqrt{2\pi\sigma_\beta^2}} \exp \left(- \frac{(\beta - \mu_\beta)^2}{2\sigma_\beta^2} \right) \right)
\end{aligned}
$$

We can combine and simplify a bunch of terms . . .

Computationally, we like to work with the log posterior (we can add small numbers rather than multiplying) and then exponentiate.

Our log posterior is

$$
\begin{aligned}
\ln p(\alpha, \beta | \mathbf{y}, \mathbf{x}) \quad \propto \quad & \sum_{i=1}^{n} \ln \left(\frac{e^{-\exp(\alpha+\beta x_i)} (\exp(\alpha + \beta x_i))^{y_i}}{y_i!} \right) \\
& + \ln \left(\frac{1}{\sqrt{2\pi\sigma_\alpha^2}} \exp\left(-\frac{(\alpha - \mu_\alpha)^2}{2\sigma_\alpha^2} \right) \right) \\
& + \ln \left(\frac{1}{\sqrt{2\pi\sigma_\beta^2}} \exp\left(-\frac{(\beta - \mu_\beta)^2}{2\sigma_\beta^2} \right) \right)
\end{aligned}
$$

We can combine and simplify a bunch of terms ... or we can use some canned R functions.

$$\ln \text{Posterior}(\alpha, \beta) \quad \propto \quad \sum_{i=1}^{n} \ln \text{Poisson}(\exp(\alpha + \beta x_i)) + \ln \text{Normal}(\mu_\alpha, \sigma_\alpha^2)$$
$$+ \ln \text{Normal}(\mu_\beta, \sigma_\beta^2)$$

$$\ln \text{Posterior}(\alpha, \beta) \propto \sum_{i=1}^{n} \ln \text{Poisson}(\exp(\alpha + \beta x_i)) + \ln \text{Normal}(\mu_\alpha, \sigma_\alpha^2) + \ln \text{Normal}(\mu_\beta, \sigma_\beta^2)$$

```
> poisson.posterior <- function(theta, y, x, prior.mean.a, prior.var.a,
+     prior.mean.b, prior.var.b) {
+     a <- theta[1]
+     b <- theta[2]
+     lambda <- exp(a + b * x)
+     log.like <- sum(dpois(y, lambda = lambda, log = T))
+     log.prior.a <- dnorm(a, mean = prior.mean.a, sd = sqrt(prior.var.a),
+         log = T)
+     log.prior.b <- dnorm(b, mean = prior.mean.b, sd = sqrt(prior.var.b),
+         log = T)
+     log.post <- log.like + log.prior.a + log.prior.b
+     return(exp(log.post))
+ }
```

An Example

An Example

Let's do an example using the `sanction` dataset from the `Zelig` library.

An Example

Let's do an example using the `sanction` dataset from the `Zelig` library.

```
> library(Zelig)
> data(sanction)
```

An Example

Let's do an example using the `sanction` dataset from the `Zelig` library.

```
> library(Zelig)
> data(sanction)
```

Our dependent variable is the number of sanctions, and our covariate is level of cooperation.

An Example

Let's do an example using the `sanction` dataset from the `Zelig` library.

```
> library(Zelig)
> data(sanction)
```

Our dependent variable is the number of sanctions, and our covariate is level of cooperation.

```
> y.vec <- sanction$num
> x.vec <- sanction$coop
```

An Example

Let's do an example using the `sanction` dataset from the `Zelig` library.

```
> library(Zelig)
> data(sanction)
```

Our dependent variable is the number of sanctions, and our covariate is level of cooperation.

```
> y.vec <- sanction$num
> x.vec <- sanction$coop
```

For the priors on α and β, let's use two Normal(0,20) distributions, which are relatively flat.

An Example

Let's do an example using the `sanction` dataset from the `Zelig` library.

```
> library(Zelig)
> data(sanction)
```

Our dependent variable is the number of sanctions, and our covariate is level of cooperation.

```
> y.vec <- sanction$num
> x.vec <- sanction$coop
```

For the priors on α and β, let's use two Normal(0,20) distributions, which are relatively flat.

```
> mu.a <- mu.b <- 0
> sigma2.a <- sigma2.b <- 20
```

Since we don't know the normalizing constant and we don't have conjugacy, we need to simulate from the posterior somehow.

Since we don't know the normalizing constant and we don't have conjugacy, we need to simulate from the posterior somehow.

Fortunately, since our posterior is bivariate, we can still use the grid approximation method fairly easily.

Since we don't know the normalizing constant and we don't have conjugacy, we need to simulate from the posterior somehow.

Fortunately, since our posterior is bivariate, we can still use the grid approximation method fairly easily.

One possible problem with a grid approximation approach is that our parameter space is unbounded.

Since we don't know the normalizing constant and we don't have conjugacy, we need to simulate from the posterior somehow.

Fortunately, since our posterior is bivariate, we can still use the grid approximation method fairly easily.

One possible problem with a grid approximation approach is that our parameter space is unbounded. If our grid misses an area of the posterior that has high density, then our simulations will be incorrect.

Since we don't know the normalizing constant and we don't have conjugacy, we need to simulate from the posterior somehow.

Fortunately, since our posterior is bivariate, we can still use the grid approximation method fairly easily.

One possible problem with a grid approximation approach is that our parameter space is unbounded. If our grid misses an area of the posterior that has high density, then our simulations will be incorrect.

One reasonable solution is to center our grid at the MLE.

Since we don't know the normalizing constant and we don't have conjugacy, we need to simulate from the posterior somehow.

Fortunately, since our posterior is bivariate, we can still use the grid approximation method fairly easily.

One possible problem with a grid approximation approach is that our parameter space is unbounded. If our grid misses an area of the posterior that has high density, then our simulations will be incorrect.

One reasonable solution is to center our grid at the MLE.

Since we don't know the normalizing constant and we don't have conjugacy, we need to simulate from the posterior somehow.

Fortunately, since our posterior is bivariate, we can still use the grid approximation method fairly easily.

One possible problem with a grid approximation approach is that our parameter space is unbounded. If our grid misses an area of the posterior that has high density, then our simulations will be incorrect.

One reasonable solution is to center our grid at the MLE.

```
> mle <- glm(num ~ coop, data = sanction, family = poisson)$coef
> mle.se <- summary(glm(num ~ coop, data = sanction,
+     family = poisson))$coef[, 2]
```

I then create a grid that is centered at the MLE and spans an arbitrary 5 standard errors away on both sides.

I then create a grid that is centered at the MLE and spans an arbitrary 5 standard errors away on both sides.

```
> grid.a <- seq(from = mle[1] - 5 * mle.se[1], to = mle[1] + 5 *
+     mle.se[1], length.out = 200)
> grid.b <- seq(from = mle[2] - 5 * mle.se[2], to = mle[2] + 5 *
+     mle.se[2], length.out = 200)
> grid.points <- expand.grid(grid.a, grid.b)
```

I then create a grid that is centered at the MLE and spans an arbitrary 5 standard errors away on both sides.

```
> grid.a <- seq(from = mle[1] - 5 * mle.se[1], to = mle[1] + 5 *
+     mle.se[1], length.out = 200)
> grid.b <- seq(from = mle[2] - 5 * mle.se[2], to = mle[2] + 5 *
+     mle.se[2], length.out = 200)
> grid.points <- expand.grid(grid.a, grid.b)
```

Once we have the grid points, we can evaluate at the unnormalized posterior

I then create a grid that is centered at the MLE and spans an arbitrary 5 standard errors away on both sides.

```
> grid.a <- seq(from = mle[1] - 5 * mle.se[1], to = mle[1] + 5 *
+     mle.se[1], length.out = 200)
> grid.b <- seq(from = mle[2] - 5 * mle.se[2], to = mle[2] + 5 *
+     mle.se[2], length.out = 200)
> grid.points <- expand.grid(grid.a, grid.b)
```

Once we have the grid points, we can evaluate at the unnormalized posterior

```
> post.ord <- apply(grid.points, MARGIN = 1, FUN = poisson.posterior,
+     y = y.vec, x = x.vec, prior.mean.a = mu.a, prior.var.a = sigma2.a,
+     prior.mean.b = mu.b, prior.var.b = sigma2.b)
```

I then create a grid that is centered at the MLE and spans an arbitrary 5 standard errors away on both sides.

```
> grid.a <- seq(from = mle[1] - 5 * mle.se[1], to = mle[1] + 5 *
+     mle.se[1], length.out = 200)
> grid.b <- seq(from = mle[2] - 5 * mle.se[2], to = mle[2] + 5 *
+     mle.se[2], length.out = 200)
> grid.points <- expand.grid(grid.a, grid.b)
```

Once we have the grid points, we can evaluate at the unnormalized posterior

```
> post.ord <- apply(grid.points, MARGIN = 1, FUN = poisson.posterior,
+     y = y.vec, x = x.vec, prior.mean.a = mu.a, prior.var.a = sigma2.a,
+     prior.mean.b = mu.b, prior.var.b = sigma2.b)
```

and then sample (letting R normalize for us) to get the simulated posterior.

I then create a grid that is centered at the MLE and spans an arbitrary 5 standard errors away on both sides.

```
> grid.a <- seq(from = mle[1] - 5 * mle.se[1], to = mle[1] + 5 *
+     mle.se[1], length.out = 200)
> grid.b <- seq(from = mle[2] - 5 * mle.se[2], to = mle[2] + 5 *
+     mle.se[2], length.out = 200)
> grid.points <- expand.grid(grid.a, grid.b)
```

Once we have the grid points, we can evaluate at the unnormalized posterior

```
> post.ord <- apply(grid.points, MARGIN = 1, FUN = poisson.posterior,
+     y = y.vec, x = x.vec, prior.mean.a = mu.a, prior.var.a = sigma2.a,
+     prior.mean.b = mu.b, prior.var.b = sigma2.b)
```

and then sample (letting R normalize for us) to get the simulated posterior.

```
> sample.indices <- sample(1:nrow(grid.points), size = 10000, replace = T,
+     prob = post.ord)
> sim.posterior <- grid.points[sample.indices, ]
```

Posterior Summary

Posterior Summary

Our posterior means are comparable to our MLEs.

Posterior Summary

Our posterior means are comparable to our MLEs.

```
> posterior.means <- colMeans(sim.posterior)
> posterior.means

  Var1   Var2
-1.007  1.207

> mle

(Intercept)        coop
     -1.003       1.207
```

Posterior Summary

Our posterior means are comparable to our MLEs.

```
> posterior.means <- colMeans(sim.posterior)
> posterior.means

  Var1   Var2
-1.007  1.207

> mle

(Intercept)        coop
     -1.003       1.207
```

as well as our posterior standard deviations and our MLE SEs.

Posterior Summary

Our posterior means are comparable to our MLEs.

```
> posterior.means <- colMeans(sim.posterior)
> posterior.means

  Var1   Var2
-1.007  1.207

> mle

(Intercept)        coop
     -1.003       1.207
```

as well as our posterior standard deviations and our MLE SEs.

```
> posterior.sd <- apply(sim.posterior, MARGIN = 2, FUN = sd)
> posterior.sd

  Var1   Var2
0.1466 0.0453

> mle.se

(Intercept)        coop
    0.14602     0.04504
```