Bayesian Basics

Michael Clark Statistician Lead

2016-11-22

Contents

Chapter 1

Home

Chapter 2

Preface

Initial draft posted 2014 Summer. Recent updates (2016) include major overhaul from pdf to web-based presentation (twice!). The last pdf version I made is here if anyone wants it, however, it will no longer be updated. You can download the auto-generated (and ugly) pdf for this document here. The following serves as a practical and applied introduction to Bayesian estimation methods for the uninitiated. The goal is to provide just enough information in a brief format to allow one to feel comfortable exploring Bayesian data analysis for themselves, assuming they have the requisite context to begin with. The idea is to cover a similar amount of material as one would in part of a standard statistics sequence in various applied disciplines where statistics is being introduced in general.

After a conceptual introduction, a fully visible by-hand example is provided using the binomial distribution. After that, the document proceeds to introduce fully Bayesian analysis with the standard linear regression model, as that is the basis for most applied statistics courses and is assumed to be most familiar to the reader. Model diagnostics, model enhancements, and additional modeling issues are then explored. Supplemental materials provide more technical detail if desired, and include a maximum likelihood refresher, overview of programming options in Bayesian analysis, the same regression model using BUGS and JAGS, and 'by-hand' code for the model using the Metropolis-Hastings and Hamiltonian Monte Carlo algorithms.

2.1 Prerequisites

Prerequisites include a basic statistical exposure such as what would be covered in typical (probably graduate) applied science statistics course. At least some familiarity with R is necessary to follow the code (but that itself is not necessary), and one may go through any number of introductions on the web to acquire enough knowledge in that respect. However, note that for the examples here, at least part of the code will employ some Bayesian-specific programming language (e.g. Stan primarily, BUGS and JAGS in the appendix). No attempt is made to teach those languages though, as it would be difficult to do so efficiently in this more conceptually oriented setting. As such, it is suggested that one follow the code as best they can, and investigate the respective manuals, relevant texts, etc. further on their own. Between the text and comments within the code it is hoped that what the code is accomplishing will be fairly clear. However, I provide a set of notes that can serve as an overview of Stan here.

This document relies heavily on Gelman et al. (2013), which I highly recommend, if one is ready for it. Other sources used or particularly pertinent to the material in this document can be found in the references section at the end. Some are more introductory, and which might be more suitable depending on the context you bring. This document was created using Rmarkdown within RStudio.

Color coding:

- emphasis
- package

- function
- object/class
- link

Chapter 3

Introduction

Bayesian analysis is now fairly common in applied work. It is no longer a surprising thing to see it utilized in non-statistical journals, though it is still fresh enough that many researchers feel they have to put 'Bayesian' in the title of their papers when they implement it. However, one should be clear that one doesn't conduct a Bayesian analysis per se. A Bayesian logistic regression is still just logistic regression. The *Bayesian* part comes into play with the perspective on probability that one uses to interpret the results, and in how the estimates are arrived at.

The Bayesian approach itself is very old at this point. Bayes and Laplace started the whole shebang in the 18^{th} and 19^{th} centuries, and even the modern implementation of it has its foundations in the 30s, 40s and 50s of last century[1]. So while it may still seem somewhat newer to applied researchers, much of the groundwork has long since been hashed out, and there is no more need to justify a Bayesian analysis any more than there is to use the standard maximum likelihood approach[2]. While there are perhaps many reasons why the Bayesian approach to analysis did not catch on until relatively recently, perhaps the biggest is simply computational power. Bayesian analysis requires an iterative and time-consuming approach that simply wasn't viable for most applied researchers until modern computers. But nowadays, one can conduct such analysis even on their laptop very easily.

The Bayesian approach to data analysis requires a different way of thinking about things, but its implementation can be seen as an extension of traditional approaches. In fact, as we will see later, it incorporates the very likelihood one uses in traditional statistical techniques. The key difference regards the notion of probability, which, while different than Fisherian or frequentist statistics, is actually more akin to how the average Joe thinks about probability. Furthermore, p-values and intervals will have the interpretation that many applied researchers incorrectly think their current methods provide. On top of this one gets a very flexible toolbox that can handle many complex analyses. In short, the reason to engage in Bayesian analysis is that it has a lot to offer and can potentially handle whatever you throw at it.

As we will see shortly, one must also get used to thinking about distributions rather than fixed points. With Bayesian analysis we are not so much as making guesses about specific values as in the traditional setting, but more so understanding the limits of our knowledge and getting a healthy sense of the uncertainty of those guesses.

3.1 Bayesian Probability

This section will have about all the math there is going to be in this handout and will be very minimal even then. The focus will be on the conceptual understanding though, and subsequently illustrated with a by-hand example in the next section.

[1] Jeffreys, Metropolis etc.

[2] Though some Bayesians might suggest the latter would need *more*.

3.1.1 Conditional probability & Bayes theorem

Bayes theorem is illustrated in terms of probability as follows:

$$p(\mathcal{A}|\mathcal{B}) = \frac{p(\mathcal{B}|\mathcal{A})p(\mathcal{A})}{p(\mathcal{B})}$$

In short, we are attempting to ascertain the conditional probability of $\mathcal{A}$ given $\mathcal{B}$ based on the conditional probability of $\mathcal{B}$ given $\mathcal{A}$ and the respective probabilities of $\mathcal{A}$ and $\mathcal{B}$. This is perhaps not altogether enlightening in and of itself, so we will frame it in other ways, and for the upcoming depictions we will ignore the denominator.

The $\propto$ means 'proportional to'.

$$p(hypothesis|data) \propto p(data|hypothesis)p(hypothesis)$$

In the above formulation, we are trying to obtain the probability of an hypothesis given the evidence at hand (data) and our initial (prior) beliefs regarding that hypothesis. We are already able to see at least one key difference between Bayesian and classical statistics, namely that classical statistics is only concerned with $p(data|hypothesis)$, i.e. if some (null) hypothesis is true, what is the probability I would see data such as that experienced? While useful, we are probably more interested in the probability of the hypothesis given the data, which the Bayesian approach provides.

Here is yet another way to consider this:

$$posterior \propto likelihood * prior$$

For this depiction let us consider a standard regression coefficient b. Here we have a prior belief about b expressed as a probability distribution. As a preliminary example we will assume perhaps that the distribution is normal, and is centered on some value μ_b and with some variance σ_b^2. The likelihood here is the exact same one used in classical statistics- if y is our variable of interest, then the likelihood is $p(y|b)$ as in the standard regression approach using maximum likelihood estimation. What we end up with in the Bayesian context however is not a specific value of b that would make the data most likely, but a probability distribution for b that serves as a weighted combination of the likelihood and prior. Given that posterior distribution for b, we can then get the mean, median, 95% credible interval[3] and technically a host of other statistics that might be of interest to us.

To summarize conceptually, we have some belief about the state of the world, expressed as a mathematical model (such as the linear model used in regression). The Bayesian approach provides an updated belief as a weighted combination of prior beliefs regarding that state and the currently available evidence, with the possibility of the current evidence overwhelming prior beliefs, or prior beliefs remaining largely intact in the face of scant evidence.

$$\text{updated belief} = \text{current evidence} * \text{prior belief or evidence}$$

[3] More on this later.

Chapter 4

A Hands-on Example

4.1 Prior, likelihood, & posterior distributions

The following is an attempt to provide a small example to show the connection between prior distribution, likelihood and posterior distribution. Let's say we want to estimate the probability that a soccer/football player[1] will score a penalty kick in a shootout. We will employ the binomial distribution to model this.

Our goal is to estimate a parameter θ, the probability that the random knucklehead from your favorite football team will score the penalty in a overtime shootout. Let's say it takes 10 shots per team before the game is decided.

We can represent this in R as follows, as well as setup some other things for later.

```
shots = c('goal','goal','goal','miss','miss',
          'goal','goal','miss','miss','goal')

# convert to numeric, arbitrarily picking goal=1, miss=0
shotsNum = ifelse(shots=='goal', 1, 0)
N = length(shots)                       # sample size
nGoal = sum(shots=='goal')              # number of shotsrs goal
nMiss = sum(shots=='miss')              # number of those miss
```

Recall the binomial distribution where we specify the number of trials for a particular observation and the probability of an event. Let's look at the distribution for a couple values for θ equal to .5 and .85 and $N = 10$ observations. We will repeat this 1000 times (histograms not shown).

```
x1 = rbinom(1000, size=10, p=.5)
x2 = rbinom(1000, size=10, p=.85)

mean(x1); hist(x1)
```

```
mean(x2); hist(x2)
```

```
[1] 5.043
[1] 8.569
```

We can see the means are roughly around $N * p$ as we expect with the binomial.

[1] Don't even start, it's always been both 'football' and 'soccer'.

4.2 Prior

For our current situation, we don't know θ and are trying to estimate it. We will start by supplying some possible values.

```
theta = seq(from=1/(N+1), to=N/(N+1), length=10)
```

For the Bayesian approach we must choose a prior distribution representing our initial beliefs about the estimate. I provide three possibilities and note that any one of them would work just fine for this situation. We'll go with a triangular distribution, which will put most of the weight toward values around .5. While we will talk more about this later, I will go ahead and mention that this is where some specifically have taken issue with Bayesian estimation in the past, because this part of the process is too *subjective* for their tastes. Setting aside the fact that subjectivity is an inherent part of the scientific process, and that ignoring prior information (if explicitly available from prior research) would be blatantly unscientific, the main point to make here is that this choice *is not an arbitrary one*. There are many distributions we might work with, but some will be better for us than others. Again, we'll revisit this topic later. Choose the prior that makes most sense to you. If I were thinking logically, I might choose a prior that reflects that all advantage is to a person that spends their entire life kicking a ball, and if they can simply kick to the upper right or left of the goal, even with only moderate pace, simple physics (i.e. that the goalie can never reach the ball) would mean they score 95% of the time under perfect conditions. We might revise it downward to account for unspecified things, e.g. how tired they are, weather etc. A prior based on a beta distribution `beta(9,1)` would be about right. Of course the *data* would eventually suggest something much lower.

```
### prior distribution
# uniform
# pTheta = dunif(theta)

# triangular as in Kruschke
pTheta = pmin(theta, 1-theta)

# beta prior with mean = .5
# pTheta = dbeta(theta, 10, 10)

pTheta = pTheta/sum(pTheta) # Normalize so sum to 1
```

So given some estimate of θ, we have a probability of that value based on our chosen prior.

4.3 Likelihood

Next we will compute the likelihood of the data given some value of θ. The likelihood function for the binomial can be expressed as:

$$p(y|\theta) = \binom{N}{k} \theta^k (1-\theta)^{N-k}$$

where N is the total number of possible times in which the event of interest could occur, and k number of times the event of interest occurs. Our maximum likelihood estimate in this simple setting would simply be the proportion of events witnessed out of the total number of samples[^binomll]. We'll use the formula presented aboveNote that if we had covariates as in a regression model, we would have different estimates of theta for each observation, and thus would calculate each observation's likelihood and then take their product or sum their log values, see the Maximum Likelihhood Review for further details). Even here, if you turn this into binary logistic regression with 10 outcomes of texting vs. not, the 'intercept only' model would be identical to our results here. . Technically, the first term is not required, but it serves to normalize the likelihood as we did with the prior.

```
pDataGivenTheta = choose(N, nGoal) * theta^nGoal * (1-theta)^nMiss
```

4.4 Posterior

Given the prior and likelihood, we can now compute the posterior distribution via Bayes theorem.

```
# first we calculate the denominator from Bayes theorem; this is the marginal
# probability of y
pData = sum(pDataGivenTheta*pTheta)

pThetaGivenData = pDataGivenTheta*pTheta  / pData  # Bayes theorem
```

Now lets examine what all we've got.

theta	prior	likelihood	posterior
0.091	0.033	0.000	0.000
0.182	0.067	0.003	0.002
0.273	0.100	0.024	0.018
0.364	0.133	0.080	0.079
0.455	0.167	0.164	0.203
0.545	0.167	0.236	0.293
0.636	0.133	0.244	0.242
0.727	0.100	0.172	0.128
0.818	0.067	0.069	0.034
0.909	0.033	0.008	0.002

We can see that we've given most of our prior probability to the middle values with probability tapering off somewhat slowly towards either extreme. The likelihood suggests the data is most likely for θ values .55-.64, though as we know the specific maximum likelihood estimate for θ is the proportion for the sample, or .6. Our posterior will fall somewhere between the prior and likelihood estimates, and we can see it has shifted the bulk of the probability slightly away from center of the prior towards a θ value of .6.

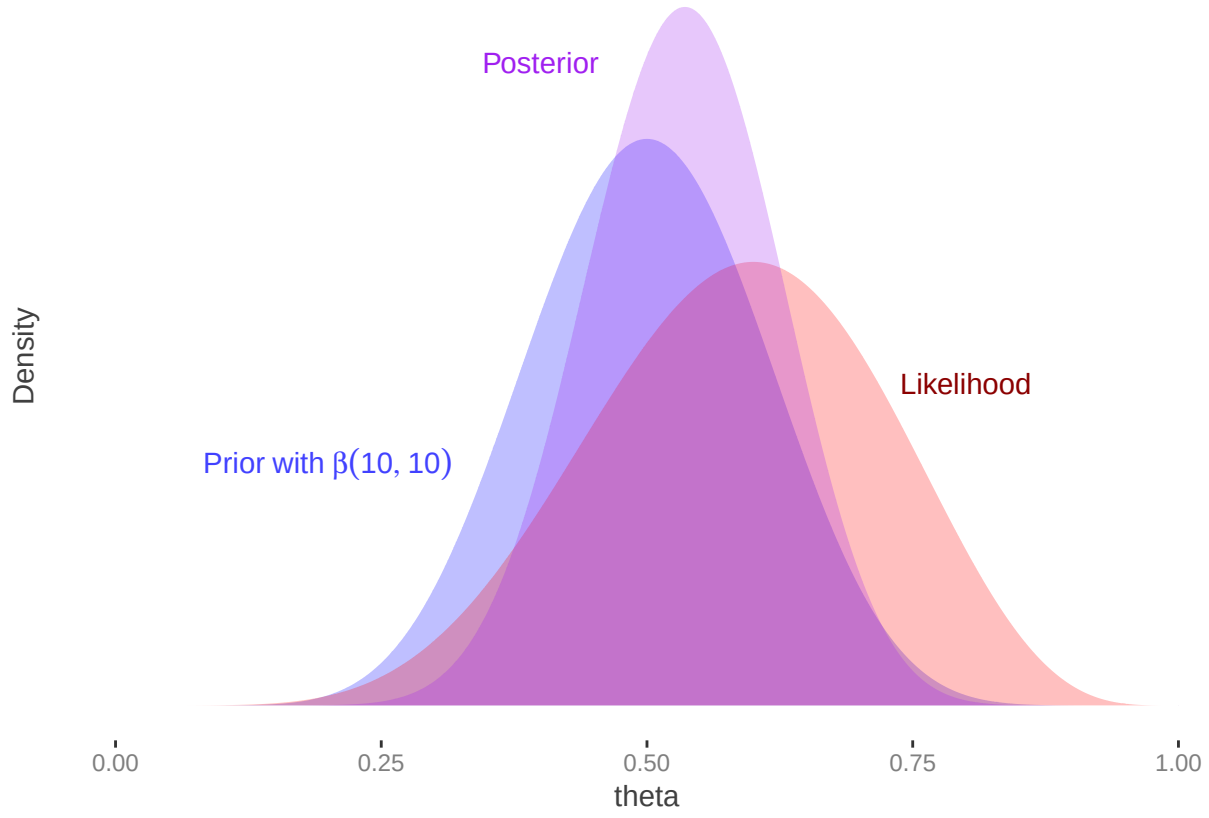
Let's go ahead and see what the mean is: The expected value for a continuous parameter is $\mathrm{E}[X] = \int_{-\infty}^{\infty} xp(x)\mathrm{d}x$, and for a discrete parameter $\mathrm{E}[X] = \sum_{i=1}^{\infty} x_i\, p_i$, i.e. a weighted sum of the possible values times their respective probability of occurrence.

```
posteriorMean = sum(pThetaGivenData*theta)
posteriorMean
```

```
[1] 0.5623611
```

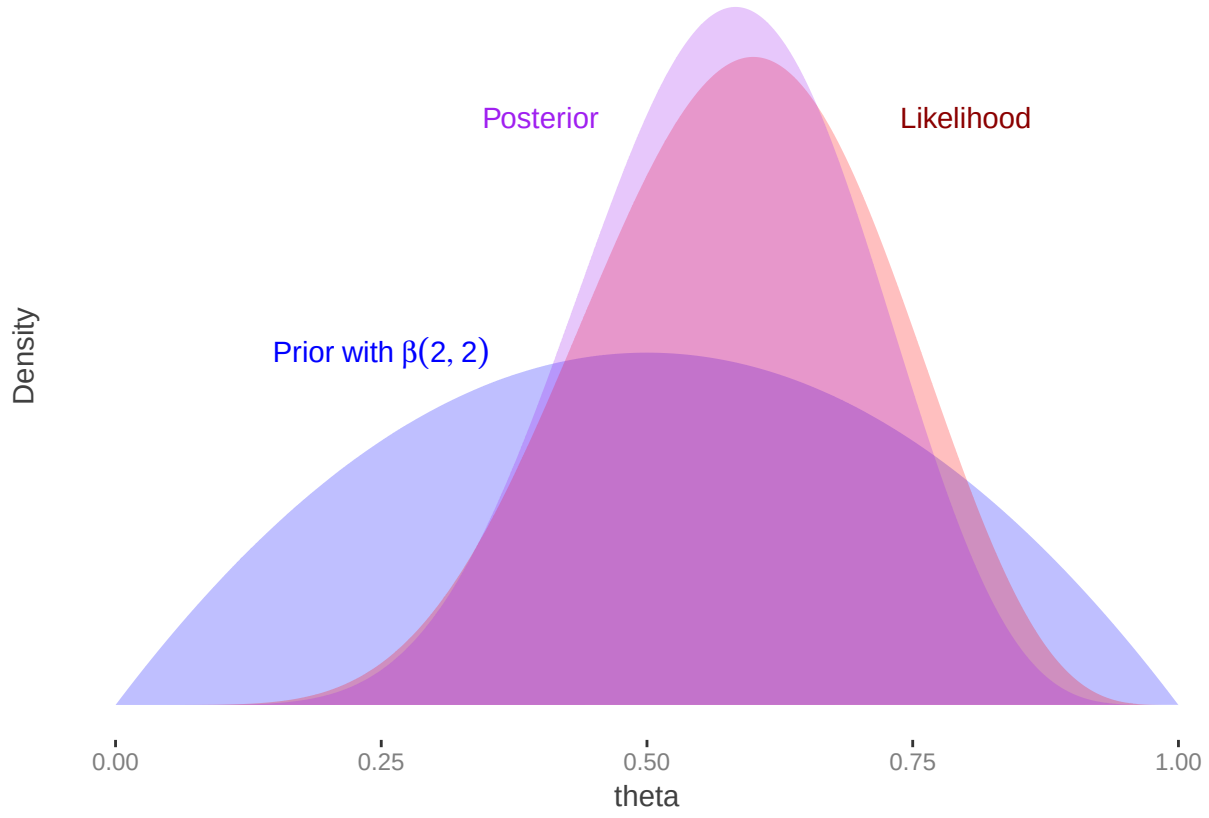
So we start with a prior centered on a value of $\theta = .5$, add data whose ML estimate is $\theta = .6$, and our posterior distribution suggests we end up somewhere in between.

We can perhaps understand this further via the following visuals. In each of these the prior is represented by the blue density, the likelihood by the red, and the posterior by purple. This first is based on a different prior than just used in our example, and instead employs with the beta distribution noted among the possibilities in the code above. While the beta distribution is highly flexible, with shape parameters $\mathcal{A}$ and $\mathcal{B}$ set to 10 and 10 we get a symmetric distribution centered on $\theta = .5$. This would actually be a somewhat stronger prior than we might normally want to use, but serves to illustrate a point. The mean of the beta is $\frac{\mathcal{A}}{\mathcal{A}+\mathcal{B}}$, and thus has a nice interpretation as a prior based on data with sample size equal to $\mathcal{A}+\mathcal{B}$. The posterior distribution that results would have a mean somewhere between the maximum likelihood value and that of the prior. With the stronger prior, the posterior is pulled closer to it.

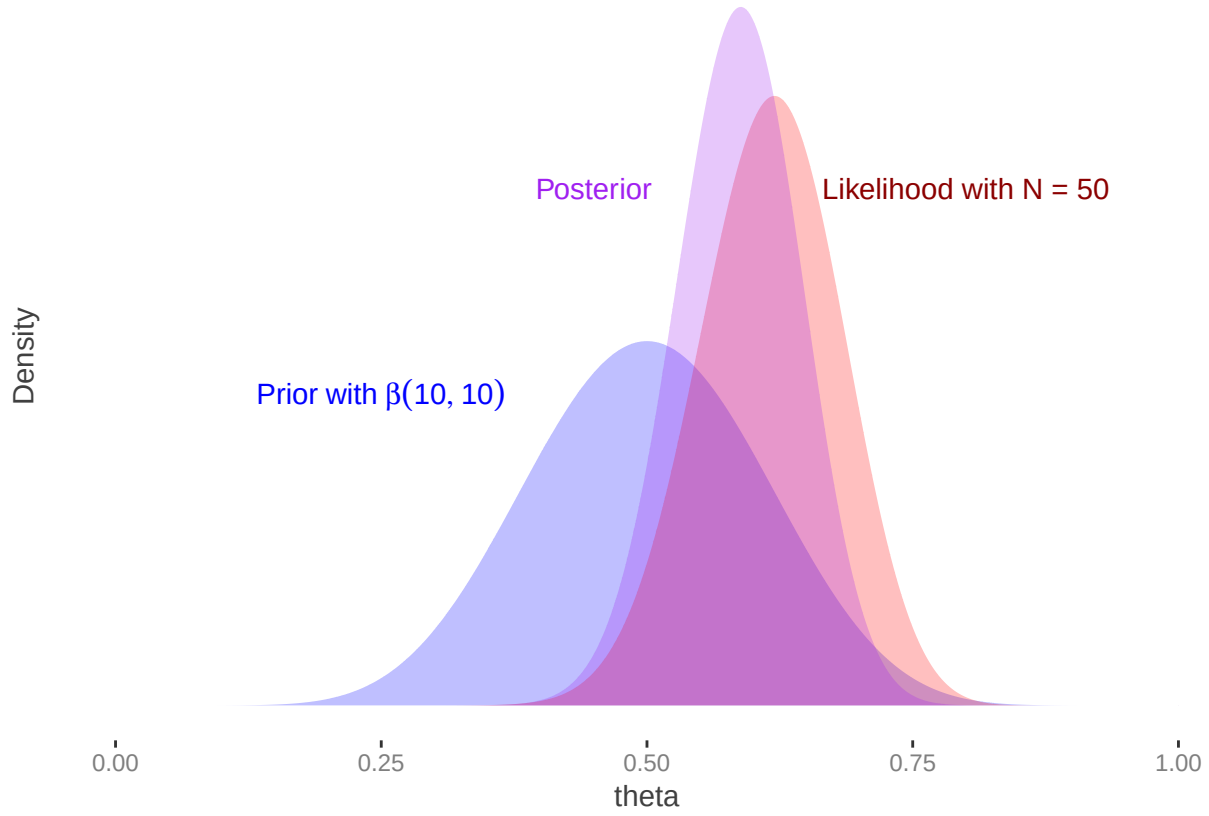


The second utilizes a more diffuse prior of $\beta(2, 2)$ [2]. The result of using the vague prior is that the likelihood gets more weight with regard to the posterior. In fact, if we used a uniform distribution, *we would be doing the equivalent of maximum likelihood estimation.* As such most of the commonly used methods that implement maximum likelihood can be seen as a special case of a Bayesian approach.

[2] $\beta(1, 1)$ is a uniform distribution.



The third graph employs the initial $\beta(10, 10)$ prior again, but this time we add more observations to the data. Again this serves to give more weight to the likelihood, which is what we want. As scientists, we'd want the evidence, i.e. data, to eventually outweigh our prior beliefs about the state of things the more we have of it.



For an interactive demonstration of the above visuals, see this.

4.5 Posterior predictive

At this point it is hoped you have a better understanding of the process of Bayesian estimation. Conceptually, one starts with prior beliefs about the state of the world and adds evidence to one's understanding, ultimately coming to a conclusion that serves as a combination of evidence and prior belief. More concretely, we have a prior distribution regarding parameters, a distribution regarding the data given those parameters, and finally a posterior distribution that is the weighted combination of the two.

However there is yet another distribution of interest to us- the posterior predictive distribution. Stated simply, once we have the posterior distribution for θ, we can then feed new or unobserved data into the process and get distributions for $\tilde{y}$ Mathematically represented as: $p(\tilde{y}|y) = \int p(\tilde{y}|\theta)p(\theta|y)\mathrm{d}\theta$ We can get a sense of the structure of this process via the following table, taken from Gelman:

As in Gelman, we can implement the simulation process given the data and model at hand. Where $\tilde{y}$ can regard *any* potential observation, we can distinguish y^{Rep} as the case where we keep to the current data (if a model had explanatory variables X, the explanatory variables are identical for producing y^{Rep} as they were in modeling y, where $\tilde{y}$ might be based on any values $\tilde{X}$). In other words, y^{Rep} is an attempt to replicate the observed data based on the parameters θ. We can then compare our simulated data to the observed data to see how well they match.

When using typical Bayesian programming languages, we could have simulated values of the posterior predictive distribution given the actual values of θ we draw from the posterior. I provide the results of such a process with the graph at right. Each histogram represents a replication of the data, i.e. y^{Rep}, given an estimate of θ.

Chapter 5

Regression Models

Now armed (hopefully) with a conceptual understanding of the Bayesian estimation process, we will actually investigate a regression model using the Bayesian approach. To keep things simple, we start with a standard linear model for regression. Later, we will show how easy it can be to add changes to the sampling distribution or priors for alternative modeling techniques. But before getting too far, you should peruse the Modeling Languages section of the appendix to get a sense of some of the programming approaches available. We will be using the programming language Stan via R and the associated R package rstan.

5.1 Example: Linear Regression Model

In the following we will have some initial data set up and also run the model using the standard lm function for later comparison. I choose simulated data so that not only should you know what to expect from the model, it can easily be modified to enable further understanding. I will also use some matrix operations, and if these techniques are unfamiliar to the reader, you'll perhaps want to do some refreshing or learning on your own beforehand.

5.2 Setup

First we need to create the data we'll use here and for most of the other examples in this document.

```
# set seed for replicability
set.seed(8675309)

# create a N x k matrix of covariates
N = 250
K = 3

covariates = replicate(K, rnorm(n=N))
colnames(covariates) = c('X1', 'X2', 'X3')

# create the model matrix with intercept
X = cbind(Intercept=1, covariates)

# create a normally distributed variable that is a function of the covariates
coefs = c(5,.2,-1.5,.9)
mu = X %*% coefs
```

```
sigma = 2
y <- rnorm(N, mu, sigma)

# same as
# y = 5 + .2*X1 - 1.5*X2 + .9*X3 + rnorm(N, mean=0, sd=2)

# Run lm for later comparison; but go ahead and examine now if desired
modlm = lm(y~., data=data.frame(X[,-1]))
# summary(modlm)
```

Just to make sure we're on the same page, at this point we have three covariates, and a y that is a normally distributed, linear function of them with standard deviation equal to 2. The population values for the coefficients including the intercept are 5, 0.2, -1.5, and 0.9, though with the noise added, the actual estimated values for the sample are slightly different. Now we are ready to set up an R list object of the data for input into Stan, as well as the corresponding Stan code to model this data. I will show all the Stan code, which is implemented in R via a single character string, and then provide some detail on each corresponding model block. However, the goal here isn't to provide a tutorial on Stan, as you might prefer BUGS or JAGS, and related code for this same model in those languages is provided in the appendix, e.g. [here][Bugs Example]. I don't think there is an easy way to learn these programming languages except by diving in and doing them yourself with models and data you understand. Furthermore, the focus here is on concepts over tools.

The data list for Stan should include any matrix, vector, or value that might be used in the Stan code. For example, along with the data one can include things like sample size, group indicators (e.g. for mixed models) and so forth. Here we can get by with just the N, the number of columns in the model matrix, the target variable and the model matrix itself.

```
# Create the data list object for stan inupt
dat = list(N=N, K=ncol(X), y=y, X=X)
```

Next comes the Stan code. In R2OpenBugs or rjags one would call a separate text file with the code, and one can do the same with rstan[1], but for our purposes, we'll display it within the R code. The first thing to note then is the model code. Next, Stan has programming blocks that have to be called in order. I will have all of the blocks in the code to note their order and discuss each in turn, even though we won't use them all. Anything following a // or #, or between /* */, are comments pertaining to the code. Assignments in Stan are =[2], while distributions are specified with a $\sim$, e.g. `y ~ normal(0, 1)`.

The primary goal here again is to get to the results and beyond, but one should examine the Stan manual for details about the code. In addition, to install rstan one will need to do so via CRAN or Github (quickstart guide). It does not require a separate installation of Stan itself, but it does take a couple steps and does require a C++ compiler[3]. Once you have rstan installed it is called like any other R package as will see shortly.

```
# Create the stan model object using Stan's syntax
stanmodelcode = "
data {                      // Data block
  int<lower=1> N;           // Sample size
  int<lower=1> K;           // Dimension of model matrix
  matrix[N, K] X;           // Model Matrix
  vector[N] y;              // Target variable
}

/*
```

[1] In your own Stan pursuits it's better to have the Stan model as a separate file.

[2] <- is now deprecated, but you may see older examples using it.

[3] You can examine this list for possibilities, or simply install Rtools from the R website (recommended). Note that you may already have one incidentally. Try the Stan test in their getting started guide before downloading one.

```
transformed data {          // Transformed data block. Not used presently.
}
*/

parameters {                // Parameters block
  vector[K] beta;           // Coefficient vector
  real<lower=0> sigma;      // Error scale
}

model {                     // Model block
  vector[N] mu;
  mu <- X * beta;           // Creation of linear predictor

  // priors
  beta ~ normal(0, 10);
  sigma ~ cauchy(0, 5);     // With sigma bounded at 0, this is half-cauchy

  // likelihood
  y ~ normal(mu, sigma);
}

/*
generated quantities {      // Generated quantities block. Not used presently.
}
*/
"
```

5.2.1 Stan Code

The first section is the data block, where we tell Stan the data it should be expecting from the data list. It is useful to put in bounds as a check on the data input, and that is what is being done between the < > (e.g. we should at least have a sample size of 1). The first two variables declared are N and K, both as integers. Next the code declares the model matrix and target vector respectively. As you'll note here and for the next blocks, we declare the type and dimensions of the variable and then its name. In Stan, everything declared in one block is available to subsequent blocks, but those declared in a block may not be used in earlier blocks. Even within a block, anything declared, such as N and K, can then be used subsequently, as we did to specify dimensions.

For a reference, the following is from the Stan manual, variables of interest and the associated blocks where they would be declared:

Variable Kind

Declaration Block

modeled, unmodeled data

data, transformed data

modeled parameters, missing data

parameters, transformed parameters

unmodeled parameters

data, transformed data

generated quantities

transformed data, transformed parameters, generated quantities

loop indices

loop statement

The transformed data block is where you could do such things as log or center variables and similar, i.e. you can create new data based on the input data or just in general. If you are using R though, it would almost always be easier to do those things in R first and just include them in the data list. You can also declare any unmodeled parameters here.

The primary parameters of interest that are to be estimated go in the parameters block. As with the data block you can only declare these variables, you cannot make any assignments. Here we note the β and σ to be estimated, with a lower bound of zero on the latter. In practice you might prefer to split out the intercept or other coefficients to be modeled separately if they are on notably different scales.

The transformed parameters block is where optional parameters of interest might be included. What might go here is fairly open, but for efficiency's sake you will typically want to put things only of specific interest that are dependent on the parameters block. These are evaluated along with the parameters, so if not of special interest you can generate them in the model or generated quantities block to save time.

The model block is where your priors and likelihood are specified, along with the declaration of any variables necessary. As an example, the linear predictor is included here, as it will go towards the likelihood[4]. Note that we could have instead put the linear predictor in the transformed parameters section, but this would slow down the process, and again, we're not so interested in those specific values.

I use a normal prior for the coefficients with a zero mean and a very large standard deviation to reflect my notable ignorance here[5]. For the σ estimate I use a Cauchy distribution[6]. Many regression examples using BUGS will use an inverse gamma prior, which is perfectly okay for this model, though it would not work so well for other variance parameters. Had we not specified anything for the prior distribution for the parameters, vague (discussed more in the [Choice of Prior section][Choice of Prior]), uniform distributions would be the default. The likelihood is specified in pretty much the same manner as we did with R. BUGS style languages would actually use dnorm as in R, though Stan uses 'normal' for the function name.

Finally, we get to the generated quantities, which is kind of a fun zone. *Anything* you want to calculate can go here- predictions on new data, ratios of parameters, how many times a parameter is greater than x, transformations of parameters for reporting purposes, and so forth. We will demonstrate this later.

5.2.2 Running the Model

Now that we have an idea of what the code is doing, let's put it to work. Bayesian estimation, like maximum likelihood, starts with initial guesses as starting points and then runs in an iterative fashion, producing simulated draws from the posterior distribution at each step, and then correcting those draws until finally getting to some target, or *stationary* distribution. This part is key and different from classical statistics. We are aiming for a distribution, not a point estimate.

The simulation process is referred to as Markov Chain Monte Carlo, or MCMC for short. The specifics of this process are what sets many of the Bayesian programming languages/approaches apart, and something we will cover in more detail in a later section (see [Sampling Procedure][Sampling Procedure]). In MCMC, all of the simulated draws from the posterior are based on and correlated with the previous[7], as the process moves along the path toward a stationary distribution. Typically we will allow the process to *warm up*, or rather get a bit settled down from the initial starting point, which might be way off, and thus the subsequent

[4]The position within the model block isn't crucial. I tend to like to do all the variable declarations at the start, but others might prefer to have them under the likelihood heading at the point they are actually used.

[5]By setting the prior mean to zero, this will have the effect of shrinking the coefficients toward zero to some extent. In this sense, it is equivalent to penalized regression in the non-Bayesian setting, ridge regression in particular.

[6]Actually a half-Cauchy as it is bounded to be positive. This is equivalent to a student t with df=1, and there is some tendency of late to use the student t directly with df=3 for slight gains in performance for some models.

[7]In a Markov Chain, θ_t is independent of previous $\theta_{t-2\ldots t_1}$, conditional on θ_{t-1}.

estimates will also be way off for the first few iterations. How far one wants to go down the rabbit hole regarding MCMC is up to the reader. A great many applied researchers do classical statistical analysis without putting much thought into the actual maximum likelihood estimation process, and I suppose one could do so here as well. Rest assured, assuming the model and data are otherwise acceptable, the process will get to where it needs to go. However, as a further check, we will run the whole thing multiple times, i.e. have more than one *chain*. As the chains will start from different places (sometimes only very slightly so), if multiple chains get to the same place in the end, we can feel more confident about our results.

While this process may sound like it might take a long time to complete, for the following you'll note that it will likely take more time for Stan to compile its code to C++ than it will to run the model[8], and on my computer each chain only takes a little less than three seconds. However it used to take a very long time even for a standard regression such as this, and that is perhaps the primary reason why Bayesian analysis only caught on in the last couple decades; we simply didn't have the machines to do it efficiently. Even now though, for highly complex models and large data sets it can still take a long time to run, though typically not prohibitively so.

In the following code, we note the object that contains the Stan model code, the data list, how many iterations we want (12000)[9], how long we want the process to run before we start to keep any estimates (warmup=2000), how many of the post-warmup draws of the posterior we want to keep (thin=10 means every tenth draw), and the number of chains (3). In the end we will have three chains of $1000\frac{12000-2000}{10} = 1000$ draws from the posterior distribution of the parameters. Stan spits out a lot of output to the R console even with `verbose = FALSE`, and I omit it here, but you will see some initial info about the compiling process, updates as each chain gets through 10% of iterations specified in the `iter` argument, and finally an estimate of the elapsed time. You may also see *informational messages* which, unless they are highly repetitive, should not be taken as an error.

```
library(rstan)

### Run the model and examine results ###
fit = stan(model_code=stanmodelcode, data=dat, iter=12000,
```

With the model run, we can now examine the results. In the following, we specify the digit precision to display, which parameters we want (not necessary here), and which quantiles of the posterior draws we want, which in this case are the median and those that would produce a 95% interval estimate.

```
# summary
print(fit, pars=c('beta', 'sigma'), digits=3, prob=c(.025,.5,.975))
```

```
Inference for Stan model: 2c7fc0327865079c07fd921e7cbb2b0a.
3 chains, each with iter=12000; warmup=2000; thin=10;
post-warmup draws per chain=1000, total post-warmup draws=3000.

          mean se_mean    sd   2.5%    50%  97.5% n_eff  Rhat
beta[1]  4.895   0.002 0.130  4.637  4.896  5.142  2951 1.000
beta[2]  0.083   0.002 0.133 -0.183  0.084  0.341  3000 1.001
beta[3] -1.467   0.002 0.127 -1.711 -1.469 -1.215  3000 1.000
beta[4]  0.822   0.002 0.122  0.582  0.824  1.060  2862 1.000
sigma    2.030   0.002 0.094  1.859  2.026  2.224  2886 1.000

Samples were drawn using NUTS(diag_e) at Sat May 07 12:00:28 2016.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

So far so good. The mean estimates reflect the mean of posterior draws for the parameters of interest,

[8] Not usually the case except for simple models with smaller data.
[9] This is way overkill for a simple model like this. One probably would be fine with 500 warmup and 500 iterations.

and are the typical coefficients reported in standard regression analysis. The 95% probability, or, credible intervals are worth noting, because *they are not confidence intervals as you know them.* There is no repeated sampling interpretation here[10]. The probability interval is more intuitive. It means simply that, based on the results of this model, there is a 95% chance the true value will fall between those two points. The other values printed out I will return to in just a moment.

Comparing the results to those from R's lm function, we can see we obtain similar estimates. Had we used uniform priors[11], we would doing essentially the same as what is being done in standard maximum likelihood estimation. Here, we have a decent amount of data for a model that isn't complex, so we would expect the likelihood to notably outweigh the prior, as we demonstrated previously with our binomial example.

```
summary(modlm)
```

```
Call:
lm(formula = y ~ ., data = data.frame(X[, -1]))

Residuals:
    Min      1Q  Median      3Q     Max
-6.8632 -1.4696  0.2431  1.4213  5.0406

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.89777    0.12845  38.131  < 2e-16 ***
X1           0.08408    0.12960   0.649    0.517
X2          -1.46861    0.12615 -11.642  < 2e-16 ***
X3           0.81959    0.12065   6.793 8.21e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.021 on 246 degrees of freedom
Multiple R-squared:  0.4524,    Adjusted R-squared:  0.4458
F-statistic: 67.75 on 3 and 246 DF,  p-value: < 2.2e-16
```

But how would we know if our model was working out okay otherwise? There are several standard diagnostics, and we will talk in more detail about them in the next section, but let's take a look at some presently. In the summary, `se_mean` is the Monte Carlo error, and is an estimate of the uncertainty contributed by only having a finite number of posterior draws. `n_eff` is effective sample size given all chains and essentially accounts for autocorrelation in the chain, i.e. the correlation of the estimates as we go from one draw to the next. It actually doesn't have to be very large, but if it was small relative to the total number of draws desired that might be cause for concern. `Rhat` is a measure of how well chains mix, and goes to 1 as chains are allowed to run for an infinite number of draws. In this case, `n_eff` and `Rhat` suggest we have good convergence, but we can also examine this visually with a traceplot.

```
# Visualize
stan_trace(fit, pars=c('beta[4]'))
```

I only show one parameter for the current demonstration, but one should always look at the traceplots for all parameters. What we are looking for after the warmup period is a "fat hairy caterpillar" or something that might be labeled as "grassy", and this plot qualifies as such[12]. One can see that the estimates from each chain find their way from the starting point to a more or less steady state quite rapidly (warmup perior in gray). Furthermore, all three chains, each noted by a different color, are mixing well and bouncing around the same conclusion. The statistical measures and traceplot suggest that we are doing okay.

[10] A standard confidence implies that if we'd done the study exactly the same over and over, and calculated a confidence interval each time, 95% of them would capture the true value. The one you have is just one from that process.

[11] In Stan code this can be done by not explicitly specifying a prior.

[12] Like all model diagnostics, we aren't dealing with an exact science.

There are other diagnostics available in the coda package, and Stan model results can be easily converted to work with it. The following code demonstrates how to get started.

```
library(coda)
betas = extract(fit, pars='beta')$beta
betas.mcmc = as.mcmc(betas)
plot(betas.mcmc)
```

So there you have it. Aside from the initial setup with making a data list and producing the language-specific model code, it doesn't necessarily take much to running a Bayesian regression model relative to standard models[13]. The main thing perhaps is simply employing a different mindset, and interpreting the results from within that new perspective. For the standard models you are familiar with, it probably won't take too long to be as comfortable here as you were with those, and now you will have a flexible tool to take you into new realms with deeper understanding.

[13]Other R packages would allow for regression models to be specified just like you would with the lm and glm functions. See the rstanarm (from the developers of Stan) and brms especially.

Chapter 6

Summary

Hopefully this document has provided a path toward easing into Bayesian analysis for those that are interested but might not have had the confidence or particular skill set that many texts and courses assume. Conceptually, Bayesian inference can be fairly straightforward, and inferentially is more akin to the ways people naturally think about probability. Many of the steps taken in classical statistical analysis are still present, but have been enriched via the incorporation of prior information, a more flexible modeling scheme, and the ability to enhance even standard analyses with new means of investigation.

Of course it will not necessarily be easy, particularly for complex models, though such models might actually be relatively easier compared to the classical framework. While not necessary for all models, oftentimes the process will involve a more hands-on approach. However this allows for more understanding of the model and its results, and gets easier with practice just like anything else.

You certainly don't have to abandon classical and other methods either. Scientific research involves applying the best tool for the job, and in some cases the Bayesian approach may not be the best fit for a particular problem. But when it is, it's hoped you'll be willing to take the plunge, and know there are many tools and a great community of people to help you along the way.

Best of luck with your research!

http://mc-stan.org/ Main website

http://stan.fit/ The Stan Group

More resources here

Chapter 7

Appendix

NOTE EVAL AND CACHE FALSE UNTIL WE GET TO THIS SECTION. CHange back `startvals[1]` inline to r.

7.1 Maximum Likelihood Review

This is a very brief refresher on maximum likelihood estimation using a standard regression approach as an example, and more or less assumes one hasn't tried to roll their own such function in a programming environment before. Given the likelihood's role in Bayesian estimation and statistics in general, and the ties between specific Bayesian results and maximum likelihood estimates one typically comes across, I figure one should be comfortable with some basic likelihood estimation.

In the standard model setting we attempt to find parameters θ that will maximize the probability of the data we actually observeThe principle of maximum likelihood.. We'll start with an observed random target vector y with $i...N$ independent and identically distributed observations and some data-generating process underlying it $f(\cdot|\theta)$. We are interested in estimating the model parameter(s), θ, that would make the data most likely to have occurred. The probability density function for y given some particular estimate for the parameters can be noted as $f(y_i|\theta)$. The joint probability distribution of the (independent) observations given those parameters, $f(y_i|\theta)$, is the product of the individual densities, and is our *likelihood function*. We can write it out generally as:

$$\mathcal{L}(\theta) = \prod_{i=1}^{N} f(y_i|\theta)$$

Thus the *likelihood* for one set of parameter estimates given a fixed set of data y, is equal to the probability of the data given those (fixed) estimates. Furthermore we can compare one set, $\mathcal{L}(\theta_A)$, to that of another, $\mathcal{L}(\theta_B)$, and whichever produces the greater likelihood would be the preferred set of estimates. We can get a sense of this with the graph to the right, based on a single parameter, Poisson distributed variable. The data is drawn from a variable with mean $\theta = 5$. We note the calculated likelihood increases as we estimate values for θ closer to 5.

For computational reasons we instead work with the sum of the natural log probabilitiesMath refresher on logs: `log(A*B) = log(A)+log(B)`. So summing the log probabilities will result in the same values for θ, but won't result in extremely small values that will break our computer., and thus the *log likelihood*:

$$\ln \mathcal{L}(\theta) = \sum_{i=1}^{N} \ln[f(y_i|\theta)]$$

Concretely, we calculate a log likelihood for each observation and then sum them for the total likelihood for parameter(s) θ.

The likelihood function incorporates our assumption about the sampling distribution of the data given some estimate for the parameters. It can take on many forms and be notably complex depending on the model in question, but once specified, we can use any number of optimization approaches to find the estimates of the parameter that make the data most likely. As an example, for a normally distributed variable of interest we can write the log likelihood as follows:

$$\ln \mathcal{L}(\theta) = \sum_{i=1}^{N} \ln[\frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(y-\mu)^2}{2\sigma^2})]$$

7.1.1 Example

In the following we will demonstrate the maximum likelihood approach to estimation for a simple setting incorporating a normal distribution where we estimate the mean and variance/sd for a set of values yOf course we could just use the sample estimates, but this is for demonstration.. First the data is created, and then we create the function that will compute the log likelihood. Using the built in R distributionsType `?Distributions` at the console for some of the basic R distributions available. makes it fairly straightforward to create our own likelihood function and feed it into an optimization function to find the best parameters. We will set things up to work with the bbmle package, which has some nice summary functionality and other features. However, one should take a glance at optim and the other underlying functions that do the work.

```
# for replication
set.seed(1234)

# create the data
y = rnorm(1000, mean=5, sd=2)
startvals = c(0, 1)

# the log likelihood function
LL = function(mu=startvals[1], sigma=startvals[2]){
  ll = -sum(dnorm(y, mean=mu, sd=sigma, log=T))
  message(paste(mu, sigma, ll))
  ll
}
```

The LL function takes starting points for the parameters as arguments, in this case we call them μ and σ, which will be set to `startvals[1]` and `startvals[2]` respectively. Only the first line (ll = -sum…) is actually necessary, and we use dnorm to get the density for each pointMuch more straightforward than writing the likelihood function as above.. Since this optimizer is by default minimization, we reverse the sign of the sum so as to minimize the negative log likelihood, which is the same as maximizing the likelihood. Note that the bit of other code just allows you to see the estimates as the optimization procedure searches for the best values. I do not show that here but you'll see it in your console.

We are now ready to obtain maximum likelihood estimates for the parameters. For the mle2 function we will need the function we've created, plus other inputs related to that function or the underlying optimizing function used (by default optim). In this case we will use an optimization procedure that will allow us to set a lower bound for σ. This isn't strictly necessary, but otherwise you would get get warnings and possibly lack of convergence if negative estimates for σ were allowedAn alternative approach would be to work with the log of σ which can take on negative values, and then convert it back to the original scale..

```
library(bbmle)
# using optim, and L-BFGS-B so as to contrain sigma to be positive by setting the lower bound at zero
mlnorm =  mle2(LL, method="L-BFGS-B", lower=c(sigma=0))
```

```
mlnorm

# compare to an intercept only regression model
summary(lm(y~1))
```

We can see that the ML estimates are the sameActually there is a difference between the sigma estimates in that OLS estimates are based on a variance estimate divided by $N-1$ while the MLE estimate has a divisor of N. as the intercept only model estimates, which given the sample size are close to the true values.

In terms of the parameters we estimate, in the typical case of two or more parameters we can think of a likelihood surface that represents the possible likelihood values given any particular set of estimates. Given some starting point, the optimization procedure then travels along the surface looking for a minimum/maximum pointWhich is equivalent to finding the point where the slope of the tangent line to some function, i.e. the derivative, to the surface is zero. The derivative, or gradient in the case of multiple parameters, of the likelihood function with respect to the parameters is known as the score function.. For simpler settings such as this, we can visualize the likelihood surface and its minimum point. The optimizer travels along this surface until it finds a minimum. I also plot the the path of the optimizer from a top down view. The large blue dot noted represents the minimum negative log likelihood.

Please note that there are many other considerations in optimization completely ignored here, but for our purposes and the audience for which this is intended, we do not want to lose sight of the forest for the trees. We now move next to a slightly more complicated regression example.

A bit of jitter was added to the points to better see what's going on.

7.2 Linear Model

In the standard regression context, our expected value for the target comes from our linear predictor, i.e. the weighted combination of our explanatory variables, and we estimate the regression weights/coefficients and possibly other relevant parameters. We can expand our previous example to the standard linear model without too much change. In this case we estimate a mean for each observation, but otherwise assume the variance is constant across observations. Again we first construct some data so that we know exactly what to expect, then write out the likelihood function with starting parameters. As we need to estimate our intercept and coefficient for the X predictor (collectively referred to as β), we can can think of our likelihood explicitly as before:

$$\ln \mathcal{L}(\beta, \sigma^2) = \sum_{i=1}^{N} \ln[\frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(y - X\beta)^2}{2\sigma^2})]$$

```
# for replication
set.seed(1234)

# predictor
X = rnorm(1000)

# coefficients for intercept and predictor
theta = c(5,2)

# add intercept to X and create y with some noise
y = cbind(1,X)%*%theta + rnorm(1000, sd=2.5)

regLL = function(sigma=1, Int=0, b1=0){
  coefs = c(Int, b1)
  mu = cbind(1,X)%*%coefs
```

```
  ll = -sum(dnorm(y, mean=mu, sd=sigma, log=T))

  message(paste(sigma, Int, b1, ll))
  ll
}

library(bbmle)
mlopt =  mle2(regLL, method="L-BFGS-B", lower=c(sigma=0))
summary(mlopt)
# plot(profile(mlopt), absVal=F)

modlm = lm(y~X)
summary(modlm)
-2*logLik(modlm)
```

As before, our estimates and final log likelihood value are about where they should be, and reflect the lm output. The visualization becomes more difficult, but we can examine slices similar to the previous plot.

To move to generalized linear models, very little changes of the process outside of the distribution assumed and that we are typically modeling a function of the target variable (e.g. $\log(y) = X\beta; mu = e^{X\beta}$).

7.3 Binomial Likelihood Example

This regards the example seen in the early part of the document with the hands-on example.

```
x1 = rbinom(1000, size=10, p=.5)
x2 = rbinom(1000, size=10, p=.85)

binomLL = function(theta, x) {
  -sum(dbinom(x, size=10, p=theta, log=T))
}

optimize(binomLL, x=x1, lower=0, upper=1); mean(x1)
optimize(binomLL, x=x2, lower=0, upper=1); mean(x2)
```

7.4 Modeling Languages

I will talk only briefly about a couple of the modeling language options available, as you will have to make your own choice among many.

Bibliography

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2013). *Bayesian Data Analysis*. CRC Press, 3rd edition.