Lab Session-4: Numerical Optimization

MATH350 – Statistical Inference

STATISTICS + MACHINE LEARNING + DATA SCIENCE

Dr. Tanujit Chakraborty
Assistant Professor in Statistics at Sorbonne University
tanujit.chakraborty@sorbonne.ae
Course Webpage: https://www.ctanujit.org/SI.html
R Code: https://github.com/tanujit123/MATH350

Let's start

Topics for Today's Session

- No Closed-form Solutions in MLE
- Newton-Raphson Method
- Examples & Implementations
- Gradient Ascent (Descent)
- Examples and Implementations
- References

No closed-form MLEs

- Obtaining MLE estimates for a problem requires maximizing the likelihood. However, it is possible that no analytical form of the maxima is possible!
- This is a common challenge in many models and estimation problems and requires sophisticated optimization tools. Throughout these slides, we will go over some of these optimization methods.

Example: Gamma Distribution

Let $X_1, \ldots, X_n \overset{iid}{\sim} \text{Gamma}(\alpha, 1)$. The likelihood function is

$$L(\alpha \mid x) = \prod_{i=1}^{n} \frac{1}{\Gamma(\alpha)} x_i^{\alpha-1} e^{-x_i} = \frac{1}{\Gamma(\alpha)^n} \prod_{i=1}^{n} x_i^{\alpha-1} e^{-\sum x_i}$$

$$\Rightarrow l(\alpha) := \log L(\alpha \mid x) = -n \log(\Gamma(\alpha)) + (\alpha - 1) \sum_{i=1}^{n} \log x_i - \sum_{i=1}^{n} x_i$$

Taking the first derivative

$$\frac{dl(\alpha)}{d\alpha} = -n \frac{\Gamma'(\alpha)}{\Gamma(\alpha)} + \sum_{i=1}^{n} \log X_i \overset{\text{set}}{=} 0$$

Example: Gamma Distribution

Taking the second derivative

$$\frac{d^2 l(\alpha)}{d\alpha^2} = -n \frac{d^2}{d\alpha^2} \log(\Gamma(\alpha)) < 0 \quad \text{(polygamma function of order 1 is } > 0\text{)}$$

Here, $\frac{d^2}{d\alpha^2} \log(\Gamma(\alpha))$is the polygamma function of order 1, which is always positive (look it up). So we know that the function is concave and a unique maximum exists, but not available in closed form. We cannot get an analytical form of the MLE for α. In such cases, we will use numerical optimization methods.

Numerical optimization methods

We will cover these optimization methods:

- Newton-Raphson method
- Gradient ascent (descent)

Problem

- A general numerical optimization problem is framed in the following way. Let $f(\theta)$ be an objective function that is the main function of interest and needs to be either maximized or minimized. Then, we want to solve the following maximization

$$\theta_* = \arg\max_{\theta} f(\theta)$$

- All the above three algorithms are such that they generate a sequence of $\{\theta_{(k)}\}$ such that the goal is for $\theta_{(k)} \to \theta_*$ in a deterministic manner (non-random convergence).

Problem

- All methods that we will learn will find local optima. Some will guarantee a local maxima, but not guarantee a global maxima, some will guarantee a local optima (so max or min), but not a global maxima. If the objective function is concave, then all methods will guarantee a global maxima!

- Recall that a (univariate) function f is concave if $f'' < 0$ and a (multivariate) function f is concave if its Hessian is negative definite: for all $a \neq 0 \in \mathbb{R}^p$,

$$a^T \nabla^2 f a < 0.$$

Newton-Raphson's method

- To solve this optimization problem, consider starting at a point $\theta_{(0)}$. Then subsequent elements of the sequence are determined in the following way. Suppose that the objective function f is such that a second derivative exists.
- Since $f(\theta)$ is maximized at the unknown $\theta_*, f'(\theta_*) = 0$. Applying first order Taylor's expansion (and ignoring higher orders) to $f'(\theta_*)$ about the current iterate $\theta_{(k)}$

$$f'(\theta_*) \approx f'(\theta_{(k)}) + (\theta_* - \theta_{(k)}) f''(\theta_{(k)}) = 0$$

$$\Rightarrow \theta_* \approx \theta_{(k)} - \frac{f'(\theta_{(k)})}{f''(\theta_{(k)})}$$

where the approximation is best when $\theta_{(k)} = \theta^*$ and the approximation is weak when $\theta_{(k)}$ is far from θ_*. Thus, if we start from an arbitrary point using successive updates of the right-hand side, we will get closer and closer to θ_*.

Newton-Raphson's method

- The Newton-Raphson method is

$$\theta_{(k+1)} = \theta_{(k)} - \frac{f'\left(\theta_{(k)}\right)}{f''\left(\theta_{(k)}\right)}$$

- This works because when $f'\left(\theta_k\right) < 0$, the function is increasing at $\theta_{(k)}$, and thus Newton-Raphson increases $\theta_{(k)}$; and vice-versa. You stop iterating when $\left|\theta_{(k+1)} - \theta_{(k)}\right| < \epsilon$ for some chosen tolerance ϵ.

- *If the objective function is concave, the Newton-Raphson method will converge to the global maxima. Otherwise, it converges to a local optima or diverges!*

Example: Gamma distribution continuted

Our objective function is the log-likelihood:

$$f(\alpha) = -n \log(\Gamma(\alpha)) + (\alpha - 1) \sum_{i=1}^{n} \log x_i - \beta \sum x_i$$

First derivative

$$f'(\alpha) = -n \frac{\Gamma'(\alpha)}{\Gamma(\alpha)} + \sum_{i=1}^{n} \log X_i$$

Second derivative

$$f''(\alpha) = -n \frac{d^2}{d\alpha^2} \log(\Gamma(\alpha)) < 0$$

Thus the log-likelihood is concave, which implies there is a global maxima!
The Newton-Raphson algorithm will converge to this global maxima.
Start with a reasonable starting value: α_0. Then iterate with

$$\alpha_{(k+1)} = \alpha_{(k)} - \frac{f'\left(\alpha_{(k)}\right)}{f''\left(\alpha_{(k)}\right)}$$

Gamma Distribution Implementation

Polygamma functions are in *psi* function in the 'pracma' R package.
What is a good starting value α_0? Well, we know that the mean of a $\text{Gamma}(\alpha, 1)$ is α, so a good starting value is $\alpha_0 = n^{-1} \sum_{i=1}^{n} X_i$.

```
### MLE for Gamma(alpha, 1)
set.seed(100)
library(pracma) #for psi function
# Original data sample size is small at first.
# The NR method estimates the MLE. Here the blue and red lines will not match
# because the data is not large enough for the consistency of the MLE to kick in.
alpha <- 5 #true value of alpha
n <- 10 # actual data size is small first
dat <- rgamma(n, shape = alpha, rate = 1)
alpha_newton <- numeric()
epsilon <- 1e-8 #some tolerance level preset
alpha_newton[1] <- 2 #alpha_0
count <- 1
tol <- 100 # large number
```

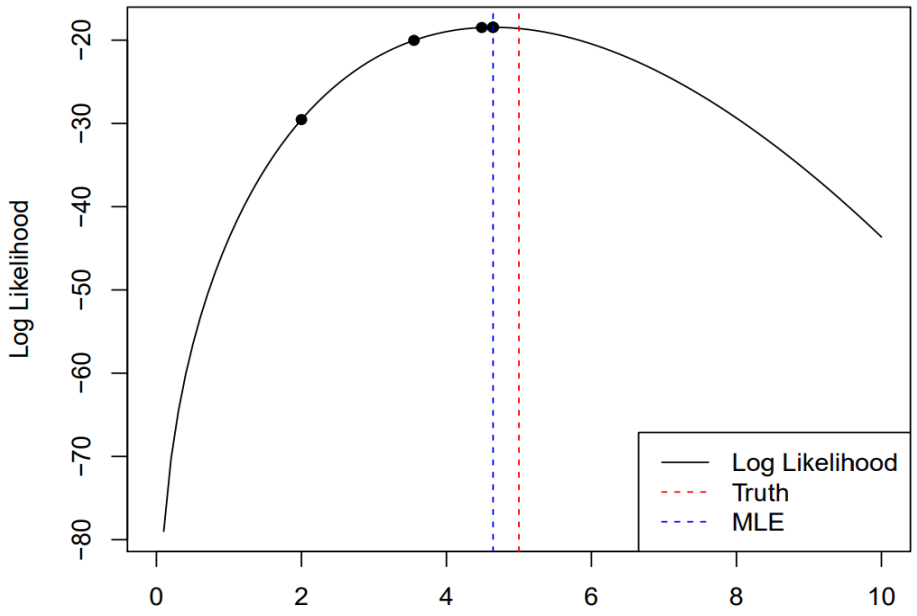
Gamma Distribution Implementation

```
while(tol > epsilon)
{
count <- count + 1
#first derivative
f.prime <- -n*psi(k = 0, alpha_newton[count - 1]) + sum(log(dat))
#second derivative
f.dprime <- -n*psi(k = 1, alpha_newton[count - 1])
alpha_newton[count] <- alpha_newton[count - 1] - f.prime/f.dprime
tol <- abs(alpha_newton[count] - alpha_newton[count-1])
}
alpha_newton
# [1] 2.000000 3.552357 4.487264 4.640581 4.643535 4.643536 4.643536
#Plot the log.likelihood for different values of alpha
alpha.grid <- seq(0, 10, length = 100)
log.like <- numeric(length = 100)
```

Gamma Distribution Implementation

```
for(i in 1:100)
{
    log.like[i] <- sum(dgamma(dat, shape = alpha.grid[i], log = TRUE))
}
plot(alpha.grid, log.like, type = 'l', xlab = expression(alpha), ylab =
"Log Likelihood")
abline(v = alpha, col = "red", lty = 2)
for(t in 1:count)
{
    points(alpha_newton[t], sum(dgamma(dat, shape = alpha_newton[t], log =
    TRUE)), pch = 16)
}
abline(v = tail(alpha_newton[count]), col = "blue", lty = 2)
legend("bottomright", legend = c("Likelihood", "Truth", "MLE"), lty =
c(1,2,2), col = c("black", "red", "blue"))
```

Gamma Distribution Results



Note the impact of the sample size of the original data. The Newton-Raphson algorithm converges to the MLE. If the data size is small, the MLE may not be close to the truth. This is why we see that the blue and red lines are far from each other. However, when increase the observed data to be 1000 observations, the consistency of the MLE should kick in and we expect to see the blue and red lines to be similar (below).

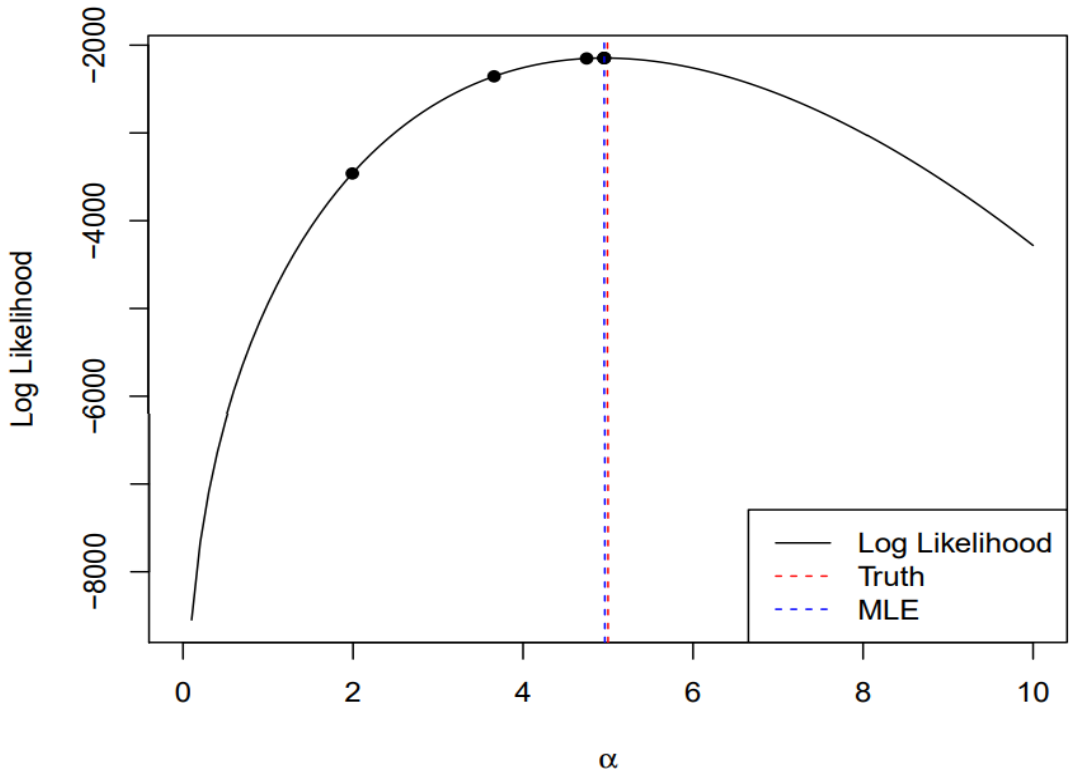
Gamma Distribution Implementation

```
# Increasing original data sample size. Now the MLE is closer to the "truth"
# and our NR method obtains the MLE. Blue and red lines should match a lot
# Randomly generate data
alpha <- 5 #true value of alpha
n <- 1000 # actual data size is small first
dat <- rgamma(n, shape = alpha, rate = 1)
alpha_newton <- numeric()
epsilon <- 1e-8 #some tolerance level preset
alpha_newton[1] <- 2 #alpha_0
count <- 1
tol <- 100 # large number
while(tol > epsilon)
{
count <- count + 1
f.prime <- -n*psi(k = 0, alpha_newton[count - 1]) + sum(log(dat))
f.dprime <- -n*psi(k = 1, alpha_newton[count - 1])
alpha_newton[count] <- alpha_newton[count - 1] - f.prime/f.dprime
tol <- abs(alpha_newton[count] - alpha_newton[count-1])
}
```

Gamma Distribution Implementation

```
alpha_newton
# [1] 2.000000 3.666788 4.755252 4.957521 4.962359 4.962361 4.962361
#Plot the log.likelihood for different values of alpha
alpha.grid <- seq(0, 10, length = 100)
log.like <- numeric(length = 100)
for(i in 1:100)
{
log.like[i] <- sum(dgamma(dat, shape = alpha.grid[i], log = TRUE))
}
plot(alpha.grid, log.like, type = 'l', xlab = expression(alpha), ylab =
"Log Likelihood")
abline(v = alpha, col = "red", lty = 2)
for(t in 1:count)
{
points(alpha_newton[t], sum(dgamma(dat, shape = alpha_newton[t], log =
TRUE)), pch = 16)
}
abline(v = tail(alpha_newton[count]), col = "blue", lty = 2)
legend("bottomright", legend = c("Likelihood", "Truth", "MLE"), lty =
c(1,2,2), col = c("black", "red", "blue"))
```

Gamma Distribution Results



Example: Location Cauchy distribution

Consider the location Cauchy distribution with mode at $\mu \in \mathbb{R}$. The goal is to find the MLE for μ.

$$f(x \mid \mu) = \frac{1}{\pi} \frac{1}{(1 + (x - \mu)^2)}$$

First, we find the log-likelihood

$$L(\mu \mid X) = \prod_{i=1}^{n} f(X_i \mid \mu) = \pi^{-n} \prod_{i=1}^{n} \frac{1}{1 + (x_i - \mu)^2}$$

$$\Rightarrow l(\mu) := \log L(\mu \mid X) = -n \log \pi - \sum_{t=1}^{n} \log \left(1 + (X_i - \mu)^2\right) = f(\mu).$$

It is evident that a closed form solution is difficult. So, we find the derivates.

$$f'(\mu) = 2 \sum_{i=1}^{n} \frac{X_i - \mu}{1 + (X_i - \mu)^2}$$

$$f''(\mu) = 2 \sum_{i=1}^{n} \left[2 \frac{(X_i - \mu)^2}{\left[1 + (X_i - \mu)^2\right]^2} - \frac{1}{1 + (X_i - \mu)^2} \right],$$

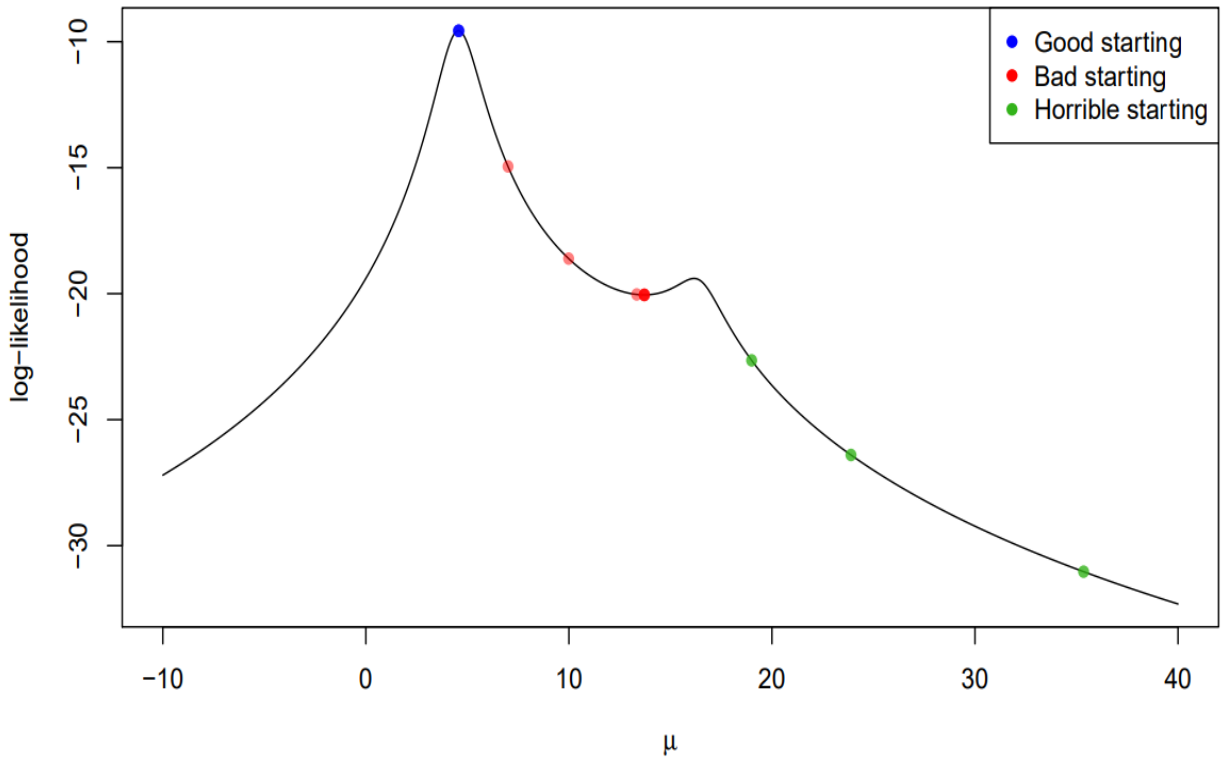
Example: Location Cauchy distribution

which may be positive or negative. So this is not a concave function, so we will be careful in choosing starting values.

1. Set $\mu_0 = \text{Median}\,(X_i)$ since the mean of Cauchy does not exist and the Cauchy centered at μ is symmetric around μ.
2. Determine:
$$\mu_{(k+1)} = \mu_{(k)} - \frac{f'\left(\mu_{(k)}\right)}{f''\left(\mu_{(k)}\right)}$$
3. Stop when $\left|\mu_{(k+1)} - \mu_{(k)}\right| < \epsilon$ for a chosen tolerance level ϵ.

Below is a figure of what the loglikelihood looks like and the behavior of the Newton-Raphson algorithm for different starting values. It shows that choosing a good starting value is very important.

Cauchy Distribution Results



Newton-Raphson in Higher Dimensions

- The NR method can be found in the same way using the multivariate Taylor's expansion. Let $\theta = (\theta_1, \theta_2, \ldots, \theta_p)$. Then first let ∇f denote the gradient of f and $\nabla^2 f$ denote the Hessian. So

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial \theta_1} \\ \vdots \\ \frac{\partial f}{\partial \theta_p} \end{bmatrix} \quad \text{and} \quad \nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial \theta_1^2} & \frac{\partial^2 f}{\partial \theta_1 \theta_2} & \cdots \\ \vdots & \vdots & \vdots \\ \frac{\partial^2 f}{\partial \theta_p \theta_1} & \cdots & \frac{\partial^2 f}{\partial \theta_p^2} \end{bmatrix}$$

- Then, the function f is concave if $\left|\nabla^2 f\right| < 0$, where $|\cdot|$ is the determinant. So always check that first to know if there is a unique maximum.

Newton-Raphson in Higher Dimensions

- Using a similar multivariate Taylor series expansion, the Newton-Raphson update solves the system of linear equations

$$\nabla f\left(\theta_{(k)}\right) + \nabla^2 f\left(\theta_k\right)\left(\theta_{k+1} - \theta_k\right) = 0.$$

- If $\nabla^2 f\left(\theta_{(k)}\right)$ is invertible, then you have that

$$\theta_{(k+1)} = \theta_{(k)} - \left[\nabla^2 f\left(\theta_{(k)}\right)\right]^{-1} \nabla f\left(\theta_{(k)}\right)$$

- Iterations are stopped with when $\left\|\theta_{(k+1)} - \theta_{(k)}\right\| < \epsilon$ for some chosen tolerance level, ϵ.

Questions

1. Can you implement the Newton-Raphson procedure for linear regression and ridge regression?
2. What are some of the issues in implementing Newton-Raphson? Can we use it for any problem?
3. If the function is not concave and different starting values yield convergence to different points (or divergence), then what do we do?

Gradient Ascent (Descent)

- For concave objective functions, Newton-Raphson is essentially the "best" algorithm. However, one significant flaw in the algorithm is that information about the Hessian is required to implement it. The Hessian (or the double derivative) may be unavailable or difficult to calculate in some situations (e.g., polygamma).
- In such a case, gradient ascent (or gradient descent if the problem is a minimizing problem) is a useful alternative as it does not require the gradient.
- Consider the objective function $f(\theta)$ that we want to maximize and suppose θ_* is the true maximum. Then, by the Taylor series approximation at a fixed θ_0

$$f(\theta) \approx f(\theta_0) + f'(\theta_0)(\theta - \theta_0) + \frac{f''(\theta_0)}{2}(\theta - \theta_0)^2$$

Gradient Ascent (Descent)

- If $f''(\theta)$ is unavailable, consider assuming that the double derivative is a negative constant. That is, that f is quadratic and concave. Then for a $t > 0$,

$$f(\theta) \approx f(\theta_0) + f'(\theta_0)(\theta - \theta_0) - \frac{1}{2t}(\theta - \theta_0)^2$$

- Maximizing $f(\theta)$ and using this approximation would imply maximizing the right-hand side. Taking the derivative with respect to θ setting it to zero:

$$f'(\theta_0) - \frac{\theta - \theta_0}{t} \stackrel{set}{=} 0 \Rightarrow \theta = \theta_0 + tf'(\theta_0)$$

Gradient Ascent (Descent)

- The algorithm essentially does a locally quadratic approximation at the current point $\theta_{(k)}$ and then maximizes that quadratic equation. The value of t indicates how far we want to jump and is a tuning parameter. If t is large, we take big jumps instead of t small, where the jumps are smaller.
- Given a $\theta_{(k)}$, the gradient ascent algorithm does the update

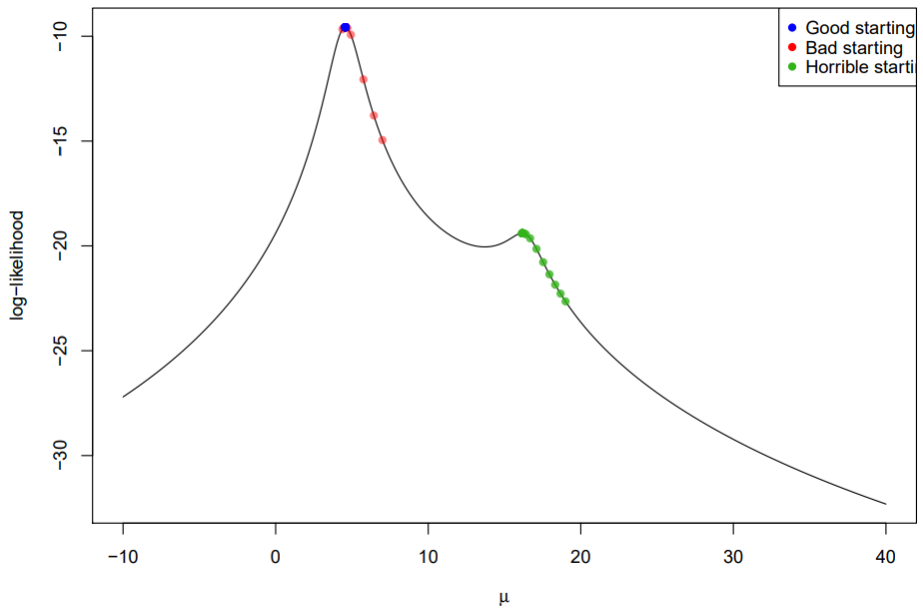
$$\theta = \theta_{(k)} + tf'\left(\theta_{(k)}\right)$$

for $t > 0$. The iteration can be stopped when $\left|\theta_{(k+1)} - \theta_{(k)}\right| < \epsilon$ for $\epsilon > 0$.

- For concave functions, gradient ascent converges to the global maxima. In general, gradient ascent always converges to a local maxima.

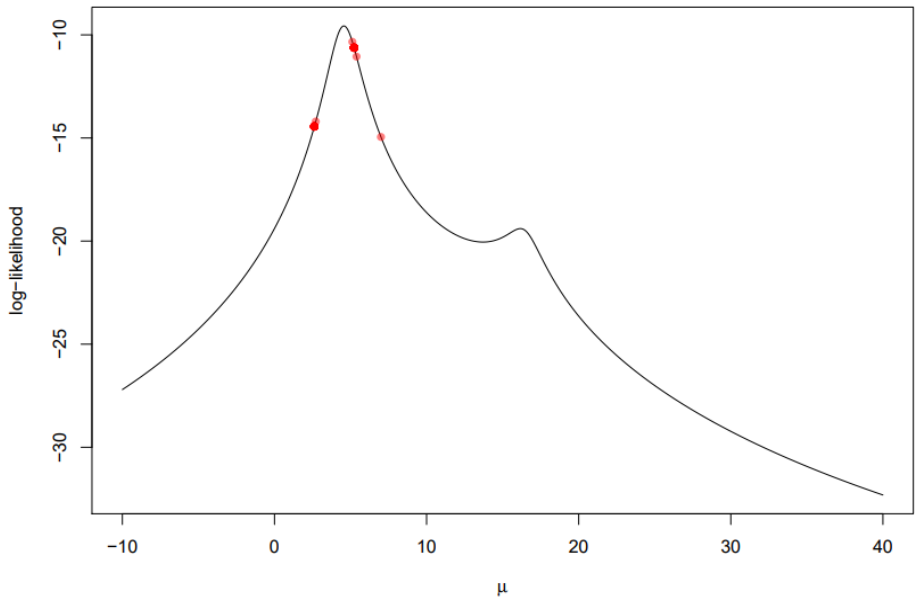
Location Cauchy

Recall the location Cauchy example where the likelihood is not concave. We can implement gradient ascent here and since gradient ascent always converges to a local maxima, it does something more reasonable here (we set $t = .3$)



Location Cauchy

If we rerun this with the red starting values and $t = 1$, this is too much of a jump size, and the optimization starts oscillating between two points.



So it is really important to try different step sizes t when implementing gradient ascent. There are other solutions to this like adaptive step sizes which we will not discuss.

Gradient Ascent in Higher Dimensions

By a multivariate Taylor series expansion, we can obtain a similar motivation and the iteration in the algorithm is:

$$\theta_{(k+1)} = \theta_{(k)} + t\nabla f\left(\theta_{(k)}\right)$$

Example: Logistic regression We have studied linear regression for modeling continuous responses. But when Y is a response of 1s and 0s (Bernoulli) then assuming the Y s are normally distributed is not appropriate. Instead, when the ith covariate is $\left(x_{i1}, \ldots, x_{ip}\right)^T$, then for $\beta \in \mathbb{R}^p$ logistic regression assumes the model

$$Y_i \sim \text{Bern}\left(\frac{e^{x_i^T\beta}}{1+e^{x_i^T\beta}}\right)$$

In other words, the probability that any response takes the value 1 is

$$\Pr\left(Y_i = 1\right) = \frac{e^{x_i^T\beta}}{1+e^{x_i^T\beta}} := p_i$$

Gradient Ascent in Higher Dimensions

Our goal is to obtain the MLE of β. As usual, first, we write down the log-likelihood.

$$L(\beta \mid Y) = \prod_{i=1}^{n} (p_i)^{y_i} (1 - p_i)^{1-y_i}$$

$$\Rightarrow l(\beta) = \sum_{i=1}^{n} y_i \log p_i + \sum_{i=1}^{n} (1 - y_i) \log (1 - p_i)$$

$$= \sum_{i=1}^{n} \log (1 - p_i) + \sum_{i=1}^{n} y_i (\log p_i - \log (1 - p_i))$$

$$= -\sum_{i=1}^{n} \log \left(1 + \exp \left(x_i^T \beta\right)\right) + \sum_{i=1}^{n} y_i x_i^T \beta$$

Gradient Ascent in Higher Dimensions

Taking derivative:

$$\nabla l(\beta) = \sum_{i=1}^{n} x_i \left[y_i - \frac{e^{x_i^T \beta}}{1 + e^{x_i^T \beta}} \right] \stackrel{set}{=} 0$$

An analytical solution here is not possible, thus a numerical optimization tool is required.

Note: You can verify by studying the Hessian that this likelihood function is indeed concave and thus we can use either Newton-Raphson or Gradient Ascent successfully. We will use gradient ascent, and you can implement the NR method on your own.

MLE for Logistic Regression using Gradient Descent

```
library(mcmc) #to load a dataset
data(logit)
head(logit) # y is response and 4 covariates
# y x1 x2 x3 x4
# 1 0 -0.162 -0.348 -0.524 -0.312
# 2 1 0.187 -0.410 0.362 -0.366
# 3 1 0.160 1.649 -0.664 0.051
# 4 1 1.536 0.084 0.403 1.732
# 5 1 -0.162 -0.061 0.192 0.256
# 6 0 -2.855 -3.341 -2.549 -1.461
y <- logit$y
X <- as.matrix(logit[, 2:5])
p <- dim(X)[2]
f.gradient <- function(y, X, beta)
{
beta <- matrix(beta, ncol = 1)
pi <- exp(X %*% beta) / (1 + exp(X%*%beta))
rtn <- colSums(X* as.numeric(y - pi))
return(rtn)
}
```

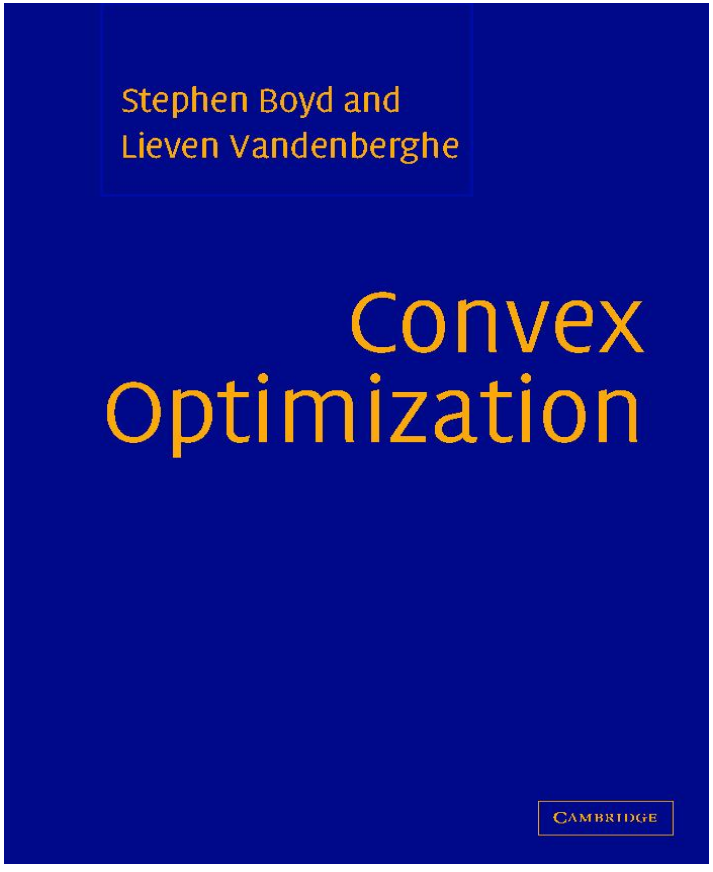
MLE for Logistic Regression using Gradient Descent

```
store.beta <- matrix(0, nrow = 1, ncol = p
beta_k <- rep(0, p) # start at all 0s
t <- .1
tol <- 1e-5
iter <- 0
diff <- 100
while((diff > tol) && iter < 100) #not too many iterations
{
    iter <- iter+1
    old <- beta_k
    beta_k = old + t* f.gradient(y = y, X= X, beta = old)
    store.beta <- rbind(store.beta, beta_k)
    diff <- sum( (beta_k - old)^2)
}
iter # number of iterations
#[1] 10
beta_k # last step in the optimization
# x1 x2 x3 x4
#0.7169059 0.8792911 0.4231445 0.5884162
```

Questions to think about

1. Can you implement Newton-Raphson for the logistic regression problem? Which of the two algorithms is better?
2. What might be a way to adapt the step size t in a problem?

Reference



https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf