Chapter 2, Optimization

January 31, 2018

1 Introduction

- Estimating parameters in statistics often involves minimizing or maxing a function.
  - MLE: the parameter estimates maximize the likelihood function given the observed data
  - Least Square estimation: minimize the sum of residual squares.
- In many situations, it is not possible to work this out analytically, so we must turn to numerical solutions.

Example 1.1 Maximum likelihood for the Gamma distribution. We observe a random sample $\mathbf{X} = (X_1, \ldots, X_n)'$ following a Gamma density given by

$$f(x; \alpha, \beta) = \frac{1}{\beta^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{-x/\beta},$$

where $\Gamma(\alpha) = \int_0^\infty t^{\alpha-1} e^{-t}\, dt$. Find the MLE for (α, β)

- The log-likelihood can be written

$$\log f(\mathbf{X}; \alpha, \beta) = -n\alpha \log \beta - n \log \Gamma(\alpha) + (\alpha - 1) \sum_{i=1}^n \log(X_i) - \frac{1}{\beta} \sum_{i=1}^n X_i.$$

- Taking the derivative with respect to β gives

$$\frac{\partial \log f(\mathbf{X}; \alpha, \beta)}{\partial \beta} = -\frac{n\alpha}{\beta} + \frac{1}{\beta^2} \sum_{i=1}^n X_i.$$

So, given α, the MLE of β is

$$\hat{\beta} = \frac{\sum_{i=1}^{n} X_i}{n\alpha} = \bar{X}/\alpha.$$

- Taking the derivative with respect to α gives

$$\frac{\partial \log f(\mathbf{X}; \alpha, \beta)}{\partial \alpha} = -n \log \beta - n \frac{\partial}{\partial \alpha} \log \Gamma(\alpha) + \sum_{i=1}^{n} \log(X_i).$$

No analytic expression for α in terms of the data. But plugging in $\bar{X}/\alpha$ in place of β, $\widehat{\alpha}$ should solve

$$-\log \bar{X} + \log \alpha - \frac{\partial}{\partial \alpha} \log \Gamma(\alpha) + \frac{1}{n} \sum_{i=1}^{n} \log(X_i) = 0. \qquad (1)$$

1.1 Solving a nonlinear equation of one variable

Objective: Find the value of θ for which $f(\theta) = 0$. We assume that θ is the unique solution to the equation.

1.1.1 *Bisection algorithm*

Basic ideas: Suppose that we compute $f(a)$ and $f(b)$ for some given a and b, and that these two values have opposite sign. Then if f is continuous, we know that the value of θ for which $f(\theta) = 0$ must be between a and b.

1. Choose a starting interval $[a^{(0)}, b^{(0)}]$, and split the interval into half, i.e. split point $c^{(0)} = (a + b)/2$
2. If $f(a^{(0)})f(c^{(0)} > 0$ then $a^{(1)} = c^{(0)}$, $b^{(1)} = b^{(0)}$; otherwise $a^{(1)} = a^{(0)}$, $b^{(1)} = c^{(0)}$

3. repeat until $|f(c^{(k)})| <$ tol

Back to the earlier example of Gamma MLE. We need to find a zero for:

```
> dloglikgamma = function(x, a) {
  return(-log(mean(x)) + log(a) - digamma(a) + mean(log(x)))
}
```

The built-in `digamma` function in `R` gives $\frac{d}{d\alpha} \log \Gamma(\alpha)$.

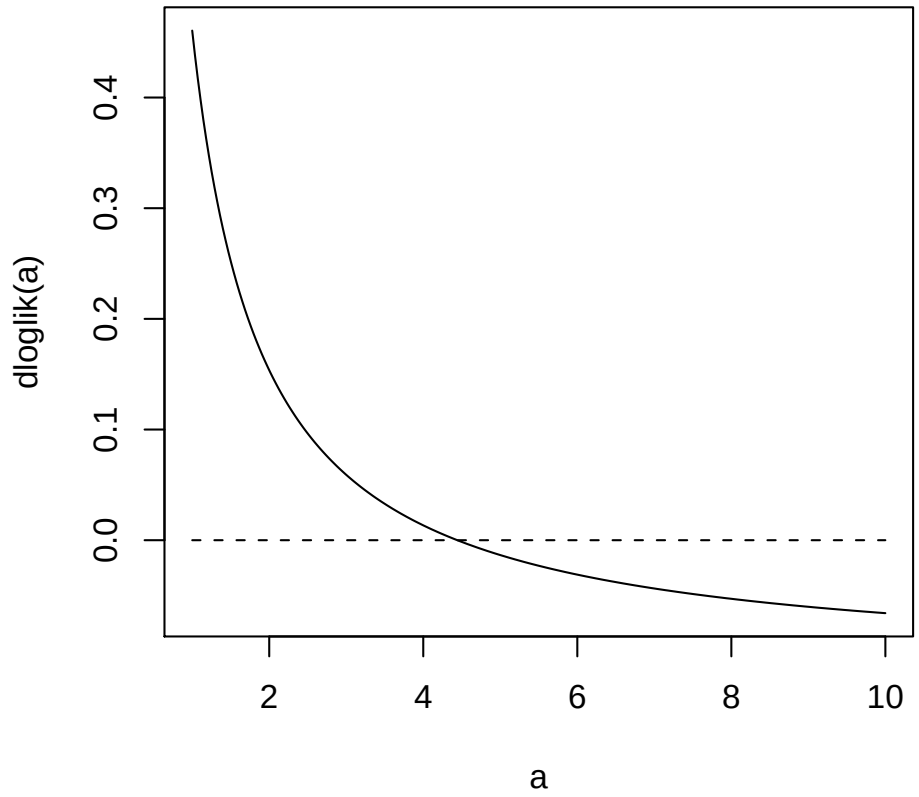


Figure 1.1: Plot of the derivative of the log likelihood of the Gamma function for the data associated with Example 1.

```
> x = rgamma(20, shape=5, scale = 2) # Generate some gammas with
> dloglikgamma(x, 0.1)   # looking for starting values for a and
> dloglikgamma(x, 100)
> a= 0.1
> b=100
> tol = 1e-10
>  i = 0  # iteration index
>  cur = (a + b) / 2
>  res = c(i, cur, dloglikgamma(x, cur))

>  while (abs(dloglikgamma(x, cur)) > tol) {
    i <- i + 1
    if (dloglikgamma(x, a) * dloglikgamma(x, cur) > 0)
      a <- cur
    else
      b <- cur
```

```
    cur <- (a + b) / 2
    res <- rbind(res, c(i, cur, dloglikgamma(x, cur)))
  }
> alphahat = res[nrow(res), 2]
> betahat = mean(x) / alphahat
> res
```

1.1.2 *Newton's method*

The bisection method is nice and safe, and will always find a root, but with additional knowledge of the function, we can speed up the convergence. One popular way of doing this is with *Newton's method*, which is based on a first-order Taylor series approximation to the function f.

- Given a "guess" θ_0, we can approximate f as:

$$f(\theta) \approx f(\theta_0) + f'(\theta_0)(\theta - \theta_0).$$

- We could solve for the zero of the linear approximation:

$$f(\theta_0) + f'(\theta_0)(\theta - \theta_0) = 0$$

  and we find

$$\theta = \theta_0 - \frac{f(\theta_0)}{f'(\theta_0)}.$$

- The Newton algorithm involves doing this iteratively:

$$\theta_i = \theta_{i-1} - \frac{f(\theta_{i-1})}{f'(\theta_{i-1})}$$

  until $|f(\theta_i)|$ is sufficiently close to zero.

Applying the Newtown's method to the same Gamma example, we need the derivative of (1) with respect to α:

$$\frac{1}{\alpha} - \frac{d^2}{d\alpha^2} \log \Gamma(\alpha).$$

Fortunately, `R` has just the function we need:

```
d2loglikgamma <- function(x, a) {
  return(1/a - trigamma(a))
}
```

```
> i = 0
```

```
>  cur <- start <- 1
> resnewton <- c(i, cur, dloglikgamma(x, cur))
  while (abs(dloglikgamma(x, cur)) > tol) {
    i <- i + 1
    cur <- cur - dloglikgamma(x, cur) / d2loglikgamma(x, cur)
    resnewton <- rbind(resnewton, c(i, cur, dloglikgamma(x, cur))
  }
> alphahat <- resnewton[nrow(resnewton), 2]
> betahat <- mean(x)/alphahat
> resnewton
```

Figure 1.2 shows the first four steps of this method in action.

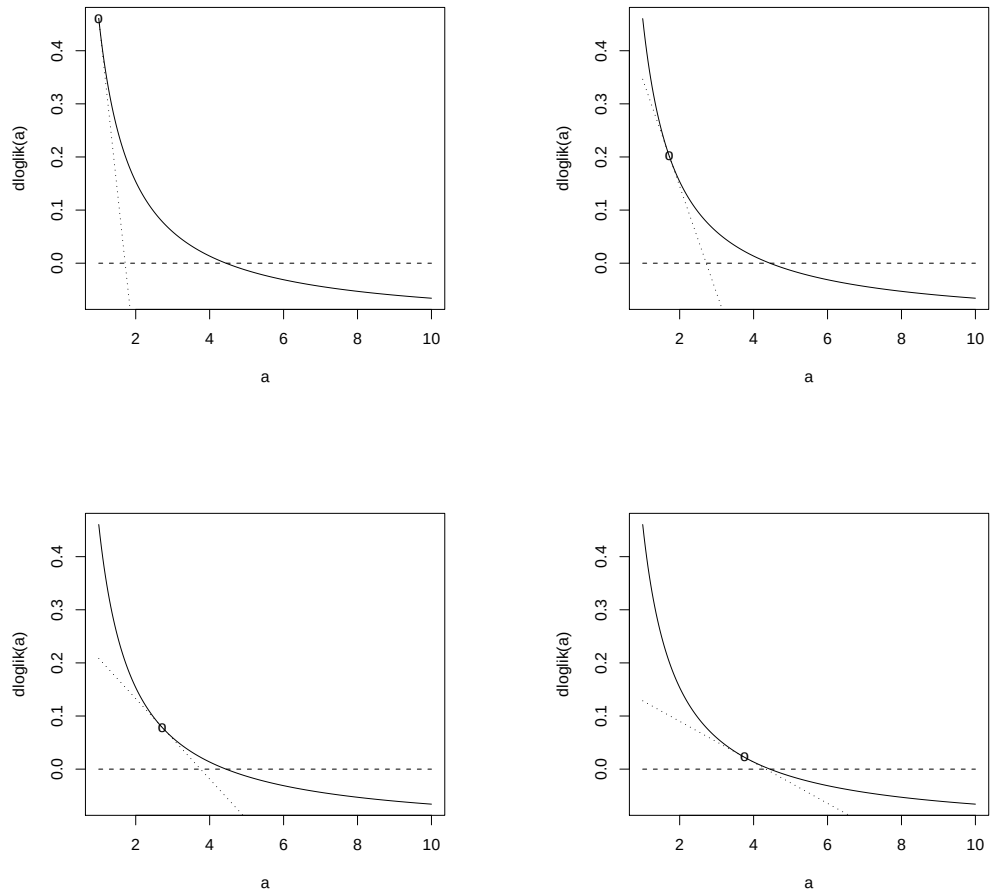


Figure 1.2: The Newton algorithm in action — first four steps for the data from Example 1

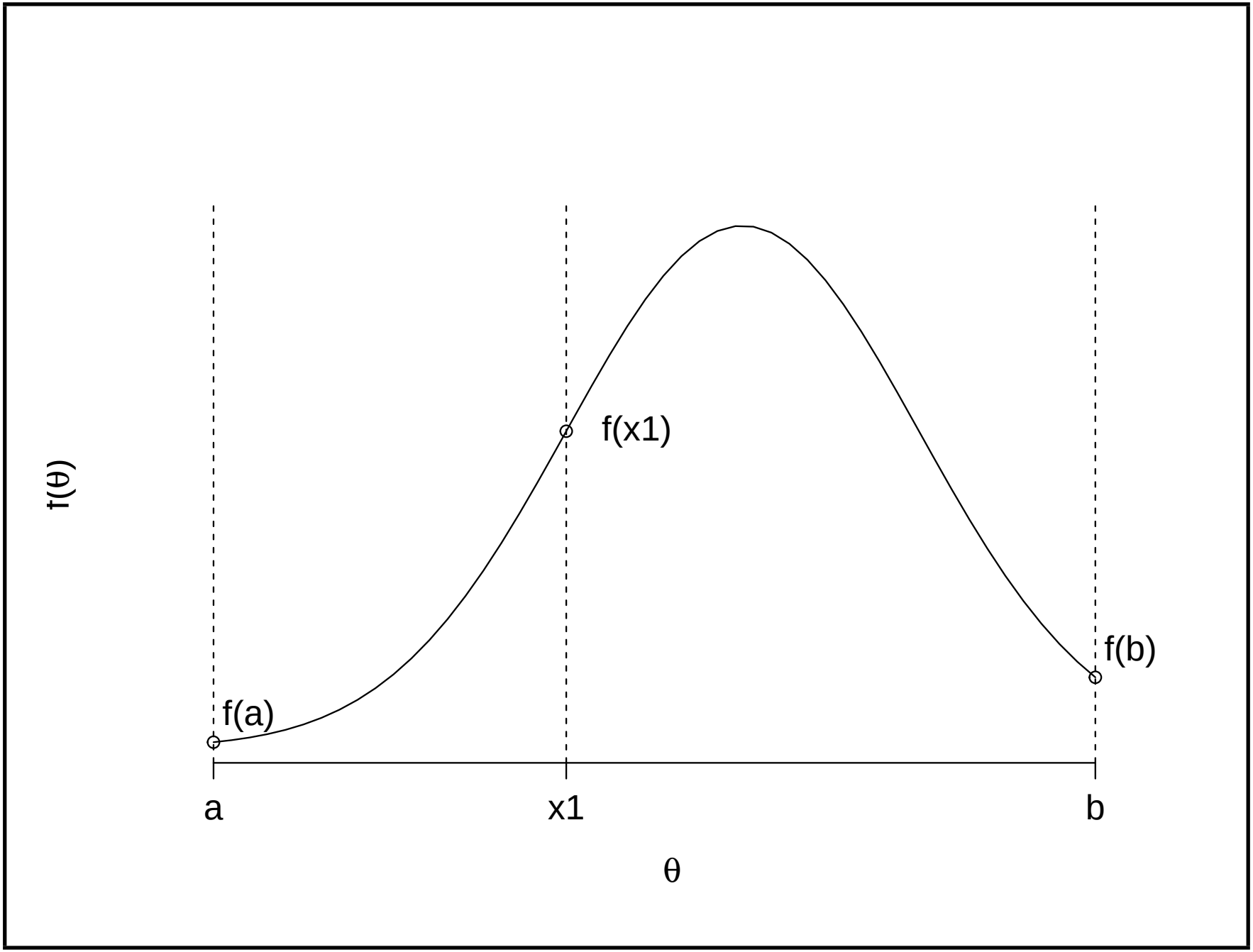
Clearly, this algorithm converges much faster than the bisection algorithm. However, it is not as ?safe? as the bisection algorithm. it does not always produce a solution. One may need to try different starting values to obtain a solution.

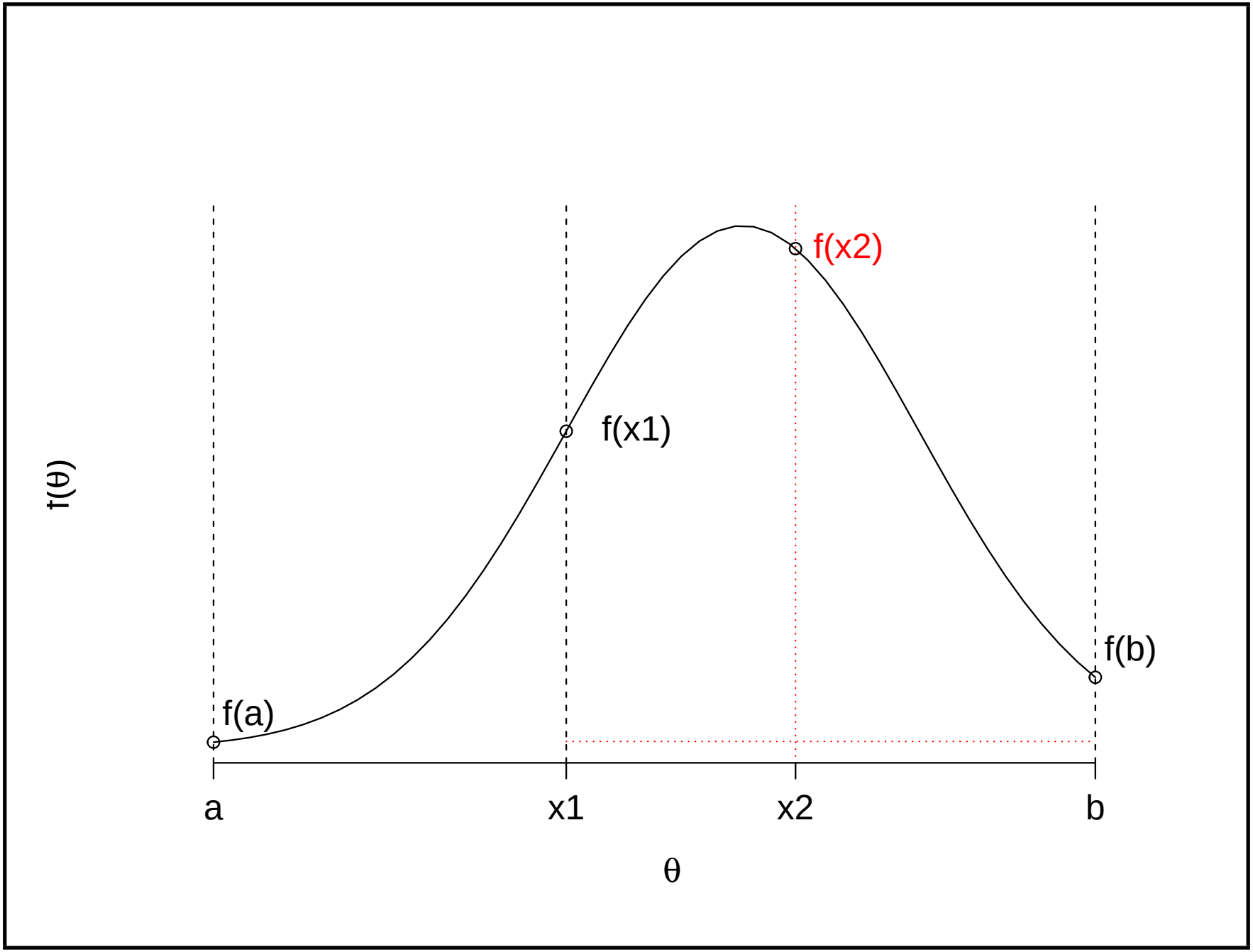
1.2 Golden section search to maximize a function

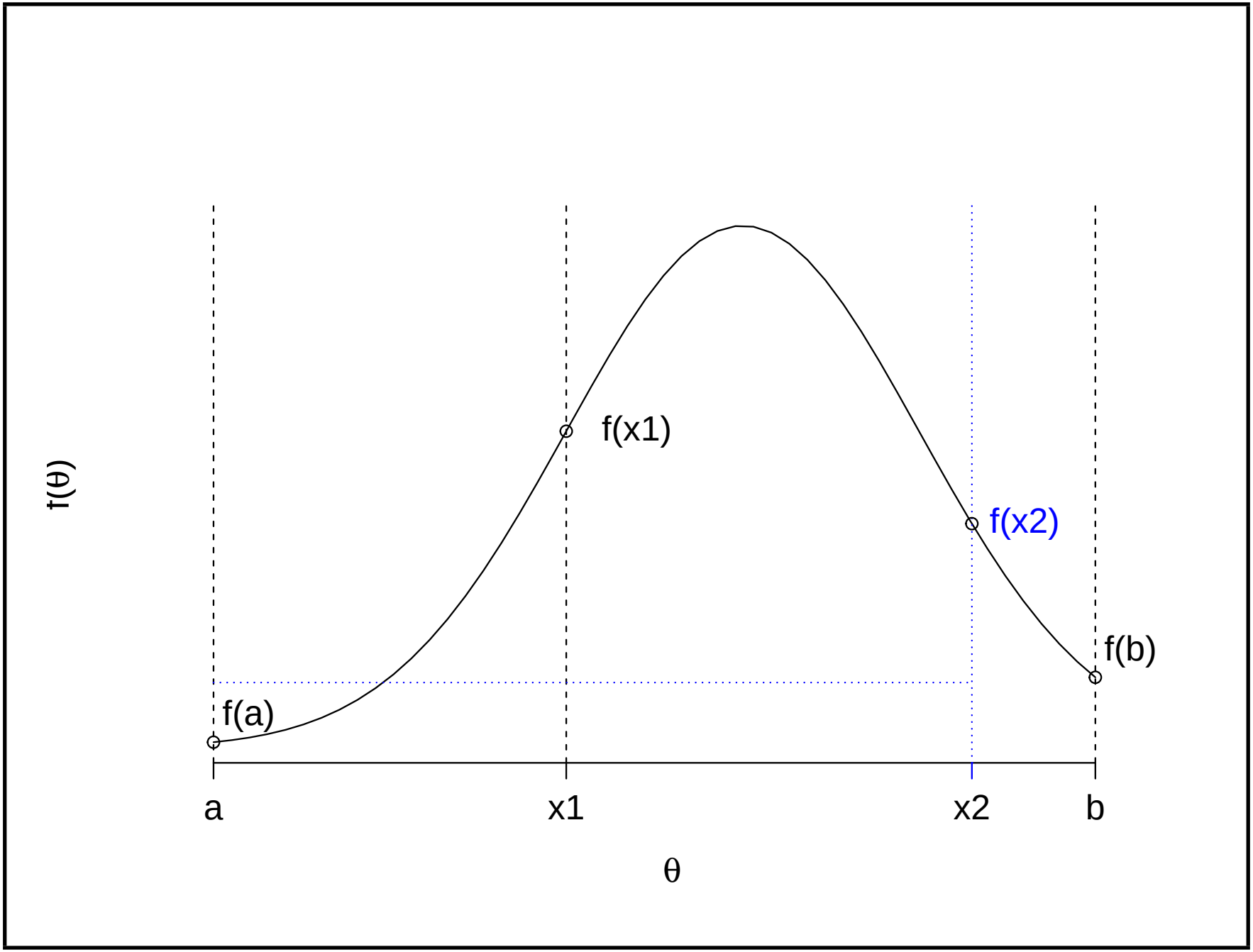
In many applications, we are not able to compute even one derivative analytically. In this case, we cannot find minimization and maximization by finding a zero of its derivative.

There are several algorithms to maximize the objective function (such like likelihood function) directly. One useful one is the *golden section search*.

Objective: find θ that maximize $f(\theta)$, assuming $f(\theta)$ is uni-modal .


f(Θ)
f(x1)
f(b)
f(a)
a
x1
b
θ


f(x2)
f(x1)
f(Θ)
f(b)
f(a)
a
x1
x2
b
θ


f(Θ)
f(x1)
f(x2)
f(b)
f(a)
a
x1
x2
b
θ

How to place a new splitting point ?

Let's assume we always split the larger interval into half.

- Best case: remove half of the interval
- Worst case: remove $1/4$ of the interval

If the worst case occurs at all the iterations, the search is inefficient.

Let w be the fraction of larger interval. Then in the initial step,

$$x_1 - a = (1 - w)(b - a) \quad \text{and} \quad b - x_1 = w(b - a).$$

What is the length of next interval?

It equals to $(1 - w)(b - a) + (x_2 - x_1)$ if $f(x_2) < f(x_1)$;

It equals to $w(b - a)$ if $f(x_2) > f(x_1)$;

The optimal way to minimize the worst case is to making them equal. i.e.

$$(1-w)(b-a)+(x_2-x_1)=(b-a)w. \tag{2}$$

That is true at all the iterations. Hence w should keep the same in all the iterations. i.e.

$$(x_2-x_1)=w(1-w)(b-a). \tag{3}$$

Combine the two equations (2) and (3), we have

$$w^2+w-1=0$$

the solution to the equation above is $w=0.618$, which is known as "golden ratio".

Example 1.2 Trinomial data. The probability mass function for the trinomial distribution with probabilities p_1, p_2, and p_3 is

$$P(X_1 = x_1, X_2 = x_2, X_3 = x_3) = \frac{(x_1 + x_2 + x_3)! p_1^{x_1} p_2^{x_2} p_3^{x_3}}{x_1! x_2! x_3!}.$$

If the probabilities are determined by a single parameter as

$$p_1 = \frac{\theta + 2}{4}; \quad p_2 = \frac{1 - \theta}{2}; \quad p_3 = \frac{\theta}{4},$$

then for data X_1, X_2, and X_3, maximizing the likelihood is equivalent to maximizing

$$x_1 \log(\theta + 2) + x_2 \log(1 - \theta) + x_3 \log \theta.$$

We generate some data:

```
>  n = 100
> theta <- 0.2
> Xdata <- rmultinom(n, 40, c((2 + theta)/4, (1 - theta)/2,
   theta/4))
```

Write R program to find MLE of θ.

1.3 Optimization of a multivariate function: Some preliminaries

We are interested in finding the maximum of a function $f : \mathbb{R}^p \to \mathbb{R}$, i.e., to find $\boldsymbol{\theta}^*$ such that

$$f(\boldsymbol{\theta}) \leq f(\boldsymbol{\theta}^*)$$

for all $\boldsymbol{\theta}$ in a neighborhood of $\boldsymbol{\theta}^*$. Note that this is the definition of a *local maximum*. Again, we assume unique solution.

The univariate algorithms based on intervals (such as the bisection algorithm and the golden section search algorithm) are not readily adapted to higher dimensions. Newton's method, on the other hand, generalizes nicely.

Definition 1.1 *gradient:* For a function $f : \mathbb{R}^p \to \mathbb{R}$ and a point $\boldsymbol{\theta} \in \mathbb{R}^p$, the gradient is the p-vector

$$\nabla f(\boldsymbol{\theta}) = \left(\frac{\partial f(\boldsymbol{\theta})}{\partial \theta_1}, \ldots, \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_p} \right)' .$$

Exercise 1.1 *Compute the gradient of the function $f : \mathbb{R}^3 \to \mathbb{R}$ defined by $f(\boldsymbol{\theta}) = \theta_1^2 \theta_2^3 + \theta_3 \cos \theta_1$ at the point $\boldsymbol{\theta} = (\pi/2, 2, 3)$.*

Definition 1.2 *Hessian:* For a function $f : \mathbb{R}^p \to \mathbb{R}$ and a point $\boldsymbol{\theta} \in \mathbb{R}^p$, the Hessian is the $p \times p$ matrix of second partial derivatives with i, j element equal to

$$\frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_i \partial \theta_j}.$$

It is denoted $\nabla^2 f(\boldsymbol{\theta})$.

Exercise 1.2 *Compute the Hessian matrix for the function* $f : \mathbb{R}^2 \to \mathbb{R}$ *defined by* $f(\boldsymbol{\theta}) = \theta_1^5 + \theta_2^4 - 5\theta_1 - 32\theta_2$ *at the following points:*

- $(1, 2)$
- $(-1, 2)$

Definition 1.3 *positive definite:* A $p \times p$ matrix W is said to be positive definite if $\boldsymbol{\theta}' W \boldsymbol{\theta} > 0$ for all non-zero p-vectors $\boldsymbol{\theta}$. Similarly, a matrix is negative definite if $\boldsymbol{\theta}' W \boldsymbol{\theta} < 0$ for all $\boldsymbol{\theta}$. The definition for non-negative definite comes from replacing the strict inequality $>$ in the definition of positive definite with $\geq$. (Similarly for non-positive definite.)

Theorem 1.1 *A symmetric matrix W is positive (negative) definite iff all of its eigenvalues are positive (negative). It is non-negative (non-positive) definite iff all of its eigenvalues are non-negative (non-positive).*

For a smooth function f, if $\boldsymbol{\theta}^*$ is a local maximum, then $\nabla f(\boldsymbol{\theta}^*) = \mathbf{0}$ and $\nabla^2 f(\boldsymbol{\theta}^*)$ is nonpositive definite. Thus, searching for local maxima can be accomplished (in part) by searching for points $\boldsymbol{\theta}^*$ for which $\nabla f(\boldsymbol{\theta}^*) = \mathbf{0}$. (This alone does not guarantee that $\boldsymbol{\theta}^*$ is a local maximum; it could be a local minimum or a "saddlepoint".)

Most maximization algorithms follow these general steps (repeating until convergence):

1. Given a current point $\boldsymbol{\theta}_0$, choose a direction $\mathbf{d}$ in which to move next.
2. Find a point $\boldsymbol{\theta}_1 = \boldsymbol{\theta}_0 + \lambda \mathbf{d}$ such that $f(\boldsymbol{\theta}_1) > f(\boldsymbol{\theta}_0)$.
3. Set $\boldsymbol{\theta}_0$ to be equal to $\boldsymbol{\theta}_1$ and go to Step 1.

In order for this to work, the direction $\mathbf{d}$ must be chosen so that the function f actually increases in that direction (from $\boldsymbol{\theta}_0$).

1.4 The Newton-Raphson algorithm

The Newton-Raphson algorithm is a multivariate generalization of Newton's method for iteratively optimizing a function of one variable. It can be regarded as a method to search for solutions to the system of nonlinear equations $\nabla f(\boldsymbol{\theta}) = \mathbf{0}$.

At each step, given the current point $\boldsymbol{\theta}_0$, the gradient $\nabla f(\boldsymbol{\theta})$ for $\boldsymbol{\theta}$ near $\boldsymbol{\theta}_0$ may be approximated by

$$\nabla f(\boldsymbol{\theta}_0) + \nabla^2 f(\boldsymbol{\theta}_0)(\boldsymbol{\theta} - \boldsymbol{\theta}_0),$$

which defines the plane tangent to $\nabla f(\boldsymbol{\theta})$ at $\boldsymbol{\theta}_0$. The next step in the algorithm is determined by solving the system of linear equations

$$\nabla f(\boldsymbol{\theta}_0) + \nabla^2 f(\boldsymbol{\theta}_0)(\boldsymbol{\theta} - \boldsymbol{\theta}_0) = \mathbf{0} \tag{4}$$

and the next "current point" is set to be the solution. Solving (4) gives

$$\boldsymbol{\theta}_1 = \boldsymbol{\theta}_0 - \left[\nabla^2 f(\boldsymbol{\theta}_0)\right]^{-1} \nabla f(\boldsymbol{\theta}_0).$$

The algorithm continues like this iteratively choosing a sequence of points $\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_i, \ldots$ until convergence is achieved. The ith step is given by

$$\boldsymbol{\theta}_i = \boldsymbol{\theta}_{i-1} - \left[\nabla^2 f(\boldsymbol{\theta}_{i-1})\right]^{-1} \nabla f(\boldsymbol{\theta}_{i-1}).$$

Example 1.3 Simple logistic regression. with a binary response variable and a single numerical explanatory variable.

$$P(Y_i = 1|x_i) = \frac{e^{\beta_0+\beta_1 x_i}}{1 + e^{\beta_0+\beta_1 x_i}}.$$

For data $(x_1, Y_1), \ldots, (x_n, Y_n)$, the likelihood is given by

$$\prod_{i=1}^{n} \left[\left(\frac{e^{\beta_0+\beta_1 x_i}}{1 + e^{\beta_0+\beta_1 x_i}} \right)^{Y_i} \left(\frac{1}{1 + e^{\beta_0+\beta_1 x_i}} \right)^{1-Y_i} \right].$$

Objective: Find MLE for β's.

Maximizing the likelihood is equivalent to maximizing the log likelihood:

$$f(\beta_0, \beta_1) = \sum_{i=1}^{n} \left(Y_i(\beta_0 + \beta_1 x_i) - \log\left(1 + e^{\beta_0 + \beta_1 x_i}\right)\right).$$

The gradient of this function is

$$\nabla f(\beta_0, \beta_1) = \begin{pmatrix} \sum_{i=1}^{n} Y_i - p_i \\ \sum_{i=1}^{n} x_i(Y_i - p_i) \end{pmatrix}, \tag{6}$$

where $p_i = P(Y_i = 1|x_i)$ as given in (5).

The Hessian is given by

$$\begin{aligned}\nabla^2 f(\beta_0,\beta_1) &= -\sum_{i=1}^{n}\begin{pmatrix}1\\ x_i\end{pmatrix}\begin{pmatrix}1 & x_i\end{pmatrix}p_i(1-p_i)\\ &= -\begin{pmatrix}\sum p_i(1-p_i) & \sum x_i p_i(1-p_i)\\ \sum x_i p_i(1-p_i) & \sum x_i^2 p_i(1-p_i)\end{pmatrix}. \quad (7)\end{aligned}$$

The Hessian can be seen to be negative definite at all parameter values since

$$(a_1,a_2)\nabla^2 f(a_1,a_2)(a_1,a_2)' = -\sum_{i=1}^{n}(a_1+a_2x_i)^2 p_i(1-p_i) < 0$$

for all values of a_1 and a_2. (The only way this could be zero is if all the x_i's are the same.)

A well behaved problem.

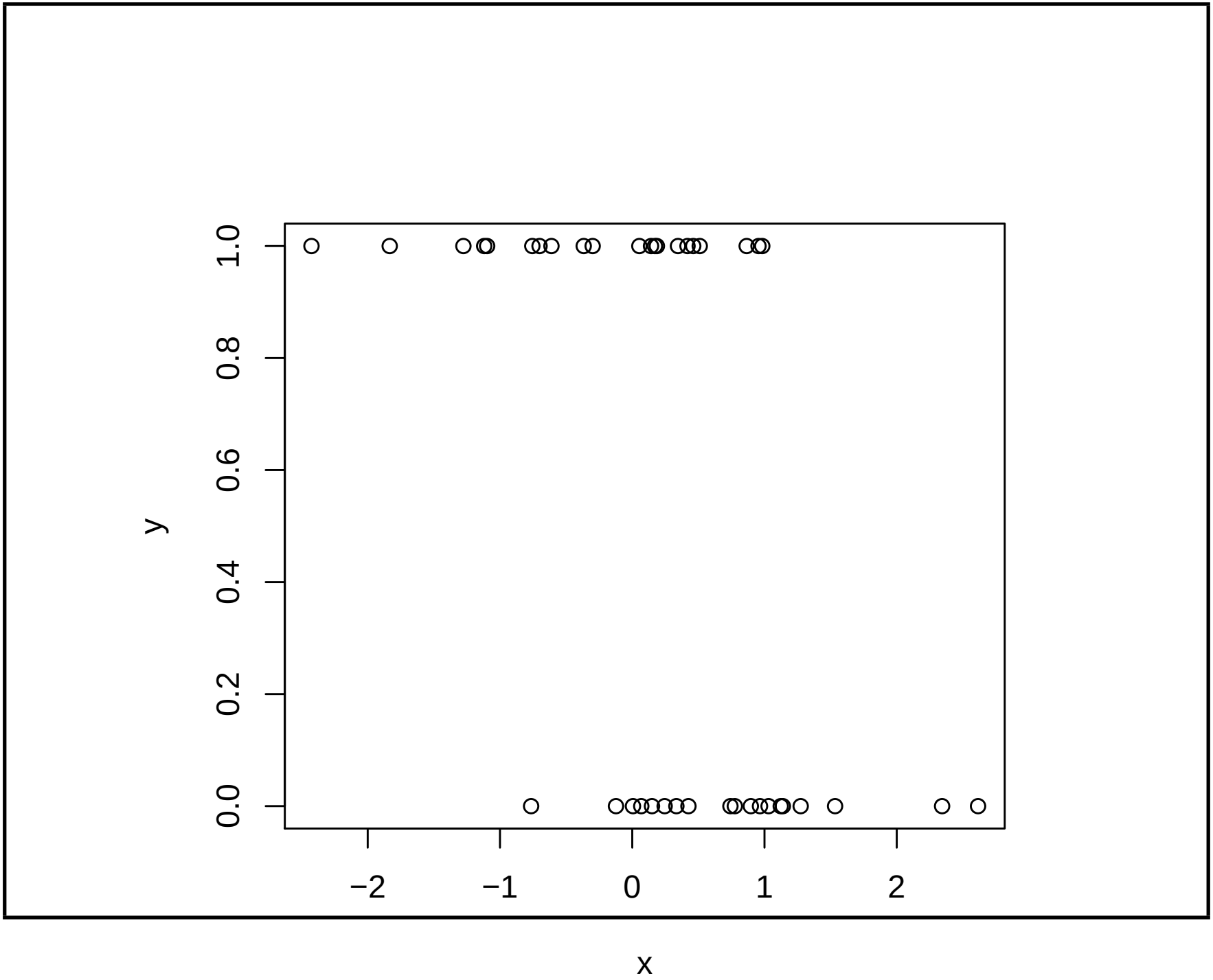
```
# Function to compute the loglikelihood, the gradient, and
# the Hessian matrix for data dat evaluated at the parameter
# value betavec.
#
# dat     - A list with components
#             x - vector of explanatory variables
#             y - vector of corresponding (binary) response
#                  variables
# betavec - [beta_0, beta_1] - the vector of parameter
#                values at which to evaluate these quantities
#
# Returns a list with the following components evaluated at beta
#   loglik - (scalar) the log likelihood
#   grad   - (vector of length 2) gradient
#   Hess   - (2 x 2 matrix) Hessian
#
```

```
logisticstuff <- function(dat, betavec) {
  u <- betavec[1] + betavec[2] * dat$x
  expu <- exp(u)
  loglik <- sum(dat$y * u - log(1 + expu))
# Log-likelihood at betavec
  p <- expu / (1 + expu)
# P(Y_i=1|x_i)
  grad <- c(sum(dat$y - p), sum(dat$x * (dat$y - p)))
# gradient at betavec
  Hess <- -matrix(c(sum(p * (1 - p)),
                    rep(sum(dat$x * p * (1-p)),2),
                   sum(dat$x^2 * p * (1 - p))), ncol=2)
# Hessian at betavec
  return(list(loglik = loglik, grad = grad, Hess = Hess))
}
```

Generate some data:

```
n <- 40
truebeta <- c(1, -2)
x <- rnorm(n)
expu <- exp(truebeta[1] + truebeta[2] * x)
y <- runif(n) < expu / (1 + expu)
```

The data are displayed in Figure 1.3.


y
0.0
0.2
0.4
0.6
0.8
1.0
−2
−1
0
1
2
x

Starting with the log-likelihood, gradient, and Hessian matrix for these data at the "true" values of β_0 and β_1:

```
> test1 <- logisticstuff(list(x = x, y = y), c(1, -2))
> test1
```

The first step of the Newton-Raphson algorithm, with starting values $(1, -2)$ is thus

```
> theta1 <- c(1, -2) - solve(test1$Hess) %*% test1$grad
> theta1
```

We would then compute the loglikelihood, gradient, and Hessian at this new value:

```
> test2 <- logisticstuff(list(x = x, y = y), theta1)
> test2
```

Does the likelihood increase at the new values?

The next step would thus be

```
> theta2 <- theta1 - solve(test2$Hess) %*% test2$grad
> theta2
```

A simple function to do Newton-Raphson:

```
# Function to maximize a likelihood using the Newton-Raphson
# algorithm.
#
# dat     - data (see specifications for func)
# func    - function to compute log likelihood, gradient,
#             and Hessian for data and a given choice of the
#             parameter vector. must be called as func(dat, par)
#             and return a list with
#             elements
#                loglik - log likelihood
#                grad   - gradient
#                Hess   - Hessian
```

```
# start   - vector of starting values -- must have same
#              dimension as second argument to func
# tol     - tolerance: the program stops when the difference
#             in consecutive log likelihood values is less than
#              tol in absolute value
# maxiter - maximum number of iterations before the program
#               stops execution
# Returns a matrix with 1st column = iteration number, 2nd column
#   likelihood, 3rd column = par[1] value, 4th column = par[2] va
#
NewtonRaphson <- function(dat, func, start, tol=1e-10,
                                  maxiter = 200) {
  i <- 0
  cur <- start
  stuff <- func(dat, cur)
  res <- c(0, stuff$loglik, cur)
```

```
  prevloglik <- -Inf      # To make sure it iterates
  while(i < maxiter && abs(stuff$loglik - prevloglik) > tol)
 {
    i <- i + 1
    prevloglik <- stuff$loglik
    prev <- cur
    cur <- prev - solve(stuff$Hess) %*% stuff$grad
    stuff <- func(dat, cur)         # log-lik, gradient, Hessian
    res <- rbind(res, c(i, stuff$loglik, cur))
    # Add current values to results matrix
  }
  return(res)
}
```

Running the algorithm with good starting values:

```
> ans <- NewtonRaphson(list(x=x,y=y),logisticstuff,c(1,-2))
```

```
> ans
```

If we're not so good at choosing starting values, things may not turn out nicely:

```
> ans <- NewtonRaphson(list(x=x,y=y),logisticstuff,c(3,-1))
> ans
```

1.4.1 Modifications to Newton-Raphson algorithm

Definition 1.4 *ascent direction:* For a function f, a direction $\mathbf{d}$ is an ascent direction for f at a given point $\boldsymbol{\theta}_0$ if there exists some $\epsilon > 0$ such that

$$f(\boldsymbol{\theta}_0 + \lambda \mathbf{d}) > f(\boldsymbol{\theta}_0)$$

for all $0 < \lambda < \epsilon$.

Definition 1.5 *directional derivative:* The derivative of a function $f : \mathbb{R}^p \to \mathbb{R}$ at $\boldsymbol{\theta}$ in the direction of $\mathbf{d}$ is defined by

$$\lim_{\lambda \to 0} \frac{f(\boldsymbol{\theta} + \lambda \mathbf{d}) - f(\boldsymbol{\theta})}{\lambda} = \frac{\partial}{\partial \lambda} f(\boldsymbol{\theta} + \lambda \mathbf{d}) \Big|_{\lambda = 0} = \mathbf{d}' \nabla f(\boldsymbol{\theta}).$$

From this definition, we can see that $\mathbf{d}$ is an ascent direction for f at $\boldsymbol{\theta}_0$ if and only if $\mathbf{d}' \nabla f(\boldsymbol{\theta}_0) > 0$.

Exercise 1.3 *Given a function* $f : \mathbb{R}^2 \to \mathbb{R}$ *defined by* $f(\boldsymbol{\theta}) = -\left(3\theta_1^2 + \theta_2^2 + 2\theta_1\theta_2 - 2\theta_1 + \theta_2 + 6\right)$*, describe the set of all ascent directions from the point* $(-1, 0)'$.

General modification approaches: at each step, choosing a direction in which to move (d) and then moving some distance along that direction(λ) so that the likelihood increases.

Modified algorithm 1 One modified Newton-Raphson algorithm searchs at each step in the direction $-\left[\nabla^2 f(\boldsymbol{\theta}_{i-1})\right]^{-1} \nabla f(\boldsymbol{\theta}_{i-1})$ to find a better point. The movement at each iteration takes a step of the form

$$\boldsymbol{\theta}_i(\lambda) = \boldsymbol{\theta}_{i-1} - \lambda \left[\nabla^2 f(\boldsymbol{\theta}_{i-1})\right]^{-1} \nabla f(\boldsymbol{\theta}_{i-1}), \tag{8}$$

where the multiplier λ must be determined.

The unmodified Newton-Raphson method will tend to converge rather quickly when it gets close to a local maximum. So in the latter stages of the algorithm, it would be less efficient to search for a good value of

λ at each step than just to make the unmodified Newton-Raphson step. Thus, the ith step of the modified Newton-Raphson method is given by:

1. Compute the full Newton-Raphson step, corresponding to $\lambda = 1$ in (8).
2. If $f(\boldsymbol{\theta}_i(1)) > f(\boldsymbol{\theta}_{i-1})$, then set $\boldsymbol{\theta}_i = \boldsymbol{\theta}_i(1)$ and go on to the next iteration.
3. Otherwise, search for a value $\lambda \in (0, 1)$ for which $f(\boldsymbol{\theta}_i(\lambda)) > f(\boldsymbol{\theta}_{i-1})$, set $\boldsymbol{\theta}_i = \boldsymbol{\theta}_i(\lambda)$ and go on to the next iteration.

The last step in this modified algorithm is known as the "backtracking step."

One strategy for backtracking is to cut the potential step in half for each sub-iteration — i.e., check $\boldsymbol{\theta}_i(1/2)$ first and if that doesn't increase f, check $\boldsymbol{\theta}_i(1/4)$ next, then $\boldsymbol{\theta}_i(1/8)$, etc. The resulting algorithm is known as the *modified Newton-Raphson algorithm with step halving*.

In the earlier example, when the starting value is $(-3, 1)$, check to see if step halving help improve the algorithm?

Modified algorithm 2 Newton-Raphson algorithm does not guarantee that the direction given by $-\left[\nabla^2 f(\boldsymbol{\theta}_{i-1})\right]^{-1}\nabla f(\boldsymbol{\theta}_{i-1})$ will be an ascent direction at $\boldsymbol{\theta}_{i-1}$. If it's not, then backtracking won't work.

Recall that a vector $\mathbf{d}$ is an ascent direction at $\boldsymbol{\theta}_{i-1}$ if and if $\mathbf{d}'\nabla f(\boldsymbol{\theta}_{i-1}) > 0$.

For the Newton-Raphson algorithm, the direction is

$$\mathbf{d} = -\left[\nabla^2 f(\boldsymbol{\theta}_{i-1})\right]^{-1}\nabla f(\boldsymbol{\theta}_{i-1}),$$

so

$$\begin{aligned}\mathbf{d}'\nabla f(\boldsymbol{\theta}_{i-1}) &= \left(-\left[\nabla^2 f(\boldsymbol{\theta}_{i-1})\right]^{-1}\nabla f(\boldsymbol{\theta}_{i-1})\right)'\nabla f(\boldsymbol{\theta}_{i-1}) \\ &= -(\nabla f(\boldsymbol{\theta}_{i-1}))'\left[\nabla^2 f(\boldsymbol{\theta}_{i-1})\right]^{-1}\nabla f(\boldsymbol{\theta}_{i-1}). \quad (9)\end{aligned}$$

$\nabla^2 f(\boldsymbol{\theta}_{i-1})$ is negative definite if $\boldsymbol{\theta}_{i-1}$ is sufficiently close to a local maximum, or $f(\cdots)$ is concave.

In the case that $\nabla^2 f(\boldsymbol{\theta}_{i-1})$ is not negative definite, we could replace it with a negative definite matrix that is similar to $\nabla^2 f(\boldsymbol{\theta}_{i-1})$. Such as

$$\nabla^2 f(\boldsymbol{\theta}_{i-1}) - \gamma I,$$

with γ chosen to be large enough to make the result negative definite.

Modified algorithm 3 - Fisher's scoring

For a log-likelihood function $\ell(\boldsymbol{\theta}|\mathbf{X})$, the *score function* is defined to be the gradient of ℓ:

$$\mathbf{U}(\boldsymbol{\theta}) = \nabla\ell(\boldsymbol{\theta}|\mathbf{X}) = \begin{bmatrix} \frac{\partial\ell(\boldsymbol{\theta}|\mathbf{X})}{\partial\theta_1} \\ \vdots \\ \frac{\partial\ell(\boldsymbol{\theta}|\mathbf{X})}{\partial\theta_p} \end{bmatrix}.$$

The MLEs solve the *likelihood equations*:

$$\mathbf{U}(\boldsymbol{\theta}) = \mathbf{0}. \tag{10}$$

Applying the Newton-Raphson, we would iteratively update parameter estimates according to

$$\boldsymbol{\theta}_i = \boldsymbol{\theta}_{i-1} - \left[\nabla^2\ell(\boldsymbol{\theta}_{i-1}|\mathbf{X})\right]^{-1}\mathbf{U}(\boldsymbol{\theta}_{i-1}).$$

Fisher scoring replace the "observed information matrix" $\nabla^2 \ell(\boldsymbol{\theta}|\mathbf{X})$ by negative *Fisher information matrix*,

$$I(\boldsymbol{\theta}) = \mathrm{Var}(\mathbf{U}(\boldsymbol{\theta})) = \mathrm{E}\left[\mathbf{U}(\boldsymbol{\theta})\mathbf{U}(\boldsymbol{\theta})'\right]$$

If things are sufficiently regular,

$$I(\boldsymbol{\theta}) = -\mathrm{E}\left[\nabla^2 \ell(\boldsymbol{\theta}|\mathbf{X})\right].$$

$$\nabla^2 \ell(\boldsymbol{\theta}|\mathbf{X}) \Rightarrow -I(\boldsymbol{\theta})$$

Fisher information matrix is guaranteed to be positive definite near the MLE (and thus -1 times the Fisher information matrix will be negative definite). Consequently Fisher's scoring algorithm is more stable.