# STAT 7650: Computational Statistics

## 3. Optimization and Solving Nonlinear Equations

Peng Zeng

Department of Mathematics and Statistics
Auburn University

Spring 2026

# Optimization Problem

An optimization problem is to solve

$$\max_{x \in D} f(x),$$

where $f(x)$ is the objective function and the domain $D$ is called the feasible set.

- Minimization or maximization
- Constrained or unconstrained, continuous or discrete ($D$)
- Differentiable or non-differentiable ($f$)
- Convex problem (concave problem)
- Global optimum or local optimum

# Differentiable Objective Function

When the objective function is differentiable, the optimization problem is usually reformulated as solving nonlinear equations.

Let $y_1, \ldots, y_n \sim^{iid} f(y \mid \theta)$. The MLE of $\theta$ solves

$$\max_{\theta} \ell(\theta), \quad \text{where } \ell(\theta) = \sum_{i=1}^{n} \log f(y_i \mid \theta)$$

When $\ell(\theta)$ is differentiable, the MLE is the root of the score function,

$$U(\theta) = \frac{\partial \ell(\theta)}{\partial \theta} = \sum_{i=1}^{n} \frac{\partial \log f(y_i \mid \theta)}{\partial \theta} = 0$$

In many applications, there exists no explicit solution for the MLE.

# Outline

1. Univariate problems
   - Bisection method
   - Newton's method
   - Secant method
   - Fixed-point iteration
2. Multivariate problems
   - Newton's method
   - Fisher scoring
   - Ascent algorithm
   - Quasi-Newton methods
   - Gauss-Newton method
   - Nonlinear Gauss-Seidel iteration
   - Nelder-Mead algorithm

References: Givens and Hoeting (2013). Chapter 2.

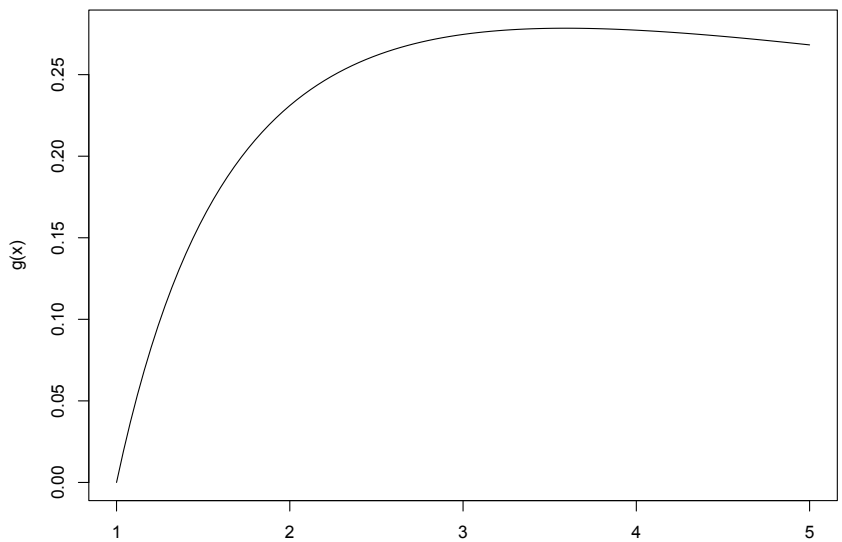
# Example

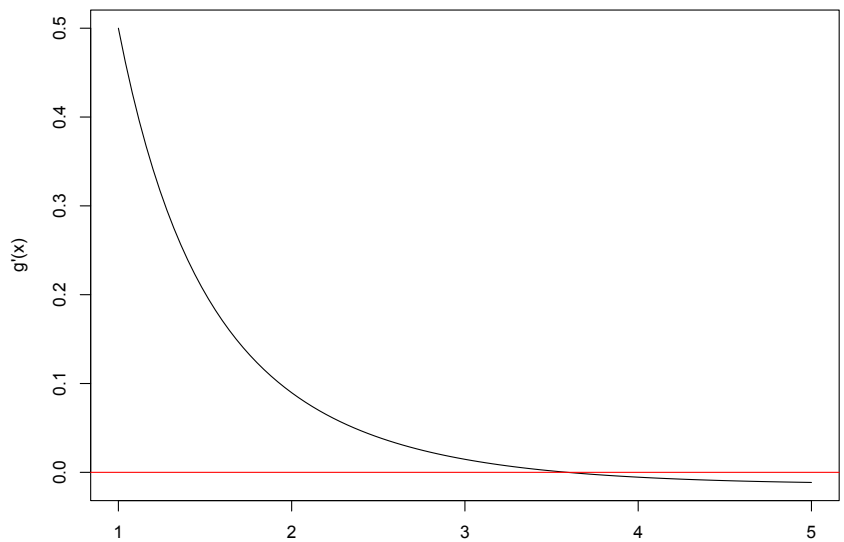
Consider a univariate optimization problem.

$$\max_x g(x), \quad \text{where } g(x) = \frac{\log(x)}{1+x}$$

Equivalently, find the root of

$$g'(x) = \frac{1 + 1/x - \log(x)}{(1+x)^2} = 0$$



# Iterative Procedure

General steps.

- Find an initial value (starting value) $x^{(0)}$
- Compute an improved value $x^{(k+1)}$ from $x^{(k)}$
- Check stopping rule. For a small value $\varepsilon$, the absolute or relative convergence criterion is

$$|x^{(t+1)} - x^{(t)}| < \varepsilon, \quad \frac{|x^{(t+1)} - x^{(t)}|}{|x^{(t)}|} < \varepsilon, \quad \frac{|x^{(t+1)} - x^{(t)}|}{|x^{(t)}| + \varepsilon} < \varepsilon$$

  Stop if the convergence criterion is met or the algorithm has run $N$ iterations. Continue otherwise.

# Code Template

```
tol = 1e-5          # tolerance
maxiter = 100       # max number of iterations

x = 0               # initial value
iter = 0            # number of iterations
conv = 1            # check convergence

while((conv > tol) && (iter < maxiter))
{
    iter = iter + 1
    x_cur = x                   # save the current x
    x = update_value(x)         # update x
    conv = abs(x - x_cur)
}
```

The absolute convergence criterion in the code can be changed to the relative convergence criterion.

# Bisection Method

Use bisection method to find the root of $m(x) = 0$.

If $m(x)$ is continuous on $[a_0, b_0]$ and $m(a_0)m(b_0) \leq 0$, then the Intermediate Value Theorem implies that there exists at least one $x^* \in [a_0, b_0]$ for each $m(x^*) = 0$.

Let $x^{(0)} = (a_0 + b_0)/2$ be the starting value. Then

$$[a_{t+1}, b_{t+1}] = \begin{cases} [a_t, x^{(t)}], & \text{if } m(a_t)m(x^{(t)}) \leq 0 \\ [x^{(t)}, b_t], & \text{if } m(a_t)m(x^{(t)}) > 0 \end{cases}$$

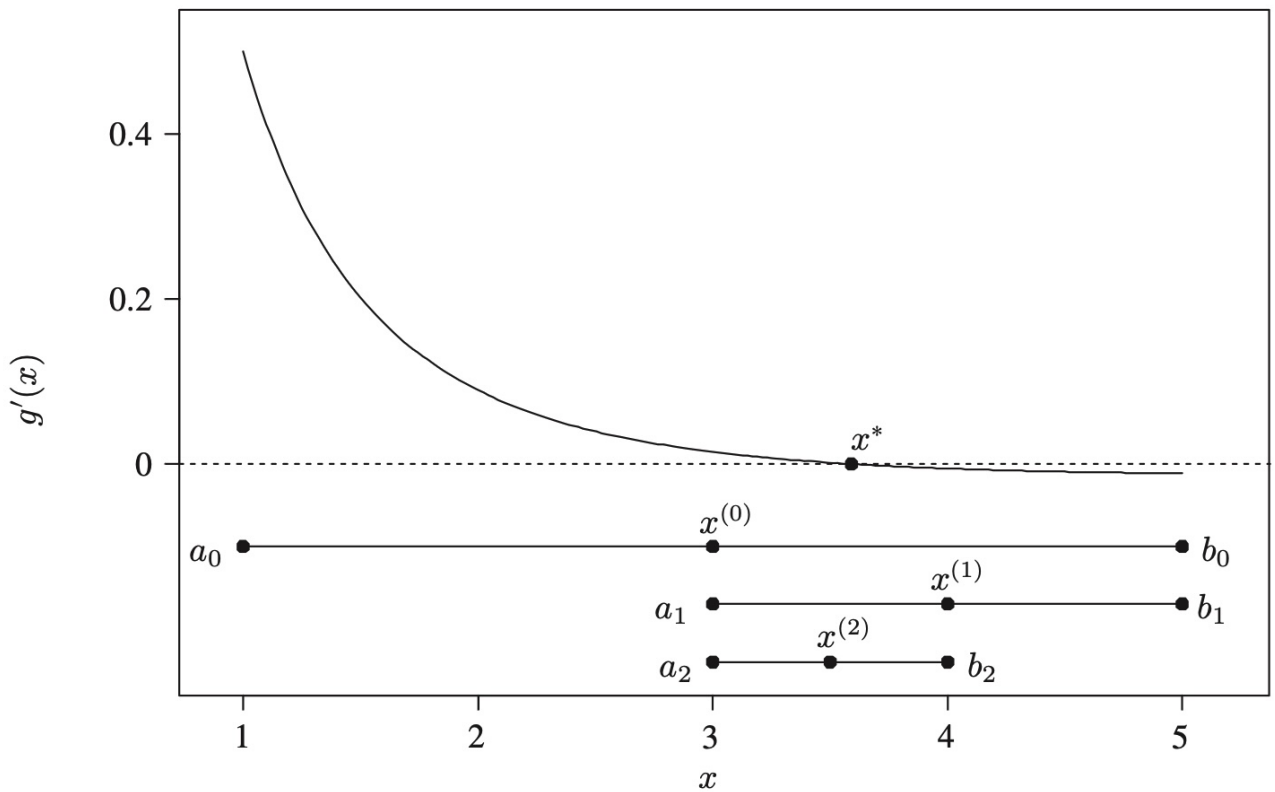
and

$$x^{(t+1)} = \frac{(a_{t+1} + b_{t+1})}{2}$$

If $m(x)$ has more than one root in the starting interval, bisection will find one of them, but will not find the rest.

# Comments

- The bisection method is guaranteed to converge to a root in $[a_0, b_0]$ if $m(x)$ is continuous.
- one root or multiple roots, local optimum or global optimum
- Bisection method is robust, but slow.



# Newton's Method

Use Newton's method to find the root of $m(x) = 0$.

Suppose that $x^*$ is the true solution and use Tylor expansion

$$0 = m(x^*) \approx m(x^{(t)}) + m'(x^{(t)})(x^* - x^{(t)}).$$

Then we have the updating equation

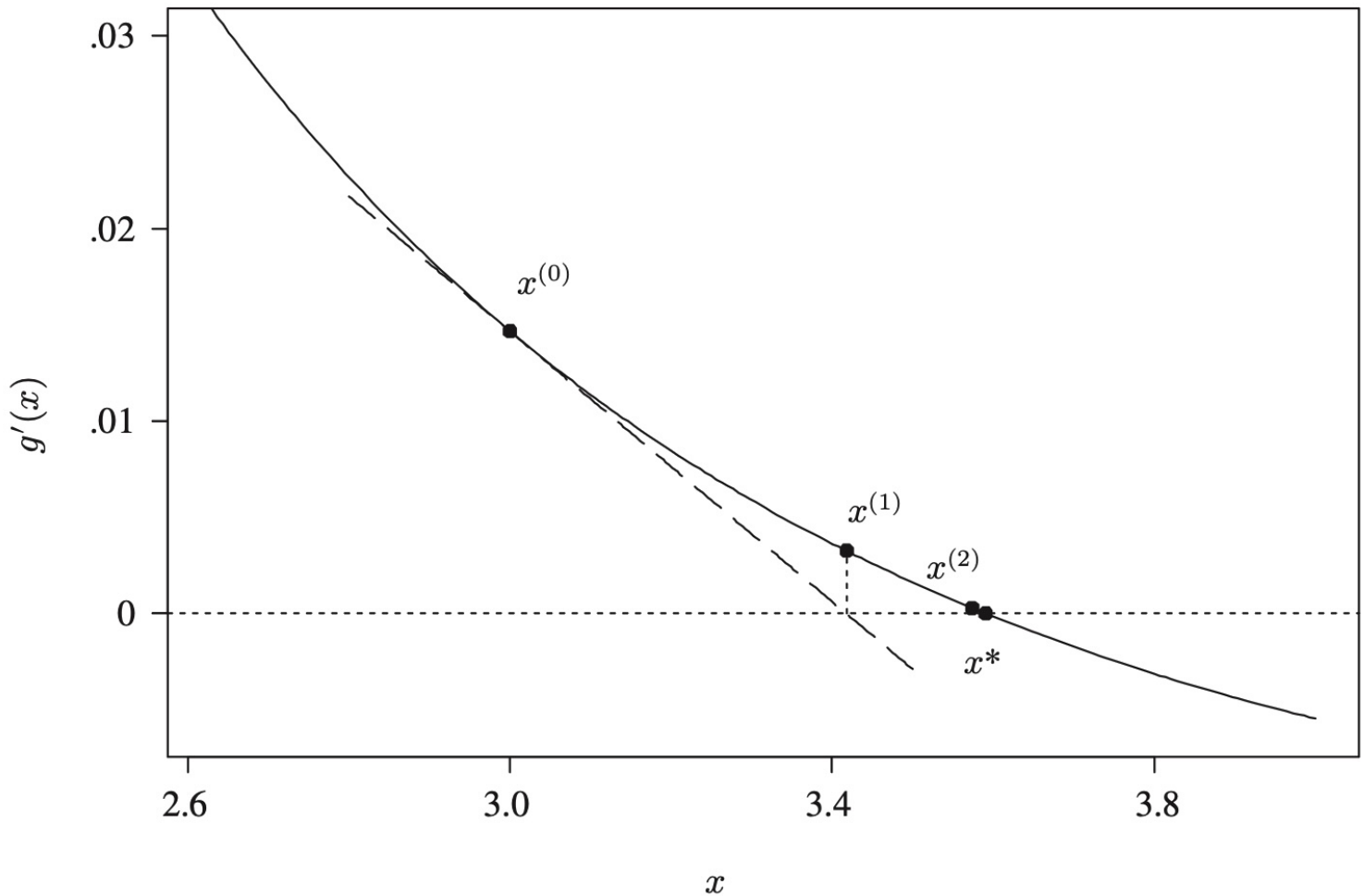
$$x^{(t+1)} = x^{(t)} - \frac{m(x^{(t)})}{m'(x^{(t)})}$$

When applying it to optimize $\max f(x)$, the updating equation is

$$x^{(t+1)} = x^{(t)} - \frac{f'(x^{(t)})}{f''(x^{(t)})}$$

# Interpretation of Newton's Method

Draw the tangent line at $x^{(t)}$. This tangent line intersects the horizontal axis at $x^{(t+1)}$.



# Code Template

```
tol = 1e-5                  # tolerance
maxiter = 100               # max number of iterations

x = 0                       # initial value
iter = 0                    # number of iterations
conv = 1                    # convergence criterion

while((iter < maxiter) && (conv > tol))
{
    iter = iter + 1
    x_cur = x
    fp = ...                    # compute 1st-order derivative at x
    fpp = ...                   # compute 2nd-order derivative at x
    x = x - fp / fpp
    conv = abs(x - x_cur)
}
```

# Example: MLE for DF in Chi-squares

The sample is drawn from $\chi^2$ with unknown df.

```
x = c(6.9, 5.4, 2.3, 5.0, 6.1, 6.8, 3.5, 1.6, 3.4, 2.9)
```

Let $X_1, \ldots, X_n$ be iid $\chi^2_d$ random variables with density

$$f(x; d) = \frac{1}{2^{d/2}\Gamma(d/2)} x^{d/2-1} e^{-x/2}, \quad x \geq 0.$$

The method of moment estimator of $d$ is $\bar{X}$.

Find the MLE of $d$ and its standard error. The log-likelihood is

$$\ell(d) = -\frac{nd}{2}\log 2 - n\log\Gamma(d/2) + (d/2-1)\sum_{i=1}^{n}\log x_i - \sum_{i=1}^{n} x_i/2.$$

# MLE of $d$ in Chi-squared Distribution

The MLE of $d$ is the root of $\ell'(d) = 0$.

$$\ell'(d) = -\frac{n}{2}\log 2 - \frac{n}{2}\frac{\partial \log \Gamma(d/2)}{\partial d} + \frac{1}{2}\sum_{i=1}^{n}\log x_i$$

$$\ell''(d) = -\frac{n}{4}\frac{\partial^2 \log \Gamma(d/2)}{\partial d^2}.$$

The updating equation is

$$d^{(t+1)} = d^{(t)} - \frac{\ell'(d^{(t)})}{\ell''(d^{(t)})}$$

Functions lgamma(), digamma() and trigamma() compute the logarithm of $\Gamma(\cdot)$ and its first and second derivatives, respectively.

# Comments

- What is a proper starting value?
  - graph
  - preliminary estimates
  - educated guess
  - random selection
- What is the standard error of the MLE $\hat{d}$?
- Is the MLE $\hat{d}$ more efficient than $\bar{X}$?

More comments for Newton's method in general.

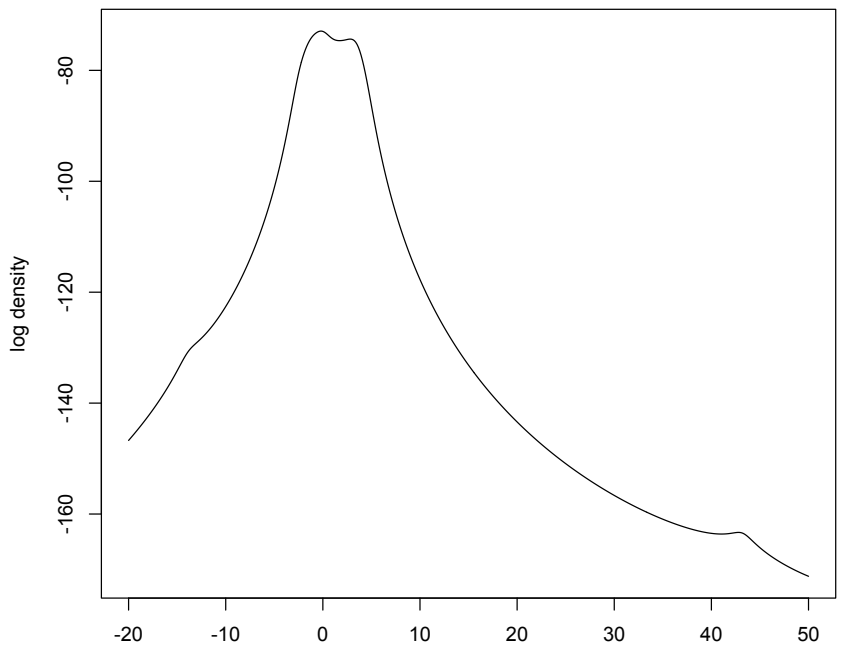
- An iterative algorithm may converge, diverge, or cycle.
- Whether Newton's methods converges depends on the shape of $m$ (or equivalently $f$) and the starting value.
- one root or multiple roots, local minimum or global minimum
- Try different starting values

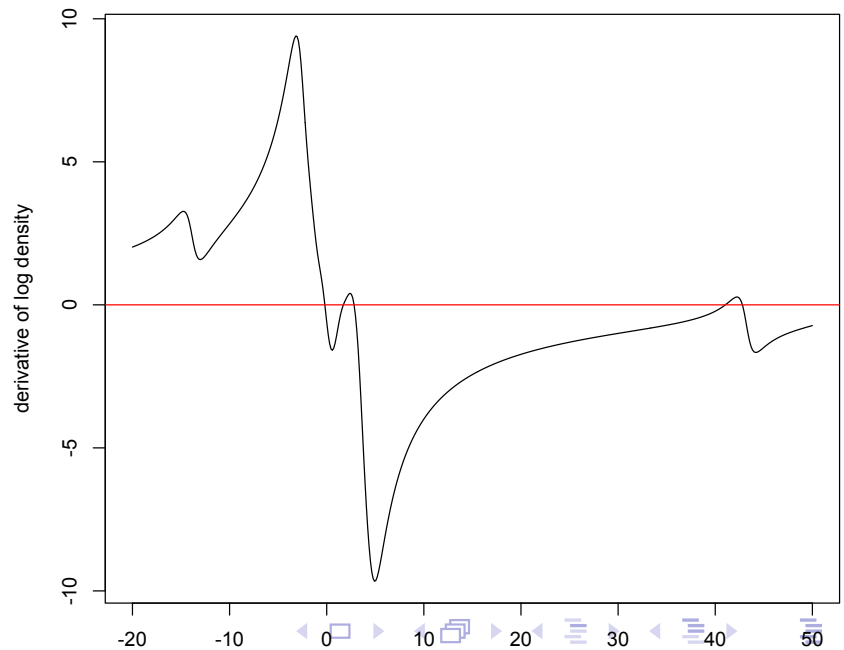
# Example: Cauchy Distribution

```
x = c(1.77, -0.23, 2.76,  3.80, 3.47, 56.75, -1.34,  4.24,
      -2.44,  3.29, 3.71, -2.40, 4.53, -0.07, -1.05, -13.87,
      -2.53, -1.75, 0.27, 43.21)
```

Find the MLE of $\theta$ in Cauchy($\theta$, 1), where the density is

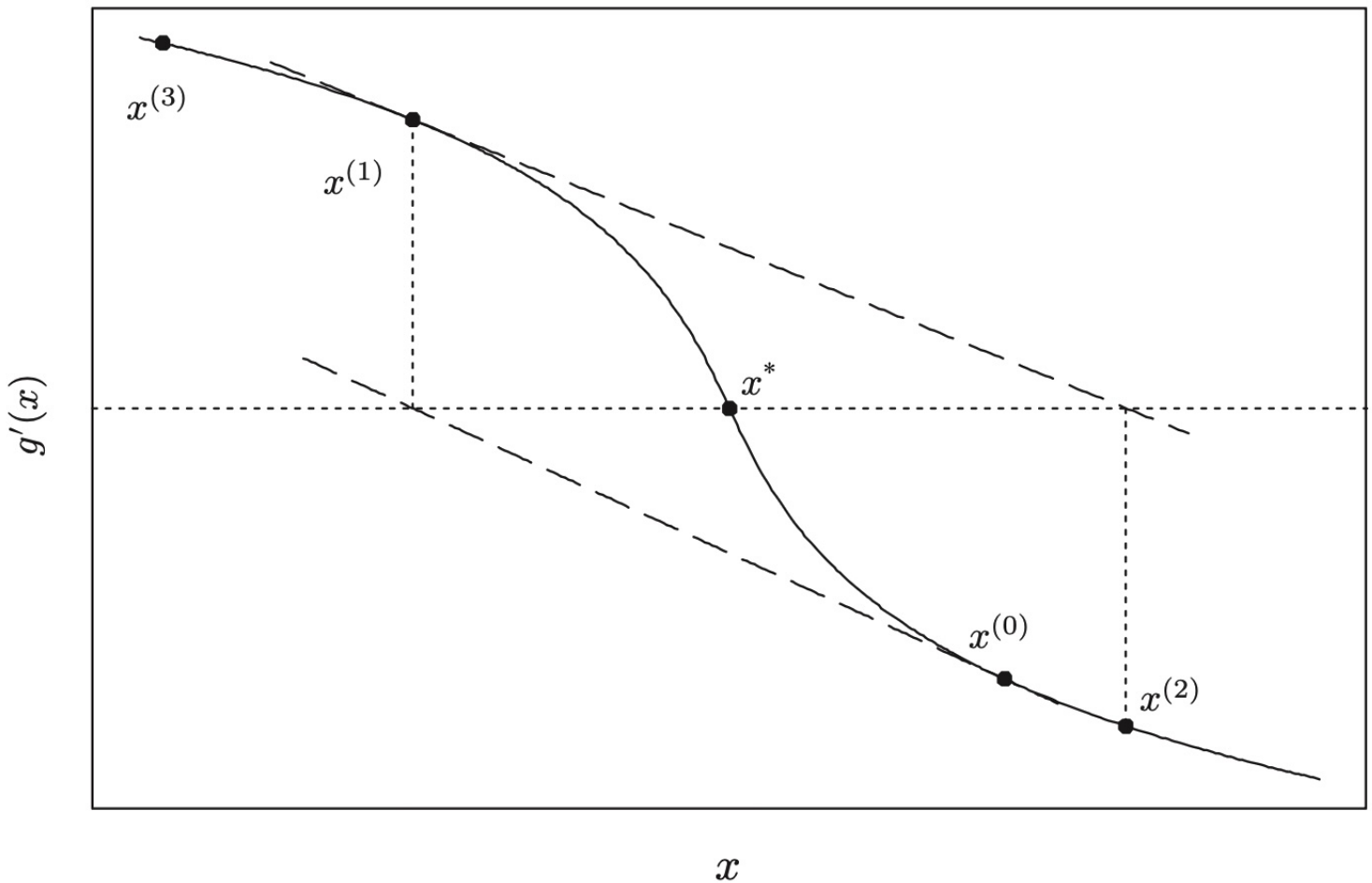
$$f(x;\theta) = \frac{1}{\pi(1+(x-\theta)^2)}$$



# Newton's Method May Fail

The Newton's method cannot guarantee $f(x^{(t+1)}) > f(x^{(t)})$.



# Convergence of Newton's Method

The Newton's method will converge to the root of $m(x) = 0$ from any starting point if

- Scenario 1:
  - $m(x)$ is twice continuously differentiable
  - $m(x)$ is convex
  - $m(x)$ has one root
- Scenario 2: Consider $m(x)$ in an interval $[a, b]$
  - $m'(x) \neq 0$ on $[a, b]$
  - $m''(x)$ does not change sign on $[a, b]$
  - $m(a)m(b) < 0$
  - $|m(a)/m'(a)| < b - a$ and $|m(b)/m'(b)| < b - a$

# Convergence Order of Newton's Method

A sequence $\{x^{(n)}\}$ has convergence of order $\beta$ if

$$\lim_{n\to\infty} x^{(n)} = x^*, \quad \text{and} \quad \lim_{n\to\infty} \frac{\left|x^{(n+1)} - x^*\right|}{\left|x^{(n)} - x^*\right|^{\beta}} = c \neq 0$$

Algorithm with higher order of convergence are faster, but usually less robust.

Newton's method has quadratic convergence ($\beta = 2$) with

$$c = \left| \frac{m''(x^*)}{2m'(x^*)} \right|$$

The precision of the solution will double with each iteration.

# Secant Method

Consider

$$\max_x f(x)$$

When the second-order derivative $f''(x)$ is difficult to evaluate, it can be replaced by the discrete-difference approximation. Hence, the secant method has the updating equation

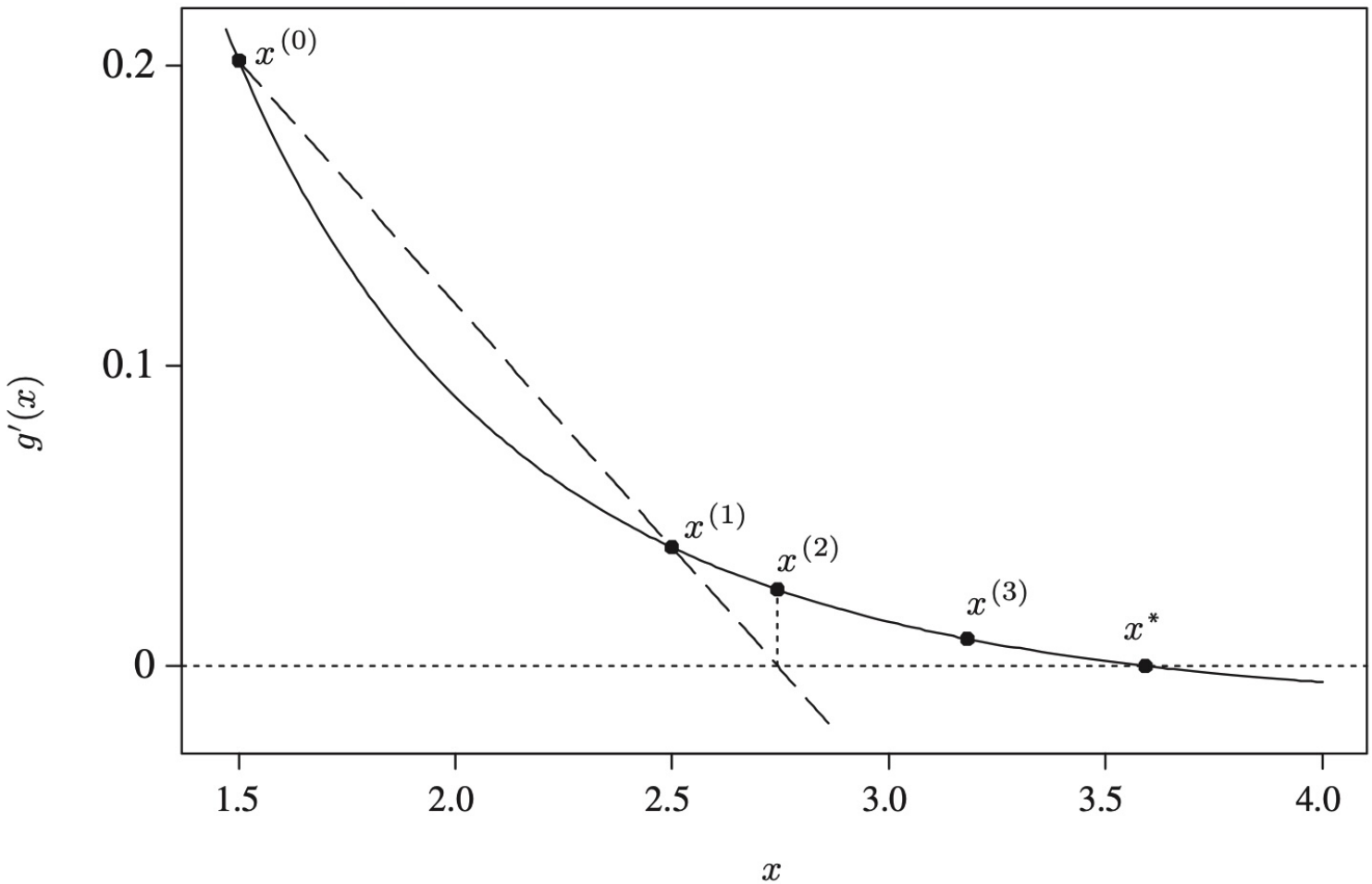
$$x^{(t+1)} = x^{(t)} - f'(x^{(t)}) \frac{x^{(t)} - x^{(t-1)}}{f'(x^{(t)}) - f'(x^{(t-1)})}$$

This method requires two starting points, $x^{(0)}$ and $x^{(1)}$.

The secant method has a slower order of convergence than Newton's method. $\beta = (1 + \sqrt{5})/2 \approx 1.62$.

# Interpretation of Secant Method

The line passing $x^{(t-1)}$ and $x^{(t)}$ intersects the horizontal axis at $x^{(t+1)}$.



# Fixed-Point Iteration

A fixed-point of a function is a point whose evaluation by that function equals itself.

$$x = G(x)$$

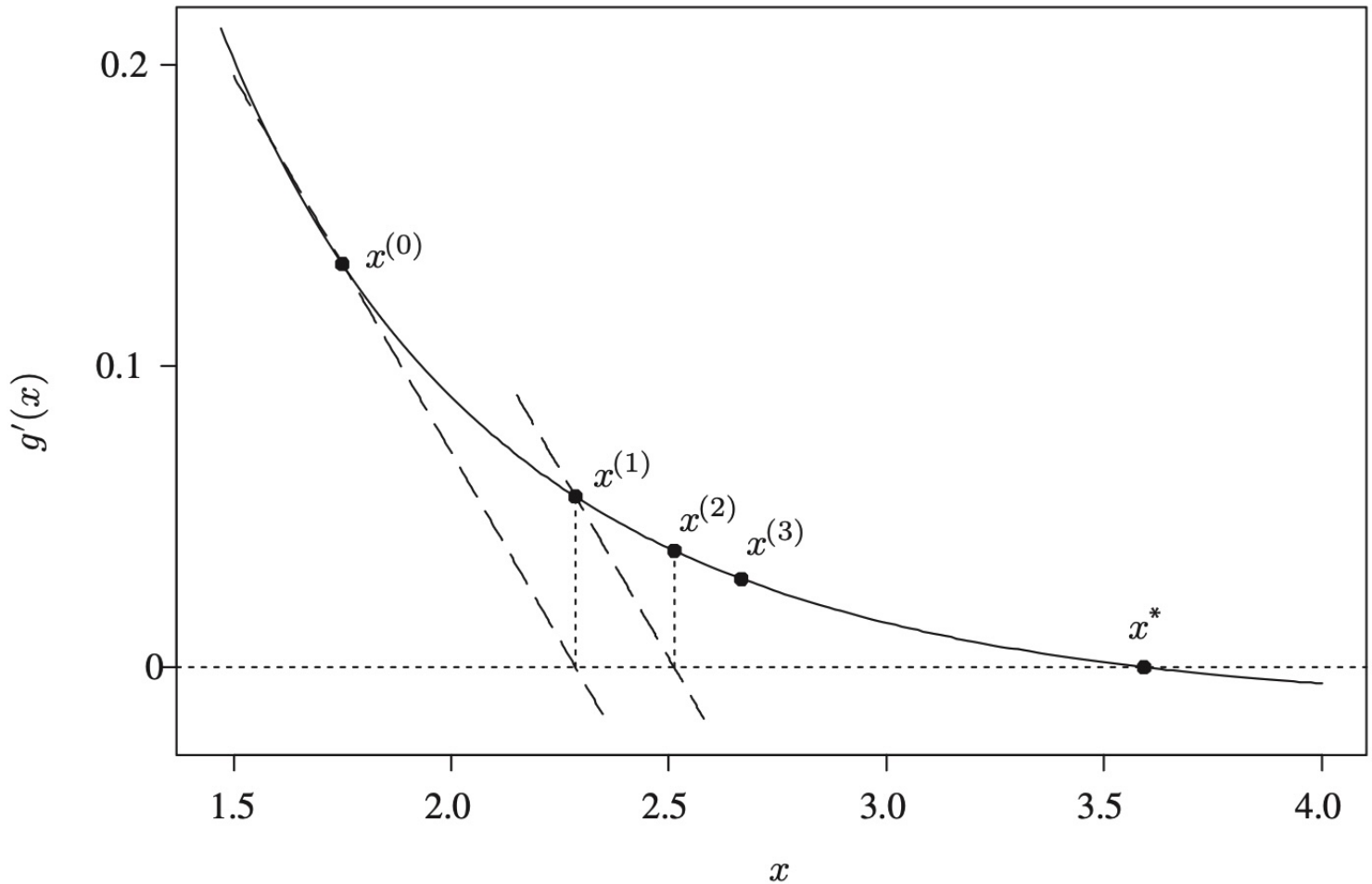
If we can find a function $G$ such that $x = G(x)$ if and only if $f'(x) = 0$, then the updating equation to solve $\max f(x)$ is

$$x^{(t+1)} = G(x^{(t)})$$

One obvious choice is $G(x) = f'(x) + x$. Hence,

$$x^{(t+1)} = x^{(t)} + f'(x^{(t)})$$

# Fixed-Point Method



# Comments

- The fixed-point iteration converges if $G$ is contractive.
  - $G(x) \in [a, b]$ whenever $x \in [a, b]$
  - For all $x_1, x_2 \in [a, b]$ and some $\lambda \in [0, 1)$,

$$|G(x_1) - G(x_2)| \leq \lambda |x_1 - x_2|$$

    It is a Lipschitz condition and $\lambda$ is the Lipschitz constant.
- Both Newton's method and the secant method are special cases of fixed-point iteration.
- Effectiveness of fixed point iteration depends strongly on the chosen functional form of $G$.

# Scaled Fixed-Point Iteration

- It converges if $|G'(x)| \leq \lambda < 1$ for all $x \in [a, b]$.
- Consider solving $\max f(x)$. When $f''(x)$ is bounded and does not change sign on $[a, b]$, choose

$$G(x) = af'(x) + x$$

where $a \neq 0$ and $|af''(x) + 1| < 1$ on $[a, b]$.

- If the log-likelihood is nearly quadratic near $\hat{\theta}$, $\ell''$ is roughly a constant, say $\gamma$. The updating equation can be

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\ell'(\theta)}{\gamma}$$

where we choose $a = -1/\gamma$.

# How to Describe an Algorithm

- Objective: the objective function $f(\theta)$ and the goal (min/max)
- Input arguments:
  - (required) data
  - (optional) initial value, tolerance, max number of iterations
- Output arguments
  - estimated solution $\hat{\theta}$
  - value of the objective function $f(\hat{\theta})$ at the solution $\hat{\theta}$
  - number of iterations performed
  - convergence status or criterion
- Initialization:
  - specify initial values $\theta^{(0)}$ if not provided
  - precompute or cache intermediate quantities as needed
- Updating equation: define how to compute $\theta^{(t+1)}$ from $\theta^{(t)}$
- Stopping rule: absolute or relative convergence criterion

# Example: Compute MLE of DF in Chi-square

The algorithm to compute the MLE of $d$ in $\chi^2_d$ is as follows.

- Input argument: $x = (x_1, \ldots, x_n)$
- Initialization:
  - initial value $d^{(0)} = \bar{x}$
  - default tolerance $\varepsilon = 10^{-5}$ and maxiter $= 100$
  - compute $c = \sum_{i=1}^{n} \log x_i / n - \log 2$
- Updating equation:

$$d^{(t+1)} = d^{(t)} - \frac{\phi'(d^{(t)}/2) - c}{\phi''(d^{(t)}/2)/2}$$

  where $\phi'(d) = (\log \Gamma(d))'$ and $\phi''(d) = (\log \Gamma(d))''$.
- Stopping rule: absolute convergence criterion $|d^{(t+1)} - d^{(t)}|$

# Code Template

```
find_mle = function(x, tht0 = NULL, tol = 1e-5, maxiter = 100)
{
    if(is.null(tht0)) tht = 0 else tht = tht0

    iter = 0; conv = 1;
    while((iter < maxiter) && (conv > tol))
    {
        iter = iter + 1
        tht_cur = tht
        tht = update_solution_from_current(tht_cur)
        conv = abs(tht - tht_cur)
    }

    loglik = evaluate_log_likelihood(tht, x)
    list(tht = tht, loglik = loglik, iter = iter, conv = conv)
}
```

# Comments

- Pre-computation: precompute constant or reusable quantities to reduce computational cost and improve efficiency.
- Validation using simulated data
  - Are the estimates close to the true parameter values?
  - Does estimation performance improve as the sample size increases?
- Simple test cases first: start with simple scenarios where the true solution or an explicit/closed-form solution is available.
- Progress to larger-scale simulations: conduct simulations under more complex and diverse scenarios, including different sample size and dimension, randomly generated true parameter values.

# Multivariate Problem

Many of the general principles discussed for the univariate case also apply for multivariate optimization.

$$\max_{x \in \mathbb{R}^p} f(x), \quad \text{where } f(x) = f(x_1, \ldots, x_p).$$

When $f(x)$ is differentiable, it is equivalent to find the root of

$$m(x) = f'(x) = \frac{\partial f(x)}{\partial x} = \left( \frac{\partial f(x)}{\partial x_i} \right)_{p \times 1} = \begin{pmatrix} \frac{\partial f(x)}{\partial x_p} \\ \vdots \\ \frac{\partial f(x)}{\partial x_p} \end{pmatrix} = 0$$

# Newton's Method

Newton's method for solve a system of equations $m(x) = 0$ starts with an initial value $x^{(0)}$ and keep updating

$$x^{(t+1)} = x^{(t)} - J_m^{-1}(x^{(t)})m(x^{(t)}).$$

where the Jacobian $J_m$ is

$$J_m = \frac{\partial m(x)}{\partial x^T} = \frac{\partial^2 f(x)}{\partial x \partial x^T} = \left( \frac{\partial^2 f(x)}{\partial x_i \partial x_j} \right)_{p \times p}$$

The updating equation for solving $\max f(x)$ is

$$x^{(t+1)} = x^{(t)} - \left[ \frac{\partial^2 f(x^{(t)})}{\partial x \partial x^T} \right]^{-1} \frac{\partial f(x^{(t)})}{\partial x}$$

# Convergence Criterion

The absolute and relative convergence criteria are

$$D(x^{(t+1)}, x^{(t)}) < \delta, \quad \frac{D(x^{(t+1)}, x^{(t)})}{D(x^{(t)}, 0)} < \delta, \quad \frac{D(x^{(t+1)}, x^{(t)})}{D(x^{(t)}, 0) + \delta} < \delta,$$

where $D(u, v)$ be a distance measure, for example,

$$D(u, v) = \|u - v\|_2, \qquad D(u, v) = \|u - v\|_1.$$

Make sure to set a maximum number of iterations $N$ and stop iterations if the number of iterations reaches $N$.

# Code Template

```
tol = 1e-5                  # tolerance
maxiter = 100               # max number of iterations

x = rep(0, p)               # initial value
iter = 0                    # number of iterations
conv = 1                    # convergence criterion

while((iter < maxiter) && (conv > tol))
{
    iter = iter + 1
    x_cur = x
    grad = ...                      # compute gradient at x
    hess = ...                      # compute Hessian at x
    x = x - solve(hess, grad)
    conv = sum(abs(x - x_cur))
}
```

# Example: MLE for Weibull Distribution

Let $X_1, \ldots, X_n$ be iid Weibull($\alpha$, $\lambda$) random variables with density

$$f(x; \alpha, \lambda) = \alpha\lambda x^{\alpha-1} \exp(-\lambda x^\alpha), \qquad x > 0.$$

Find the MLE of $\theta = (\alpha, \lambda)^T$. The log-likelihood function is

$$\ell(\theta) = n \log \alpha + n \log \lambda + (\alpha - 1) \sum \log(x_i) - \lambda \sum x_i^\alpha.$$

The score function $U(\theta)$ and the Hessian matrix $H(\theta)$ are

$$U(\theta) = \begin{pmatrix} n\alpha^{-1} + \sum \log(x_i) - \lambda \sum x_i^\alpha \log(x_i) \\ n\lambda^{-1} - \sum x_i^\alpha \end{pmatrix}$$

$$H(\theta) = \begin{pmatrix} -n\alpha^{-2} - \lambda \sum x_i^\alpha [\log(x_i)]^2 & -\sum x_i^\alpha \log(x_i) \\ -\sum x_i^\alpha \log(x_i) & -n\lambda^{-2} \end{pmatrix}$$

# Fisher Scoring

In an MLE problem, Newton's method updates the estimate by

$$\theta^{(t+1)} = \theta^{(t)} - \left[H(\theta^{(t)})\right]^{-1} U(\theta^{(t)})$$

where the score function $U(\theta)$ and Hessian matrix $H(\theta)$ are

$$U(\theta) = \frac{\partial \ell(\theta)}{\partial \theta}, \qquad H(\theta) = \frac{\partial^2 \ell(\theta)}{\partial \theta \partial \theta^T}$$

Sometimes, the negative Hessian matrix $-H(\theta)$ can be replaced by the expected Fisher information $I(\theta)$, which yields Fisher scoring.

$$\theta^{(t+1)} = \theta^{(t)} + \left[I(\theta^{(t)})\right]^{-1} \frac{\partial \ell(\theta^{(t)})}{\partial \theta}$$

In some applications, $I(\theta)$ has a simpler expression than $-H(\theta)$.

# Example: Logistic Regression

Let $(x_i, y_i)$, $i = 1, \ldots, n$ be an iid sample from

$$y_i \sim \text{Bernoulli}(\pi_i), \quad \text{logit}(\pi_i) = \log \frac{\pi_i}{1-\pi_i} = \beta^T x_i$$

The log-likelihood function is

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

The score function and the Fisher information matrix are

$$U(\theta) = \frac{\partial \ell(\beta)}{\partial \beta} = \sum_{i=1}^{n} \frac{\partial \ell(\beta)}{\partial \pi_i} \frac{\partial \pi_i}{\partial \beta} = \sum_{i=1}^{n} (y_i - \pi_i) x_i$$

$$I(\beta) = -E\Big(\frac{\partial^2 \ell(\beta)}{\partial \beta \partial \beta^T}\Big) = \sum_{i=1}^{n} \pi_i (1 - \pi_i) x_i x_i^T$$

# Updating Equation

The updating equation is thus

$$\beta^* = \beta + \left[\sum_{i=1}^{n} \pi_i(1-\pi_i)x_i x_i^T\right]^{-1} \sum_{i=1}^{n}(y_i - \pi_i)x_i$$

or equivalently, let $w_i = \pi_i(1-\pi_i)$ and $z_i = \beta^T x_i + (y_i - \pi_i)/w_i$

$$\begin{aligned}\beta^* &= \left[\sum_{i=1}^{n} w_i x_i x_i^T\right]^{-1} \left\{\sum_{i=1}^{n} w_i x_i x_i^T \beta + (y_i - \pi_i)x_i\right\} \\ &= \left[\sum_{i=1}^{n} w_i x_i x_i^T\right]^{-1} \sum_{i=1}^{n} w_i x_i z_i\end{aligned}$$

# Iterative Reweighted Least Squares

- Find an initial value of $\beta^{(0)}$. For example, set $\beta^{(0)} = 0$ or LSE.
- For $t = 0, 1, \ldots$, calculate $\pi_i^{(t)}$ from $\beta^{(t)}$, the working response $z_i^{(t)}$, and the weight $w_i^{(t)}$,

$$z_i^{(t)} = x_i^T \beta^{(t)} + \frac{(y_i - \pi_i^{(t)})}{\pi_i^{(t)}(1 - \pi_i^{(t)})}, \qquad w_i^{(t)} = \pi_i^{(t)}(1 - \pi_i^{(t)})$$

  Fit a weighted least squares problem using $z_i^{(t)}$ as response, $x_i$ as predictor and $w_i^{(t)}$ as weight. The resulting estimate is $\beta^{(t+1)}$.
- Repeat the above step until convergence.

# Newton-Like Methods

Some very effective methods rely on updating equations of the form

$$x^{(t+1)} = x^{(t)} - \left(M^{(t)}\right)^{-1} f'(x^{(t)})$$

where $M^{(t)}$ is a matrix approximating the Hessian $f''(x^{(t)})$.

- $M^{(t)}$ is easier to compute than the Hessian matrix
- The steps taken by the original Newton's method are not necessarily always uphill.
- One choice is $M^{(t)} = -I(\theta^{(t)})$ for the Fisher scoring

# The Ascent Algorithm

Consider

$$\max_x f(x).$$

The gradient of $f$ is the direction of the steepest ascent.

$$x^{(t+1)} = x^{(t)} + \alpha^{(t)} f'(x^{(t)})$$

where the step length $\alpha^{(t)} > 0$ controls convergence.

It is gradient descent method when minimizing a function $\min g(x)$.

$$x^{(t+1)} = x^{(t)} - \alpha^{(t)} g'(x^{(t)})$$

# Backtracking

Or generally, the updating equation is

$$x^{(t+1)} = x^{(t)} - \alpha^{(t)} \left(M^{(t)}\right)^{-1} f'(x^{(t)})$$

When $-M^{(t)}$ is positive definite, ascent can be assured by choosing $\alpha^{(t)}$ sufficiently small. By the Taylor expansion,

$$f(x^{(t+1)}) - f(x^{(t)}) = -\alpha^{(t)} f'(x^{(t)})^T [M^{(t)}]^{-1} f'(x^{(t)}) + o(\alpha^{(t)})$$

Hence, $f(x^{(t+1)}) - f(x^{(t)}) > 0$ when $-M^{(t)}$ is positive definite, because $o(\alpha^{(t)})/\alpha^{(t)} \to 0$ as $\alpha^{(t)} \to 0$.

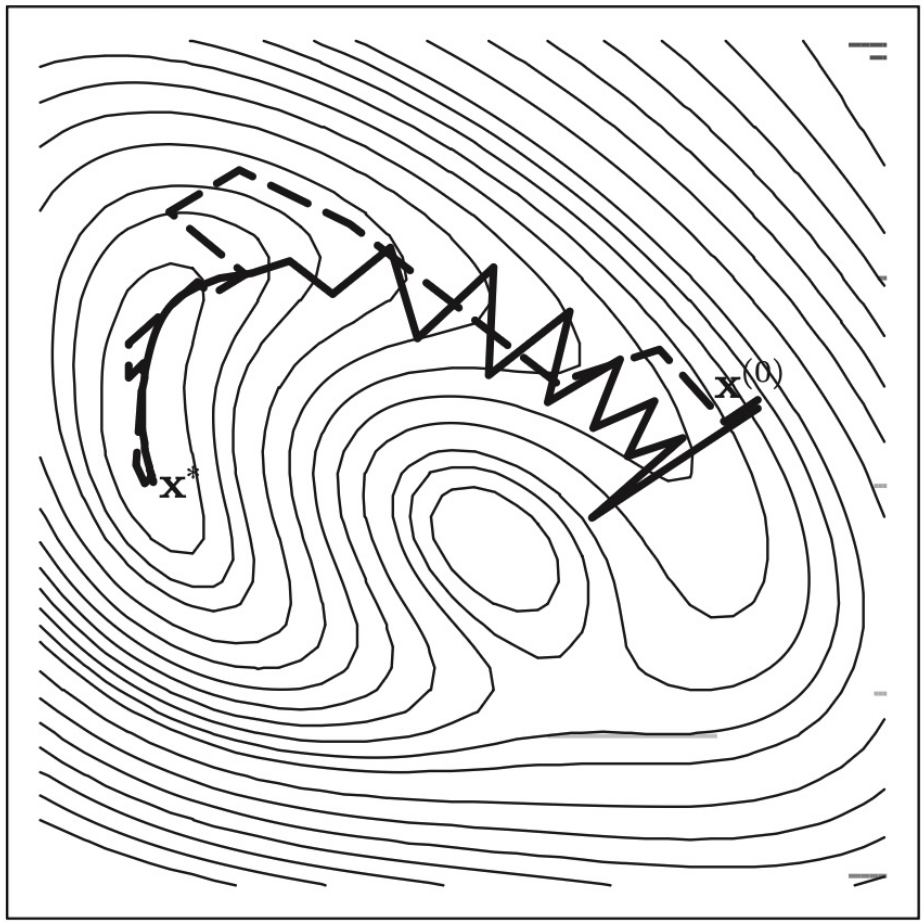
Backtracking: Start with $\alpha^{(t)} = 1$. If the original step turns out to be downhill, reduce $\alpha^{(t)}$ by half. Repeat the step if necessary.

# Code Template

```
tol = 1e-5; maxiter = 100; inner_maxiter = 10;
x = rep(0, p); f_value = f(x); iter = 0; conv = 1;
while((iter < maxiter) && (conv > tol)) {
    iter = iter + 1
    x_cur = x;   f_cur = f_value;     # save current value
    direction = ...                   # compute moving direction
    x = x_cur + direction             # compute updated x
    f_value = f(x)                    # compute updated f(x)
    inner_iter = 0
    while((inner_iter < inner_maxiter) && (f_value < f_cur))  {
        inner_iter = inner_iter + 1
        direction = direction / 2     # reduce alpha by half
        x = x_cur + direction         # recompute updated x
        f_value = f(x)                # recompute updated f(x)
    }
    if(f_value < f_cur) warning(message);   # still downhill
    conv = sum(abs(x - x_cur))
}
```

# The Steepest Ascent May not Be the Best

The solid line corresponds to the steepest ascent method, and the dashed line corresponds to the BFGS method.



# Modified Newton Method

When the step direction is not uphill, we can also modify the step direction. A vector $g$ is called a ascent direction if $g^T f'(x) < 0$.

$$x^{(t+1)} = x^{(t)} + \alpha^{(t)} g^{(t)},$$

where $\alpha^{(k)}$ is the step length.

When the negative Hessian is not positive definite, one strategy is to replace it by

$$-\frac{\partial^2 f(x^{(t)})}{\partial x \partial x^T} + E$$

where $E$ is a diagonal matrix with nonnegative elements.

# Discrete Newton Method

Discrete Newton method is similar to secant method, and avoids calculating the Hessian. For the $(i, j)$th element of $M^{(t)}$

$$M_{ij}^{(t)} = \frac{f_i'(x^{(t)} - h_{ij}^{(t)} e_j) - f_i'(x^{(t)})}{h_{ij}^{(t)}}$$

where constant $h_{ij}^{(t)} > 0$.

- Set $h_{ij}^{(t)} = h$ for all $(i, j)$ and $t$.
- Set $h_{ij}^{(t)} = x_j^{(t)} - x_j^{(t-1)}$ for all $i$.
- Average $M^{(t)}$ with its transpose to ensure symmetry.

# Multivariate Fixed-Point Method

Multivariate fixed-point method uses an initial approximation of $f''$ throughout the iterative updating.

A reasonable choice is $M = f''(x^{(0)})$, which yields

$$x^{(t+1)} = x^{(t)} - M^{-1} f'(x^{(t)})$$

When $M$ is diagonal, it essentially applied the univariate scaled fixed-point algorithm separately to each component of $x$.

# Quasi-Newton Methods

A secant condition holds for $M^{(t+1)}$ if

$$f'(x^{(t+1)}) - f'(x^{(t)}) = M^{(t+1)}(x^{(t+1)} - x^{(t)})$$

From $M^{(t)}$, quasi-Newton method generates $M^{(t+1)}$ that meets this condition.

There is a unique symmetric rank-one method.

$$M^{(t+1)} = M^{(t)} + c^{(t)} v^{(t)} (v^{(t)})^T,$$

where $z^{(t)} = x^{(t+1)} - x^{(t)}$ and $c^{(t)} = 1/[(v^{(t)})^T z^{(t)}]$ and

$$v^{(t)} = f'(x^{(t+1)}) - f'(x^{(t)}) - M^{(t)} z^{(t)},$$

# Comment

- If $c^{(t)}$ cannot be reliably caculalted when the denominator close to zero, a temporary solution is to take $M^{(t+1)} = M^{(t)}$ for that iteration.
- Need to backtrack to ensure ascent.
  - If $-M^{(t)}$ is positive definite and $c^{(t)} \leq 0$, then $-M^{(t+1)}$ will be postive definite.
  - If $c^{(t)} > 0$, it may be necessary to backtrack by shrinking $c^{(t)}$ toward zero until positive definiteness is achieved.

# BFGS Method

There are several symmetric rank-two methods to approximate the Hessian. One popular one is the BFGS update,

$$M^{(t+1)} = M^{(t)} - \frac{M^{(t)} z^{(t)} \left(M^{(t)} z^{(t)}\right)^T}{\left(z^{(t)}\right)^T M^{(t)} z^{(t)}} + \frac{y^{(t)} \left(y^{(t)}\right)^T}{\left(z^{(t)}\right)^T y^{(t)}}$$

where $z^{(t)} = x^{(t+1)} - x^{(t)}$ and $y^{(t)} = f'(x^{(t+1)}) - f'(x^{(t)})$.

- The BFGS update confers hereditary positive definiteness on $-M^{(t)}$, that is, the positive definiteness is guaranteed.
- Backtracking can ensure ascent.
- The convergence order of quasi-Netwon methods is in $(1, 2)$.
- Select the starting matrix $M^{(0)}$ as the negative identity.
- It is suggested to rescale the problem so that the elements of $x$ are on comparable scales.

# BFGS for MLE

In MLE problem, choose $M^{(0)} = -I(\theta^{(0)})$.

Do not use $M^{(t)}$ as an approximation to estimated variance. One approach is to rely on the central difference approximation, whose $(i, j)$th element is

$$\widehat{\ell''(\theta^{(t)})} = \frac{\ell_i'(\theta^{(t)} + h_{ij}e_j) - \ell_i'(\theta^{(t)} - h_{ij}e_j)}{2h_{ij}}$$

where $\ell_i'(\theta^{(t)})$ is the $i$th component of the score function evaluated at $\theta^{(t)}$. One rule of thumb is to take $h_{ij} = h = \varepsilon^{1/3}$, where $\varepsilon$ represents the computer's floating-point precision.

# Nonlinear Least Squares

Let $(y_i, x_i), i = 1, \ldots, n$ be iid from a nonlinear regression,

$$y_i = m(x_i; \theta) + \varepsilon_i, \qquad i = 1, \ldots, n.$$

Estimate $\theta$ by minimizing nonlinear least squares,

$$\min_{\theta} \sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} (y_i - m(x_i; \theta))^2$$

where $r_i = y_i - m(x_i; \theta)$ are residuals. Gauss-Newton method updates the solution by

$$\theta^{(t+1)} = \theta^{(t)} + (J_m^T J_m)^{-1} J_m^T r(\theta^{(t)}),$$

where $J_m$ is an $n \times p$ matrix whose $i$th row is $\partial m(x_i; \theta)/\partial \theta^T$.

$$J_m = \left( \frac{\partial m(x_i, \theta^{(t)})}{\partial \theta^T} \right)_{n \times p}$$

# Gauss-Newton Method

The Gauss-Newton algorithm can be understand from two different perspectives.

- Linear approximation: $r_i = r_i(\theta^{(t)}) + J_m(\theta^{(t)})(\theta - \theta^{(t)})$

$$\min_{\theta} \sum_{i=1}^{n} \left( r_i(\theta^{(t)}) + J_m(\theta^{(t)})(\theta - \theta^{(t)}) \right)^2$$

- Newton-Raphson algorithm: approximate Hessian matrix by $J_m^T J_m$

# Example

Let $(x_i, y_i)$, $i = 1, \ldots, n$ be sampled from a nonlinear regression model

$$y = m(x; \theta) + \varepsilon = \frac{\theta_1 x}{x + \theta_2} + \varepsilon.$$

Want to find $\theta = (\theta_1, \theta_2)^T$.

$$\frac{\partial m(x)}{\partial \theta} = \begin{pmatrix} \frac{x}{x+\theta_2} \\ -\frac{\theta_1 x}{(x+\theta_2)^2} \end{pmatrix}, \qquad \frac{\partial^2 m(x)}{\partial \theta \partial \theta^T} = \begin{pmatrix} 0 & -\frac{x}{(x+\theta_2)^2} \\ -\frac{x}{(x+\theta_2)^2} & \frac{2(\theta_1 x)}{(x+\theta_2)^3} \end{pmatrix}$$

# Nonlinear Gauss-Seidel Iteration

The nonlinear Gauss-Seidel iteration is also referred to as back-fitting or cyclic coordinate ascent.

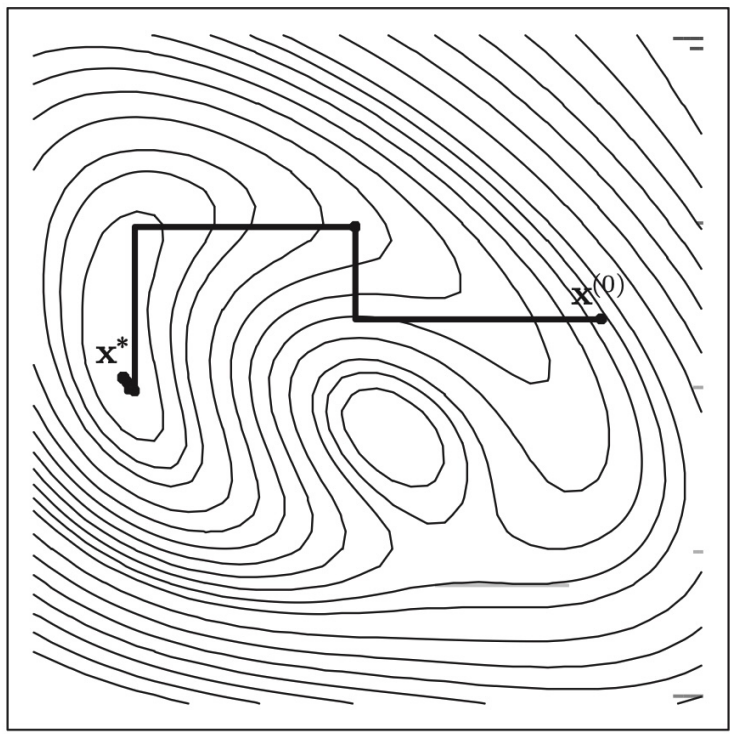
$$\max_x f(x_1, \ldots, x_p).$$

The Gauss-Seidel iteration proceeds by treating $f$ as a univariate function of $x_j$ in each step. For given $x^{(t)}$, one iteration consists of

- solve $x_1^{(t+1)} = \arg\max_{x_1} f(x_1, x_2^{(t)}, \ldots, x_p^{(t)})$
- solve $x_2^{(t+1)} = \arg\max_{x_2} f(x_1^{(t+1)}, x_2, x_3^{(t)}, \ldots, x_p^{(t)})$
- $\ldots$
- solve $x_p^{(t+1)} = \arg\max_{x_p} f(x_1^{(t+1)}, \ldots, x_{p-1}^{(t+1)}, x_p)$
- output $x^{(t+1)} = (x_1^{(t+1)}, \ldots, x_p^{(t+1)})$

# Illustration of Gauss-Seidel Iteration

All $p$ components are cycled through in succession, and at each stage of the cycle the most recent values obtained for each coordinate are used. At the end of the cycle, the complete set of most recent values constitutes $x^{(t+1)}$.



# Comments

- Gauss-Seidel iteration converts a multiviariate optimization problem to a series of univariate optimization problems.
- Solution of each univariate problem is easier and more stable.
- Any convenient univariate optimization method can be used.
- Often less clock time required due to simple computations

# Coordinate Descent for Lasso

Let $(y_i, x_i)$ be drawn from a linear regression. Estimate $\beta$ by lasso,

$$\min_{\beta} \frac{1}{2} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2 + \lambda \|\beta\|_1, \quad \lambda > 0.$$

Treat it as a function of $\beta_k$ and keep $\beta_{-k}$ fixed.

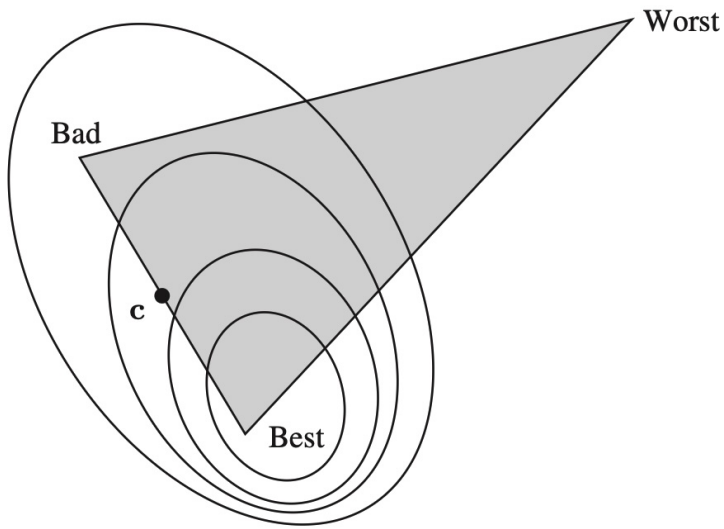
$$\hat{\beta}_k = \frac{\mathsf{soft}(X_k^T y - X_k^T X_{-k} \beta_{-k}; \lambda)}{X_k^T X_k}$$

where $y \in \mathbb{R}^n$ is the response, $X_k$ is the vector of the $k$th predictor, $X_{-k}$ is the matrix consisting of all predictors except the $k$th.

$$\mathsf{soft}(u; \lambda) = \mathsf{sign}(u)(|u| - \lambda)_+ = \begin{cases} u - \lambda, & u > \lambda \\ 0, & -\lambda \le u \le \lambda \\ u + \lambda, & u < -\lambda \end{cases}$$

# Nelder-Mead Algorithm

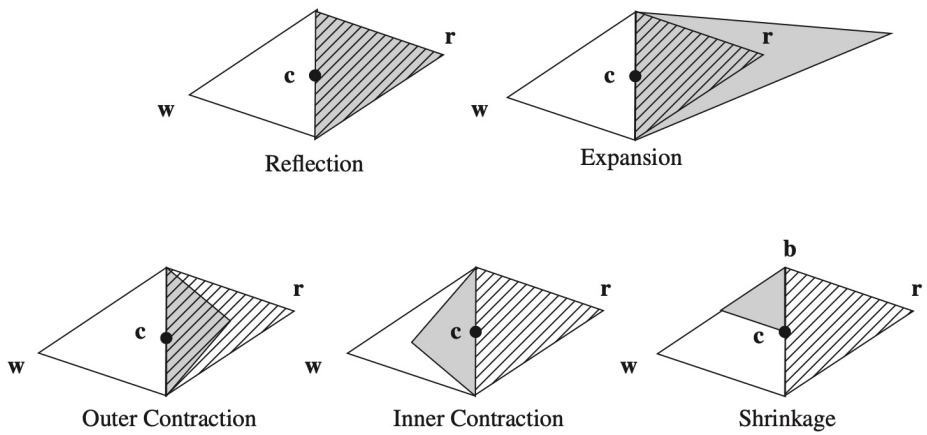
The Nelder-Mead algorithm (also called simplex search method) requires no derivative information. It is an iterative discrete search approach.




When $x \in \mathbb{R}^p$, $p+1$ distinct points $x_1, \ldots, x_{p+1}$ define a $p$-dimensional simplex. The vertices of the simplex can be ranked from the best to worst according to the ranks of $f(x_1), \ldots, f(x_{p+1})$.

# Illustration of the Nelder-Mead Algorithm

Let the centroid $c = (1/p)[\sum_i x_i - x_{\text{worst}}]$ be the mean of points excluding the worst. Replace $x_{\text{worst}}$ by a point in the ray extending from $x_{\text{worst}}$ through $c$, which is called the search direction.




Let $x_r = 2c - x_{\text{worst}}$ be the reflection vertex.

- If $f(x_r) > f(x_{\text{bad}})$ and $f(x_r) < f(x_{\text{best}})$, replace $x_{\text{worst}}$ by $x_r$
- If $f(x_r) > f(x_{\text{bad}})$ and $f(x_r) > f(x_{\text{best}})$, expansion
- If $f(x_r) < f(x_{\text{bad}})$, contraction.

# Comments

- The Nelder-Mead method is generally quite good at finding optima, especially for low to moderate dimensions.
- For high-dimensional problems, its effectiveness is more varied, depending on the nature of the problem.
- The Nelder-Mead approach is quite robust in the sense that it can successfully find optima for a wide range of functions – even discontinuous ones – and from a wide range of starting values.
- The Nelder-Mead algorithm can perform poorly in certain circumstances. Surprisingly, it is even possible for the algorithm to converge to points that are neither local maxima nor minima.