Stat 343: Numerical Optimization for Maximum Likelihood Estimation

Evan Ray

Rainfall in Illinois, all storms from 1960 - 1964

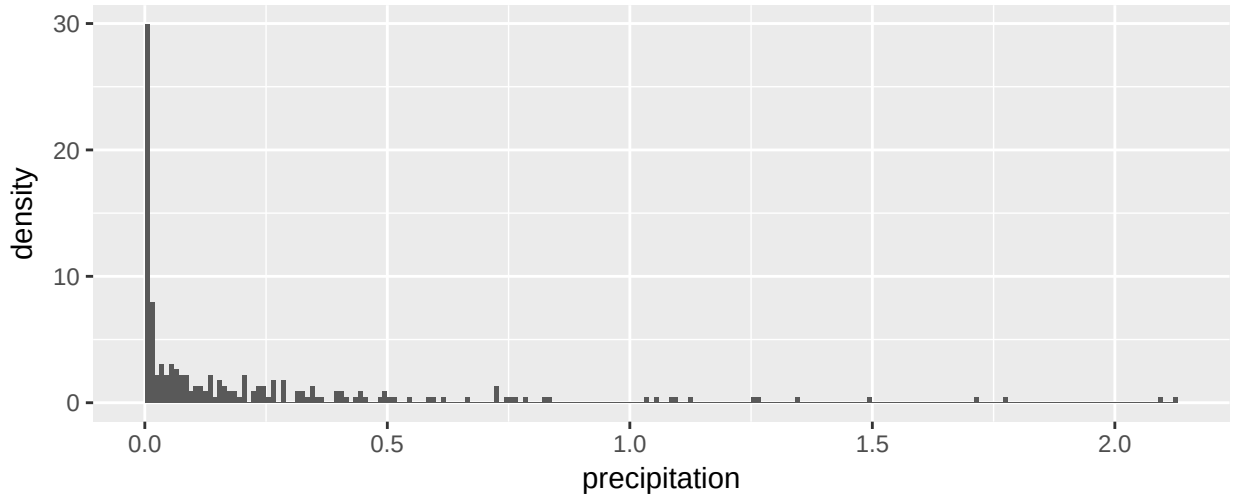
The code below reads in the data and makes an initial plot:

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 3.4.2
```

```
## Warning: package 'purrr' was built under R version 3.4.2
```

```
# Precipitation in Illinois
# Amount of rainfall in 272 storms from 1960 to 1962
il_storms <- bind_rows(
  read_csv("http://www.evanlray.com/stat343_s2018/data/rice/Chapter%2010/illinois60.txt",
    col_names = FALSE),
  read_csv("http://www.evanlray.com/stat343_s2018/data/rice/Chapter%2010/illinois61.txt",
    col_names = FALSE),
  read_csv("http://www.evanlray.com/stat343_s2018/data/rice/Chapter%2010/illinois62.txt",
    col_names = FALSE),
  read_csv("http://www.evanlray.com/stat343_s2018/data/rice/Chapter%2010/illinois63.txt",
    col_names = FALSE),
  read_csv("http://www.evanlray.com/stat343_s2018/data/rice/Chapter%2010/illinois64.txt",
    col_names = FALSE)
)
names(il_storms) <- "precipitation"

ggplot(data = il_storms, mapping = aes(x = precipitation)) +
  geom_histogram(center = 0.005, binwidth = 0.01, mapping = aes(y = ..density..))
```

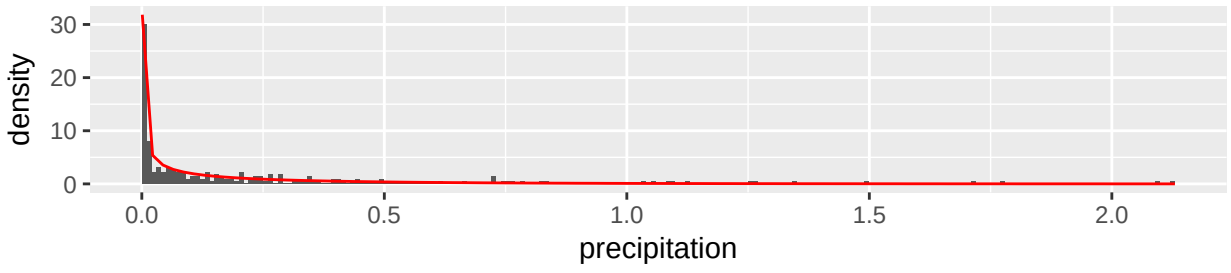


Maximum Likelihood for a Gamma(α, β) by Numerical Optimization

```
# Part 1: Define a function that calculates -1 * the log-likelihood function.
negative_loglikelihood <- function(log_params, x) {
  alpha <- exp(log_params[1])
  beta <- exp(log_params[2])

  result <- -1 * sum(dgamma(x = x, shape = alpha, rate = beta, log = TRUE))
  return(result)
}

# Part 2. Call R's optim function to numerically minimize  -1 * the log-likelihood,
#    which is equivalent to maximizing the log-likelihood.
init_params <- c(0, 0)
optim_result <- optim(
  par = init_params,
  fn = negative_loglikelihood,
  x = il_storms$precipitation)

# Part 3. Extract the parameter estimates and plot
optim_result
```

```
## $par
## [1] -0.8191  0.6752
##
## $value
## [1] -185.3
##
## $counts
## function gradient
##       57       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
log_params_hat <- optim_result$par
alpha_hat <- exp(log_params_hat[1])
beta_hat <- exp(log_params_hat[2])

ggplot(data = il_storms, mapping = aes(x = precipitation)) +
  geom_histogram(center = 0.005, binwidth = 0.01, mapping = aes(y = ..density..)) +
  stat_function(fun = dgamma, args = list(shape = alpha_hat, rate = beta_hat), color = "red")
```



Code breakdown

Here's an outline of and summary of the main points in the code above. There are three main sections:

1. Define a function that calculates -1 * the log-likelihood function.
   - **Why -1 * log-likelihood?**
     - By default the function in R that does optimization finds the minimum of a function.
     - Minimizing the negative of the log-likelihood is equivalent to maximizing the log-likelihood.
   - The first argument to this function has to be the parameter vector; any other arguments are optional
   - **Why set `alpha <- exp(log_params[1])` ?**
     - For the Gamma distribution, both parameters must be positive.
     - It's technically possible to specify bounds on the parameter values, but in practice many optimization algorithms struggle with this
     - Using `alpha <- exp(log_params[1])` enforces the constraint that `alpha` > 0
     - It's important that the transformation be differentiable
2. Call R's 'optim' function to numerically minimize the negative log-likelihood. We have specified 3 arguments to optim:
   (a) `par`: initial values for the first argument to the function we're minimizing
   (b) `fn`: the function we want to minimize, `negative_loglikelihood`
   (c) `x`: the value that should be used for the `x` argument to the `negative_loglikelihood` function when it is called.
3. Extract the parameter estimates and plot the density estimate
   - The return value from `optim` is a list with 5 elements:
     (a) `par`: the final parameter values that minimized the function
     (b) `value`: the minimum value of the function
     (c) `counts`: how many times the function (and the gradient function if provided) were evaluated during the optimization process
     (d) `convergence`: did the optimization process work? If so, this will be 0. Otherwise, an error number that you can look up online.
     (e) `message`: If convergence is something other than 0, a pretty useless message about what happened.
   - Note that we have to re-do any parameter transformations that were performed inside of the function we optimized.

A note on transformations for parameters with constraints

The two most common types of constraints on parameter values are:

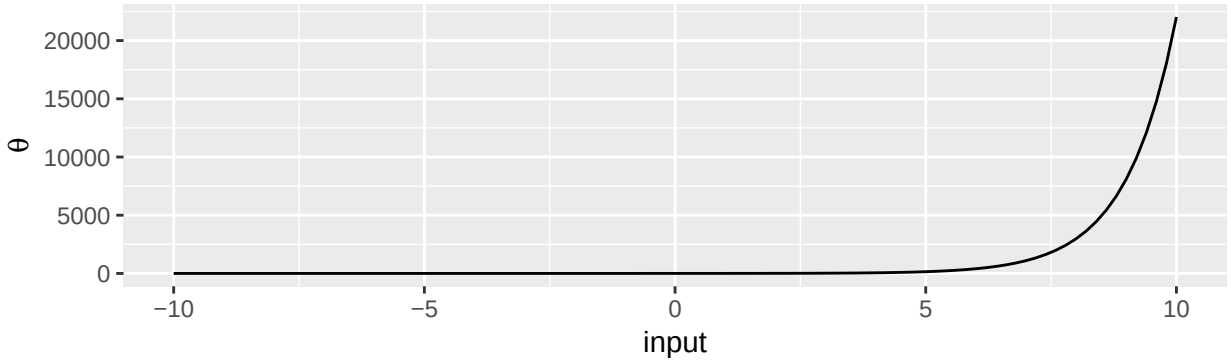
Constraint: $\theta > 0$

This type of constraint often comes up with parameters that play a role like a variance in a distribution; in some sense, the parameter says how "wide" the distribution is, and the "width" has to be positive.

In this case, it's common to use an exponential transformation as in the example above:

$\theta = \exp(\text{input})$, where "input" represents the value passed in to the function.

```
ggplot(data = data.frame(x = c(-10, 10)), mapping = aes(x = x)) +
  stat_function(fun = exp) +
  xlab("input") +
  ylab(expression(theta))
```



Constraint: $0 \leq \theta \leq 1$

This type of constraint comes up with parameters that are a probability, like p in a Binomial distribution.

In this case, it's common to use a logistic transformation:

$\theta = \frac{1}{1+exp(-\text{input})}$

```
logistic <- function(x) {1/(1 + exp(-1 * x))}

ggplot(data = data.frame(x = c(-10, 10)), mapping = aes(x = x)) +
  stat_function(fun = logistic) +
  xlab("input") +
  ylab(expression(theta))
```