STAT 5361: Statistical Computing

Jun Yan

Department of Statistics
University of Connecticut

Outline I

Non-stochastic Optimization

What we will learn:

- Some basics about statistical inference
- Univariate problems
  - Newton's method
  - Fisher scoring
  - Secant method
  - Fixed-point method
  - Connections of the above
- Multivariate problems
  - Newton's method
  - Newton-like methods

Background: likelihood inference

- Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ be an i.i.d. sample from

$$f(\boldsymbol{x} \,|\, \boldsymbol{\theta}^*),$$

  with the true parameter value $\boldsymbol{\theta}^*$ being unknown.
- The likelihood function is

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{n} f(\boldsymbol{x}_i \,|\, \boldsymbol{\theta}),$$

- The *maximum likelihood estimator* (MLE) of the parameter value is the maximizer of $L(\boldsymbol{\theta})$. MLE has the invariate property.
- Usually it is easier to work with the *log likelihood function*

$$l(\boldsymbol{\theta}) = \log L(\boldsymbol{\theta}).$$

- Typically, maximization of $l(\boldsymbol{\theta})$ is done by solving

$$l'(\boldsymbol{\theta}) = 0.$$

  - $l'(\boldsymbol{\theta})$ is called the *score function*.
  - For each possible parameter value $\boldsymbol{\theta}$, $l(\boldsymbol{\theta})$ is a random variable, because it depends on the observed values of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.

- For any $\boldsymbol{\theta}$,

$$\mathbb{E}_{\boldsymbol{\theta}} \left\{ l'(\boldsymbol{\theta}) \right\} = 0,$$
$$\mathbb{E}_{\boldsymbol{\theta}} \left\{ l'(\boldsymbol{\theta}) l'(\boldsymbol{\theta})^T \right\} = -\mathbb{E}_{\boldsymbol{\theta}} \left\{ l''(\boldsymbol{\theta}) \right\}.$$

  where $\mathbb{E}_{\boldsymbol{\theta}}$ is the expectation with respect to $f(\boldsymbol{x} \,|\, \boldsymbol{\theta})$. The equality holds true under mild regularity conditions.

- $$I(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\theta}} \left\{ l'(\boldsymbol{\theta}) l'(\boldsymbol{\theta})^T \right\}$$

  is known as the *Fisher information*.

- The importance of the Fisher information $I(\boldsymbol{\theta})$ is that it sets the limit on how accurate an unbiased estimate of $\boldsymbol{\theta}$ can be.
- As $n \to \infty$, the asymptotic distribution of $\sqrt{n}(\widehat{\boldsymbol{\theta}}_{\mathrm{MLE}} - \boldsymbol{\theta}^*)$ is $N_p(\mathbf{0}, nI(\boldsymbol{\theta}^*)^{-1})$. Since θ^* is unknown, the asymptotic covariance matrix $I(\boldsymbol{\theta}^*)^{-1})$ needs to be estimated.
  - If $\dim(\boldsymbol{\theta}) = 1$, $I(\boldsymbol{\theta})$ is a nonnegative number.
  - If $\dim(\boldsymbol{\theta}) > 1$, $I(\boldsymbol{\theta})$ is a nonnegative definite matrix.
- The *observed Fisher information* is

$$-l''(\boldsymbol{\theta}).$$

The expected Fisher information $I(\boldsymbol{\theta})$ may not be easily computed. The observed Fisher information is a good approximation to $I(\boldsymbol{\theta})$ that improves as n gets bigger.

Besides MLEs, there are other likelihoods for parameter estimation. Suppose $\boldsymbol{\theta}$ has two parts: $\boldsymbol{\theta} = (\boldsymbol{\phi}, \boldsymbol{\mu})$ and we are only interested in $\boldsymbol{\phi}$. The *profile likelihood* for $\boldsymbol{\phi}$ is

$$L(\boldsymbol{\phi}) := \max_{\boldsymbol{\mu}} L(\boldsymbol{\phi}, \boldsymbol{\mu}).$$

- $\boldsymbol{\mu}$ is the nuisance parameter.
- Need to maximize $L(\boldsymbol{\phi}, \boldsymbol{\mu})$ for every fixed $\boldsymbol{\phi}$. Also need to maximize $L(\boldsymbol{\phi})$.

A general method

Suppose $g(\boldsymbol{x})$ is a differentiable function, where

$$\boldsymbol{x} = (x_1, \ldots, x_n).$$

To find its maximum (or minimum), one method is to solve the equation

$$g'(\boldsymbol{x}) = 0$$

$$\text{where} \quad g'(\boldsymbol{x}) = \left(\frac{\partial g}{\partial x_1}, \ldots, \frac{\partial g}{\partial x_n} \right)^T.$$

Then the maximization is equivalent to solving

$$f(\boldsymbol{x}) = 0,$$

where $f = g'$.
For maximum likelihood estimation, g is the log likelihood function l, and $\boldsymbol{x}$ is the corresponding parameter vector $\boldsymbol{\theta}$.

Univariate Problems

Optimization: Univariate case

- Goal: optimize a real-valued function g with respect to its argument, a p dimensional vector $\boldsymbol{x}$. We will first consider the case $p = 1$.
- We will limit consideration mainly to smooth and differentiable functions.
- Root finding methods. Solving unconstrained nonlinear equations.
- Iterative algorithms: starting value, updating equation, stopping rule/convergence criterion.

Using bisection method to maximize

$$g(x) = \frac{\log x}{1 + x}.$$

or to sovle

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

Newton's method

This is a fast approach to finding roots of a differentiable function $f(x)$. First, set initial value x_0. Then for $t = 0, 1, \ldots$, compute

$$x_{t+1} = x_t + h_t, \text{ with } h_t = -\frac{f(x_t)}{f'(x_t)}.$$

Continue the iterations until x_t converges.

- Also known as Newton-Raphson iteration.
- Need to specify x_0.
- If $f(x) = 0$ has multiple solutions, the end result depends on x_0.

Newton's method require computing the derivative of a function. Algorithms are available to find root(s) of a univariate function without having to compute its derivative. For example, in R, one can use `uniroot`. Newton's method can be applied to optimize g by applying to

$$f = g'.$$

- Both g' (*gradient*) and g'' (*Hessian*) are needed.
- Many variants of Newton's method avoid the computation of the Hessian, which can be difficult especially for multivariate functions.

To maximize

$$g(x) = \frac{\log x}{1+x},$$

first find

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

$$f'(x) = g''(x) = -\frac{3 + 4/x + 1/x^2 - 2\log x}{(1+x)^3}.$$

So in the Newton's method,

$$h_t = \frac{(x_t+1)(1+1/x_t - \log x_t)}{3 + 4/x_t + 1/x_t^2 - 2\log x_t}.$$

Note that to solve $f(x) = 0$, one can instead solve $1 + 1/x - \log x = 0$. Treat this as a new f function. Then in the Newton's method,

$$h_t = x_t - \frac{x_t^2 \log x_t}{1+x_t} \quad \Longrightarrow \quad x_{t+1} = 2x_t - \frac{x_t^2 \log x_t}{1+x_t}.$$

To maximize the log likelihood $l(\theta)$, Newton's method computes

$$\theta_{t+1} = \theta_t - \frac{l'(\theta_t)}{l''(\theta_t)}$$

Example: consider $x_1, \ldots, x_n \sim$ i.i.d. $N(\mu, \sigma^2)$.

Example: consider the model on shift

$$p(x \mid \theta) = p(x - \theta).$$

Given observations $x_1, \ldots, x_n$ i.i.d. $\sim p(x \mid \theta)$

$$l(\theta) = \sum_{i=1}^{n} \log p(x_i - \theta)$$

$$l'(\theta) = -\sum_{i=1}^{n} \frac{p'(x_i - \theta)}{p(x_i - \theta)}$$

$$l''(\theta) = \sum_{i=1}^{n} \frac{p''(x_i - \theta)}{p(x_i - \theta)} - \sum_{i=1}^{n} \left\{ \frac{p'(x_i - \theta)}{p(x_i - \theta)} \right\}^2.$$

Note that we need to update θ in the iterations, not $x_1, \ldots, x_n$.

In R, to *minimize* a function, one can use

```
z = nlminb(x0, g, gr.g, hess.g)
```

here `x0` is the initial value, `g` is the function being minimized, `gr.g` its gradient and `hess.g` its Hessian. (Unconstrained and box-constrained optimization using PORT routines).
In the above function, the derivatives g' and g'' have to be analytically calculated. One can also use

```
z = nlminb(x0, g, gr.g)
```

without inputing the analytic expression of g'', or, even simpler,

```
z = nlminb(x0, g)
```

without inputing the analytic expressions of either derivatives. In these cases, numerical approximations of derivatives will be computed during the iterations.
Other functions for optimization in R include `optim`, `optimize`, `nlm` and `constrOptim`.

For profile likelihood

$$L(\phi) = \max_{\mu} L(\phi, \mu)$$

we need to optimize $L(\phi, \mu)$ for each given ϕ.
In R, in order to get $\min_x g(x, y)$ for each fixed y, one can do the following. First, define `g(x,y)`, `gr.g(x,y)`, etc. Then call, say,

```
nlminb(x0, g, gr.g, hess.g, y=1),   or
nlminb(x0, g, gr.g, y=.3),   or
nlminb(x0, g, y=-1)
```

If one only wants minimization for $x \in [a, b]$, use,

```
nlminb(x0, ..., lower=a, upper=b)
```

Fisher scoring

This is a variant of Newton's method specific for MLE. Recall $-l''(\theta)$ is the observed Fisher information at θ. To maximize $l(\theta)$, an alternative is to replace $-l''(\theta_t)$ with $I(\theta_t)$ to get

$$\theta_{t+1} = \theta_t + \frac{l'(\theta_t)}{I(\theta_t)}.$$

To *minimize* $-l(\theta)$, one still can use

```
z = nlminb(x0, f, grf, fs)
```

where `f` is $-l(\theta)$, `grf` is $-l'(\theta)$, and `fs` is $I(\theta)$ instead of $-l''(\theta)$. Generally, use Fisher scoring in the beginning to make rapid improvements, and Newton's method for refinement near the end.

Continuing the example on $p(x \mid \theta) = p(x - \theta)$, to use Fisher scoring, we need to compute

$$I(\theta) = -\mathbb{E}_\theta[l''(\theta)].$$

We already know

$$l''(\theta) = \sum_{i=1}^{n} \frac{p''(x_i - \theta)}{p(x_i - \theta)} - \sum_{i=1}^{n} \left\{ \frac{p'(x_i - \theta)}{p(x_i - \theta)} \right\}^2.$$

Therefore

$$I(\theta) = -n\mathbb{E}_\theta \left[\frac{p''(X - \theta)}{p(X - \theta)} - \left\{ \frac{p'(X - \theta)}{p(X - \theta)} \right\}^2 \right].$$

Since under parameter θ, X has density $p(x-\theta)$, the last $\mathbb{E}_\theta[...]$ equals

$$\begin{aligned}
&\int \left[\frac{p''(x-\theta)}{p(x-\theta)} - \left\{ \frac{p'(x-\theta)}{p(x-\theta)} \right\}^2 \right] p(x-\theta)\, \mathrm{d}x \\
&= \int p''(x-\theta)\, \mathrm{d}x - \int \frac{[p'(x-\theta)]^2}{p(x-\theta)}\, \mathrm{d}x \\
&= \frac{\mathrm{d}^2}{\mathrm{d}\theta^2} \int p(x-\theta)\, \mathrm{d}x - \int \frac{\{p'(x)\}^2}{p(x)}\, \mathrm{d}x \\
&= \frac{\mathrm{d}^2}{\mathrm{d}\theta^2} 1 - \int \frac{\{p'(x)\}^2}{p(x)}\, \mathrm{d}x = -\int \frac{\{p'(x)\}^2}{p(x)}\, \mathrm{d}x.
\end{aligned}$$

Therefore,

$$I(\theta) = n \int \frac{\{p'(x)\}^2}{p(x)}\, \mathrm{d}x.$$

So, in this case, Fisher information is a constant.

Secant method

Approximating $f'(x_t)$ by

$$\frac{f(x_t) - f(x_{t-1})}{x_t - x_{t-1}},$$

the Newton's method turns into the secant method

$$x_{t+1} = x_t - \frac{f(x_t)(x_t - x_{t-1})}{f(x_t) - f(x_{t-1})}.$$

- Need to specify x_0 and x_1.

Fixed point iteration

Compute

$$x_{t+1} = x_t + \alpha f(x_t), \quad t = 0, 1, \ldots,$$

where $\alpha \neq 0$ is a tuning parameter so that

$$|1 + \alpha f'(x)| \leq \lambda < 1, \text{ all } x$$

or more generally,

$$|x - y + \alpha[f(x) - f(y)]| \leq \lambda |x - y|, \quad \text{any } x, y$$

with $0 \leq \lambda < 1$ a constant, i.e., $F(x) = x + \alpha f(x)$ is a *contraction*.
If such α exists, the speed of convergence depends on λ. The smaller λ is, the faster the convergence.

Some Details on Fixed point iteration

Definition: A fixed point of a function is a point whose evaluation by that function equals to itself, i.e., $x = G(x)$.
Fixed point iteration: the natural way of hunting for a fixed point is to use $x_{t+1} = G(x_t)$.
Definition: A function G is contractive on $[a, b]$ if

(1). $G(x) \in [a, b]$ whenever $x \in [a, b]$,
(2). $|G(x_1) - G(x_2)| \leq \lambda |x_1 - x_2|$ for all $x_1, x_2 \in [a, b]$ and some $\lambda \in [0, 1)$.

Theorem: if G is contractive on $[a, b]$, then there is a unique fixed point $x^* \in [a, b]$, and the fixed point iteration convergence to it when starting inside the interval.
Convergence:
$|x_{t+1} - x_t| = |G(x_t) - G(x_{t-1})| \leq \lambda |x_t - x_{t-1}| \leq \lambda^t |x_1 - x_0| \to 0$, as $t \to \infty$. It follows that $\{x_t\}$ convergent to a limit x^*.

Connection between fixed-point method and Newton methods

Root-finding problems using fixed point iteration: for solving $f(x) = 0$, we can simply let $G(x) = x + \alpha f(x)$, where $\alpha \neq 0$ is a constant.
Required Lipschitz condition: $|x - y + \alpha[f(x) - f(y)]| \leq \lambda |x - y|$, for some $\lambda \in [0, 1)$ and for all $x, y \in [a, b]$. This holds if $|G'(x)| \leq \lambda$ for some $\lambda \in [0, 1)$ and for all $x \in [a, b]$, i.e., $|1 + \alpha f'(x)| \leq \lambda$. (use mean value theorem.)
Newton methods: $G(x) = x - f(x)/f'(x)$. So it is as if α_t is chosen adaptively as $\alpha_t = -1/f'(x_t)$. This leads to a faster convergence order (quadratic).

Convergence Order

Define $\varepsilon_t = x_t - x^*$. A method has convergence order β if

$$\lim_{t\to\infty} \varepsilon_t = 0, \qquad \lim_{t\to\infty} \frac{|\varepsilon_{t+1}|}{|\varepsilon_t|^\beta} = c,$$

for some constant $c \neq 0$ and $\beta > 0$.

- For Newton's method, $\beta = 2$. (If it converges.)
- For Secant method, $\beta \approx 1.62$.
- For fixed point iteration, $\beta = 1$.

Convergence Order

For Newton's method: suppose f has two continuous derivatives and $f'(x^*) \neq 0$. There then exists a neighborhood of x^* within which $f'(x) \neq 0$ for all x. By Taylor expansion,

$$0 = f(x^*) = f(x_t) + f'(x_t)(x^* - x_t) + \frac{1}{2}f''(q)(x^* - x_t)^2,$$

for some q between x^* and x_t. Rearranging terms, we find that

$$\frac{\varepsilon_{t+1}}{\varepsilon_t^2} = \frac{f''(q)}{2f'(x_t)} \to \frac{f''(x^*)}{2f'(x^*)}.$$

Convergence Order

For fixed point iteration: let G be a continuous function on the closed interval $[a, b]$ with $G : [a, b] \to [a, b]$ and suppose that G' is continuous on the open interval (a, b) with $|G'(x)| \leq k < 1$ for all $x \in (a, b)$. If $g'(x^*) \neq 0$, then for any $x_0 \in [a, b]$, the fixed point iteration converges linearly to the fixed point x^*. This is because

$$|x_{t+1} - x^*| = |G(x_t) - G(x^*)| = |G'(q)||x_t - x^*|,$$

for some q between x_t and x^*. This implies that

$$\frac{|\varepsilon_{t+1}|}{|\varepsilon_t|} \to |G'(x^*)|.$$

Stopping Rules

- Absolute convergence criterion:

$$\|x_{t+1} - x_t\| < \epsilon,$$

  where ϵ is a chosen tolerance.

- Relative convergence criterion

$$\frac{\|x_{t+1} - x_t\|}{\|x_t\|} < \epsilon.$$

  If x_t close to zero,

$$\frac{\|x_{t+1} - x_t\|}{\|x_t\| + \epsilon} < \epsilon.$$

- Many other rules depending on the algorithm.

Multivariate Problems

Newton's method

Let g now be a function in $\boldsymbol{x} = (x_1, \ldots, x_p)^T$.
The generalization is straightforward: to maximize $g(\boldsymbol{x})$, set

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - [g''(\boldsymbol{x}_t)]^{-1} g'(\boldsymbol{x}_t).$$

- $g''(\boldsymbol{x})$ is a $p \times p$ matrix with the (i, j)th entry equal to

$$\frac{\partial^2 g(\boldsymbol{x})}{\partial x_i \partial x_j};$$

- $g'(\boldsymbol{x})$ is a $p \times 1$ vector with the ith entry equal to

$$\frac{\partial g(\boldsymbol{x})}{\partial x_i}.$$

Newton-like methods

For MLE, the generalization is straightforward. Simply use

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + I(\boldsymbol{\theta}_t)^{-1} l'(\boldsymbol{\theta}_t).$$

These methods in each iteration approximate $g''(\boldsymbol{x}_t)$ by some $p \times p$ matrix $\boldsymbol{M}_t = \boldsymbol{M}_t(\boldsymbol{x}_t)$ to get

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \boldsymbol{M}_t^{-1} g'(\boldsymbol{x}_t). \tag{1}$$

Of course, $\boldsymbol{M}_t$ should be easier to compute than $g''(\boldsymbol{x}_t)$.
Fisher scoring is a Newton-like method, because it uses

$$\boldsymbol{M}_t = -I(\boldsymbol{\theta}_t)$$

in place of $-l''(\boldsymbol{\theta}_t)$.

Steepest ascent methods

In (1), set

$$\boldsymbol{M}_t = -\alpha_t^{-1}\boldsymbol{I}_p,$$

so that

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \alpha_t g'(\boldsymbol{x}_t),$$

where $\alpha_t > 0$ is the step size at t which can shrink to ensure ascent. If at step t, the original step turns out to be downhill, the updating can backtrack by halving α_t.

Discrete Newton and Fixed-point methods

In fixed-point methods, usually $\boldsymbol{M}_t$ is fixed to be $\boldsymbol{M}$, e.g., $g''(\boldsymbol{x}_0)$. This amounts to applying univariate scaled fixed-point iteration to each component.

In discrete Newton methods, $g''(\boldsymbol{x}_t)$ is approximated as follows. Compute matrix $\boldsymbol{M}_t$ with the $(i,j)^{\text{th}}$ entry equal to

$$\boldsymbol{M}_t(i,j) = \frac{g_i'(\boldsymbol{x}_t + h_t(i,j)\boldsymbol{e}_j) - g_i'(\boldsymbol{x}_t)}{h_t(i,j)}$$

for some constant $h_t(i,j) \neq 0$, where

$$g_i'(\boldsymbol{x}) = \frac{\partial g(\boldsymbol{x})}{\partial x_i}.$$

If $h_t(i,j)$ is small, then

$$\boldsymbol{M}_t(i,j) \approx \frac{\partial^2 g(\boldsymbol{x}_t)}{\partial x_i \partial x_j}$$

and so $\boldsymbol{M}_t$ approximates $g''(\boldsymbol{x}_t)$.

However, since $g''(\boldsymbol{x}_t)$ is symmetric, instead of using $\boldsymbol{M}_t$ directly, use the symmetric

$$(\boldsymbol{M}_t + \boldsymbol{M}_t^T)/2$$

as the approximation of $g''(\boldsymbol{x}_t)$.

- No need to calculate second order derivatives
- Inefficient: all p^2 entries have to be updated each time.

Quasi-Newton methods

These methods aim to achieve the following goals.

- Each step satisfies the secant condition

$$g'(\boldsymbol{x}_{t+1}) - g'(\boldsymbol{x}_t) = \boldsymbol{M}_{t+1}(\boldsymbol{x}_{t+1} - \boldsymbol{x}_t).$$

- No need to compute second order derivatives.
- Maintain symmetry of $\boldsymbol{M}_t$.
- Aim to update $\boldsymbol{M}_t$ efficiently.

There is a unique rank-one method that satisfies the secant condition and maintains the symmetry of $\boldsymbol{M}_t$: after getting

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \boldsymbol{M}_t^{-1} g'(\boldsymbol{x}_t)$$

compute

$$\boldsymbol{z}_t = \boldsymbol{x}_{t+1} - \boldsymbol{x}_t, \quad \boldsymbol{y}_t = g'(\boldsymbol{x}_{t+1}) - g'(\boldsymbol{x}_t),$$
$$\boldsymbol{v}_t = \boldsymbol{y}_t - \boldsymbol{M}_t \boldsymbol{z}_t, \quad c_t = \frac{1}{\boldsymbol{v}_t^T \boldsymbol{z}_t}.$$

Then update

$$\boldsymbol{M}_{t+1} = \boldsymbol{M}_t + c_t \boldsymbol{v}_t \boldsymbol{v}_t^T.$$

Note $\boldsymbol{M}_{t+1} - \boldsymbol{M}_t$ is of rank one. We can verify the secant condition is satisfied by multiplying both sides by $\boldsymbol{z}_t$.

There are several rank-two method satisfying the secant condition and the symmetry requirement. The Broyden class updates $\boldsymbol{M}_t$ as follows. After getting $\boldsymbol{x}_{t+1}$ and $\boldsymbol{z}_t$, $\boldsymbol{y}_t$ as above, compute

$$\boldsymbol{d}_t = \frac{\boldsymbol{y}_t}{\boldsymbol{z}_t^T \boldsymbol{y}_t} - \frac{\boldsymbol{M}_t \boldsymbol{z}_t}{\boldsymbol{z}_t^T \boldsymbol{M}_t \boldsymbol{z}_t}.$$

Then update

$$\boldsymbol{M}_{t+1} = \boldsymbol{M}_t - \frac{\boldsymbol{M}_t \boldsymbol{z}_t (\boldsymbol{M}_t \boldsymbol{z}_t)^T}{\boldsymbol{z}_t^T \boldsymbol{M}_t \boldsymbol{z}_t} + \frac{\boldsymbol{y}_t \boldsymbol{y}_t^T}{\boldsymbol{z}_t^T \boldsymbol{y}_t} + \delta_t (\boldsymbol{z}_t^T \boldsymbol{M}_t \boldsymbol{z}_t) \boldsymbol{d}_t \boldsymbol{d}_t^T,$$

where δ_t is a parameter.
A popular method to solve unconstrained nonlinear optimization problems is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method, with $\delta_t \equiv 0$.
In R, `optim` can be called to minimize a function, using the BFGS method or its variant, the "L-BFGS-B" method. "L" stands for limited memory, and "B" stands for box constraint.

Approximating Hessian for MLE

For MLE, the Hessian $l''(\widehat{\boldsymbol{\theta}}_{\mathrm{MLE}})$ is critical because it provides estimates of standard error and covariance.

- Quasi-Newton methods may provide poor approximation because it is based on the idea of using poor approximation of the Hessian to find the root for $l'(\boldsymbol{\theta}) = 0$.
- Use the discrete multivariate Newton method with

$$\boldsymbol{M}_t(i,j) = \frac{l_i'(\boldsymbol{\theta}_t + h_{ij}\boldsymbol{e}_j) - l_i'(\boldsymbol{\theta}_t - h_{ij}\boldsymbol{e}_j)}{2h_{ij}}.$$

Gauss-Newton method

Example: nonlinear regression. In many cases, we want to maximize

$$g(\boldsymbol{\theta}) = -\sum_{i=1}^{n}(y_i - f_i(\boldsymbol{\theta}))^2,$$

where each $f_i(\boldsymbol{\theta})$ is differentiable. Recall that for linear regression:

$$y_i = \boldsymbol{x}_i^T\boldsymbol{\theta} + \varepsilon, \quad i = 1, \ldots, n$$

the LS estimator of $\boldsymbol{\theta}$ maximizes $g(\boldsymbol{\theta})$ with $f_i(\boldsymbol{\theta}) = \boldsymbol{x}_i^T\boldsymbol{\theta}$ and equals

$$\widehat{\boldsymbol{\theta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

where

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_n^T \end{pmatrix}, \quad \boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}.$$

The Gauss-Newton method applies a similar idea to the nonlinear case. Let $\boldsymbol{\theta}^*$ be the (unknown) maximizer of $g(\boldsymbol{\theta})$. Given any candidate $\boldsymbol{\theta}$, the function

$$h(\boldsymbol{u}) = -\sum_{i=1}^{n}(y_i - f_i(\boldsymbol{\theta} + \boldsymbol{u}))^2.$$

is maximized by $\boldsymbol{\beta} = \boldsymbol{\theta}^* - \boldsymbol{\theta}$. Of course, $\boldsymbol{\beta}$ is unknown. However, if $\boldsymbol{\theta}$ is near $\boldsymbol{\theta}^*$, then $\boldsymbol{\beta} \approx 0$, so by Taylor expansion of $h(\boldsymbol{u})$, it may be close to the maximizer of

$$-\sum_{i=1}^{n}(y_i - f_i(\boldsymbol{\theta}) - f_i'(\boldsymbol{\theta})^T\boldsymbol{u})^2.$$

Treat $y_i - f_i(\boldsymbol{\theta})$ the same way as y_i in the linear regression, and $f_i'(\boldsymbol{\theta})$ the same way as $\boldsymbol{x}_i$, then

$$\boldsymbol{\theta}^* - \boldsymbol{\theta} \approx (\boldsymbol{A}^T \boldsymbol{A})^{-1} \boldsymbol{A}^T \boldsymbol{z}$$

where

$$\boldsymbol{z} = \boldsymbol{z}(\boldsymbol{\theta}) = \begin{pmatrix} y_1 - f_1(\boldsymbol{\theta}) \\ \vdots \\ y_n - f_n(\boldsymbol{\theta}) \end{pmatrix}, \quad \boldsymbol{A} = \boldsymbol{A}(\boldsymbol{\theta}) = \begin{pmatrix} f_1'(\boldsymbol{\theta})^T \\ \vdots \\ f_n'(\boldsymbol{\theta})^T \end{pmatrix}.$$

The update rule thus is

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + (\boldsymbol{A}_t^T \boldsymbol{A}_t)^{-1} \boldsymbol{A}_t^T \boldsymbol{z}_t,$$

where $\boldsymbol{z}_t = \boldsymbol{z}(\boldsymbol{\theta}_t)$ and $\boldsymbol{A}_t = \boldsymbol{A}(\boldsymbol{\theta}_t)$.
In R, the Gauss-Newton method is the default method of the function `nls`.

Practical Issues

- Initial value
- Convergence criteria: based on a distance measures for vectors
- Multiple local optima
  - Multiple initial values
  - Compare the objective function value at convergences
- Trade-off between efficiency and robustness

Combinatorial Optimization

Motivation:

- There are hard optimization problems for which most methods including what we have discussed so far are useless.
- These problems are usually combinatorial in nature. Maximization requires a discrete search of a very large space.
- Problem: maximize $f(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p)$, where $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ and $\boldsymbol{\Theta}$ consists of N elements.
  - Each $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ is called a candidate solution.
  - Usually N is a very large number depending on problem size p.
  - The difficulty of a particular size-p problem can be characterized by the number of operations required to solve it in the worst case scenario using the best known algorithm.
  - Suppose a problem is $\mathcal{O}(p!)$. If it requires 1 minute to solve for $p = 20$, it would take 12.1 years for $p = 25$ and 207 million years for $p = 30$.

What can we do:

- We have to take some sacrifices: we will abandon the global algorithms and focus on algorithms that can find a good local solution.
- Heuristic strategies.

What we will learn:

- Local search methods
- Simulated annealing
- Genetic algorithms

Motivating Example: Subset regression

Suppose $x_1, \ldots, x_p$ are predictors that can be used to construct a linear model for Y. Each candidate model is

$$Y = \sum_{j=1}^{s} \beta_{i_j} x_{i_j} + \varepsilon,$$

where $1 \leq i_1 < i_2 < \ldots < i_s \leq p$, $s \geq 0$. The Akaike information criterion (AIC) for the candidate model is

$$\text{AIC} = n \log \frac{\text{RSS}}{n} + 2s$$

where n is the sample size and RSS is the residual sum of squares of model. The best model is the one that minimizes AIC.
To parameterize the model, set $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p)$, such that $\theta_i = 1$ if x_i is included as a predictor, and 0 otherwise. The set $\boldsymbol{\Theta}$ of all possible $\boldsymbol{\theta}$ has 2^p values.

Local search

First, define a neighborhood $\mathcal{N}(\boldsymbol{\theta})$ for each $\boldsymbol{\theta}$, so that it

- contains candidate solutions that are "near" $\boldsymbol{\theta}$, and
- reduces the number of changes to the current $\boldsymbol{\theta}$.

For example, if $\boldsymbol{\Theta} = \{\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p) : \text{each } \theta_i = 0, 1\}$, one may define $\mathcal{N}(\boldsymbol{\theta})$ to be the set of $\boldsymbol{\theta}'$ which are different from $\boldsymbol{\theta}$ in at most one coordinate.
After neighborhoods are defined, at iteration t, choose $\boldsymbol{\theta}_{t+1}$ from $\mathcal{N}(\boldsymbol{\theta}_t)$ according to a certain rule. For example,

- steepest ascent: $\boldsymbol{\theta}_{t+1} = \arg\max_{\boldsymbol{\theta} \in \mathcal{N}(\boldsymbol{\theta}_t)} f(\boldsymbol{\theta})$;
- ascent algorithm: $\boldsymbol{\theta}_{t+1} \in \mathcal{N}(\boldsymbol{\theta}_t)$ uphill from $\boldsymbol{\theta}_t$, i.e.,

$$f(\boldsymbol{\theta}_{t+1}) \geq f(\boldsymbol{\theta}_t).$$

To avoid trapping into a local maximum, two often used variants are

- random-starts local search: repeatedly run an ascent algorithm to termination from a large number of randomly chosen starting points.
- steepest ascent/mildest descent: set to the least unfavorable $\boldsymbol{\theta}_{t+1} \in \mathcal{N}(\boldsymbol{\theta}_t)$;
  - if $\boldsymbol{\theta}_t$ is a local maximum $\boldsymbol{\theta}_{t+1}$ is the one with least decrease;
  - otherwise, $\boldsymbol{\theta}_{t+1}$ is the one with the largest increase.

To select a linear model to minimize AIC (or maximize $-$AIC),

- randomly select a set of predictors
- at iteration t, decrease the AIC by adding a predictor to the set of selected predictors or deleting a predictor that has been selected; continue the iterations until the AIC can not be decreased;
- repeat Steps 1 and 2 many times, and choose the set of predictors at termination that has the lowest AIC.

A salesman must visit each of p cities exactly once and return to his starting city, using the shortest total travel distance.
A candidate solution is

$$\boldsymbol{\theta} = (i_1, i_2, \ldots, i_p, i_1)$$

where $i_1, \ldots, i_p$ is a permutation of 1, ..., p. The objective function to be minimized is

$$f(\boldsymbol{\theta}) = d(i_1, i_2) + d(i_2, i_3) + \cdots + d(i_{p-1}, i_p) + d(i_p, i_1).$$

There are $(p-1)!/2$ all possible routes, since the point of origin and direction of travel are arbitrary. For a traveling plan to visit 20 cities, that amounts to more than 6×10^{16} possible routes.

One could define $\mathcal{N}(\boldsymbol{\theta})$ as the set of sequences that only differ from $\boldsymbol{\theta}$ at two entries. In other words, each sequence in $\mathcal{N}(\boldsymbol{\theta})$ is obtained by exchanging two entries of $\boldsymbol{\theta}$.
For example, if

$$\boldsymbol{\theta} = (1, 2, 5, 4, 6, 3, 1)$$

then

$$(1, 2, 3, 4, 6, 5, 1)$$

is a neighbor of $\boldsymbol{\theta}$, because it only has 3 and 5 exchanged; while

$$(1, 2, 4, 3, 6, 5, 1)$$

is not a neighbor of $\boldsymbol{\theta}$, because it has 5, 4, and 3 rotated.

Simulated annealing

In most cases, the simulated annealing algorithm can be thought of as a randomized local search algorithm. It uses a "temperature" parameter to control the randomness of the search. The algorithm starts with a high temperature and cools down gradually so that a global optimum may be reached. This is analogous to the annealing of metal or glass.
In the following description, a global *minimum* of f is being searched. The algorithm is run in stages, such that for the iterations within a stage, the temperature is a constant τ_j.

Suppose iteration t belongs to stage j.

1. Sample a candidate $\boldsymbol{\theta}^* \in \mathcal{N}(\boldsymbol{\theta})$ according to a *proposal density* $g_t(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t)$; for different t, the proposal density g_t can be different.
2. Let $\Delta = f(\boldsymbol{\theta}^*) - f(\boldsymbol{\theta}_t)$.
   a) If $\Delta \leq 0$, then set $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}^*$.
   b) If $\Delta > 0$, then set $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}^*$ with probability $e^{-\Delta/\tau_j}$, and $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t$ otherwise. This can be done as follows. Sample $U \sim \text{Unif}(0, 1)$. Then

$$\boldsymbol{\theta}_{t+1} = \begin{cases} \boldsymbol{\theta}^* & \text{if } \; U \leq e^{-\Delta/\tau_j} \\ \boldsymbol{\theta}_t & \text{otherwise.} \end{cases}$$

3. Repeat steps 1 and 2 a total of m_j times.
4. Update $\tau_{j+1} = \alpha(\tau_j)$, $m_{j+1} = \beta(m_j)$ and move to stage $j + 1$, where α and β are two deterministic functions that govern how to cool down the temperature and how long to stay at a given temperature.

Suppose I is an image consisting of "pixels" $I(i,j)$, $i,j = 1,\ldots,N$. The image is corrupted by noise $Z(i,j)$ and only $J = I + Z$ is observed. To reconstruct I, one way is to minimize a function

$$f(I) = \sum_{i,j} |J(i,j) - I(i,j)|^2 + K(I),$$

where $K(I)$ is a function that has large values if I has many "irregularities". The idea is that, as long as the noise is not too strong, the real image should be similar to J; on the other hand, irregularities observed in J are likely to be due to noise and should be reduced.
One way to use the simulated annealing to minimize $f(I)$ is as follows. In each iteration, only one pixel of I can be updated. That means two I and I' are neighbors only when they are different at just one pixel. Then at each iteration, choose a pixel $I(i,j)$ and select a candidate value for it. Update $I(i,j)$ according to the rule in step 2 and move to the next iteration.

Some guidelines:

- R function

  `optim`

  can implement some simple versions of simulated annealing;
- the temperature τ_j should slowly decrease to 0;
- the number m_j of iterations at each temperature τ_j should be large and increasing in j;
- reheating strategies that allow sporadic, systematic, or interactive temperature increases to prevent getting stuck in a local minimum at low temperatures can be effective.

Genetic algorithms

First, each $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ is a string of alphabets:

$$\boldsymbol{\theta} = (\theta_1, \ldots, \theta_C),$$

where each θ_i is a symbol from a finite set, such as $\{0, 1\}$, $\{0, 1, 2\}$, $\{$'a', 'b', ...$\}$.

Genetic algorithms regard the maximization of $f(\boldsymbol{\theta})$ over $\boldsymbol{\Theta}$ as a process of natural selection, with $f(\boldsymbol{\theta})$ being a measure of fitness and $\boldsymbol{\theta}$ the genetic code, or "chromosome". It assumes that fitness is a result of some "good" pieces of $\boldsymbol{\theta}$ and by inheriting these pieces plus some mutations fitness is enhanced.

In a genetic algorithm, at each iteration t, at least two candidate solutions $\boldsymbol{\theta}_{t,1}$, ..., $\boldsymbol{\theta}_{t,P}$ have to be tracked. Each iteration consists of several steps.

1. **Selection.** Randomly select from $\boldsymbol{\theta}_{t,1}, \ldots, \boldsymbol{\theta}_{t,P}$ to form a set of pairs. The selection should be based on a *fitness function* $\phi(\boldsymbol{\theta})$. In general, larger values of $f(\boldsymbol{\theta})$ result in larger values of $\phi(\boldsymbol{\theta})$, and higher chance for $\boldsymbol{\theta}$ to be selected.
   There are many selection mechanisms:
   - **a)** select one parent $\boldsymbol{\theta}$ with probability proportional to $\phi(\boldsymbol{\theta})$ and the other parent completely at random;
   - **b)** select each parent independently with probability proportional to $\phi(\boldsymbol{\theta})$;

   ϕ must be selected carefully: using $\phi(\boldsymbol{\theta}_{t,i}) = f(\boldsymbol{\theta}_{t,i})$ may result in rapid convergence into a local maximum. A common choice is

$$\phi(\boldsymbol{\theta}_{t,i}) = \frac{2r_i}{P(P+1)}$$

   where r_i is the *rank* of $f(\boldsymbol{\theta}_{t,i})$ in $f(\boldsymbol{\theta}_{t,1}), \ldots, f(\boldsymbol{\theta}_{t,P})$.

2. **Breeding.** For each pair, $(\boldsymbol{\theta}_a, \boldsymbol{\theta}_b)$, generate one or more "offspring" $\boldsymbol{\theta}' = c(\boldsymbol{\theta}_a, \boldsymbol{\theta}_b) \in \boldsymbol{\Theta}$, where c is a random operator.
Typically, c is a "crossover". If

$$\boldsymbol{\theta}_a = (\theta_{a1}, \ldots, \theta_{aC}),$$
$$\boldsymbol{\theta}_b = (\theta_{b1}, \ldots, \theta_{bC}),$$

then a crossover works as follows,

a) randomly choose a position $1 \leq d \leq C$; and
b) combine $(\theta_{a1}, \ldots, \theta_{ad})$ and $(\theta_{b,d+1}, \ldots, \theta_{bC})$ to form

$$\boldsymbol{\theta}' = (\theta_{a1}, \ldots, \theta_{ad}, \theta_{b,d+1}, \ldots, \theta_{bC}).$$

If necessary, also take

$$\boldsymbol{\theta}'' = (\theta_{a,d+1}, \ldots, \theta_{aC}, \theta_{b1}, \ldots, \theta_{bd})$$

to be another offspring.

3. **Mutation.** Make some random modifications to each offspring. Typically, if an offspring chromosome is

$$\boldsymbol{\theta} = (\theta_1, \ldots, \theta_C)$$

then for $i = 1, \ldots, C$, with a small probability $0 < p < 1$ (mutation rate), change θ_i to a different value.

The offspring produced at iteration t are taken as the candidate solutions for iteration $t + 1$.

Initialization.

Usually the first generation consists of completely random individuals.

- Large values of the size of a generation, P, are preferred. For binary encoding of $\boldsymbol{\theta}$, one suggestion is to have $C \leq P \leq 2C$, where C is the chromosome length. In most real applications, population sizes have ranged between 10 and 200.
- Mutation rates are typically very low, in the neighborhood of 1%.

Termination.

A genetic algorithm is usually terminated after a maximum number of iterations. Alternatively, the algorithm can be stopped once the genetic diversity in the current generation is sufficiently low.

In many problems, like the traveling salesman problem, it is natural to write $\boldsymbol{\theta}$ as a permutation of $1, 2, \ldots, p$.

- Standard crossover usually produces invalid sequences

For example, if

$$\boldsymbol{\theta}_a = \{7, 5, 2, 6, 3, 1, 9, 4, 8\}$$
$$\boldsymbol{\theta}_b = \{9, 1, 2, 3, 8, 6, 7, 5, 4\}$$

and if the crossover point is between 2nd and 3rd positions, then it produces

$$\{7, 5, 2, 3, 8, 6, 7, 5, 4\}$$

an invalid traveling route.

A remedy is order crossover: pick a number of positions from one parent and get the values at those positions, say $i_1, \ldots, i_s$. Next identify those values from the 2nd parent. Suppose the set of values are found at $j_1 < j_2 < \ldots < j_s$. Put i_1 at j_1, i_2 at j_2, $\ldots$, i_s at j_s while keeping the values of the 2nd on other locations unchanged.
Example: Consider parents (752631948) and (912386754) and random positions $(4, 6, 7)$. The offsprings are (612389754) and (352671948).
The drawback of the operation is that it destroys some important structures of the parent routes, in particular, the links.

Edge-recombination crossover uses a different idea. It generates an offspring whose edges belong to those of the parent routes. By edge it means a link into or out of a city in a route. For example, the above two parents have the following links, the order of numbers in each link is unimportant.

$$(1,2),(1,3),(1,9),(2,3),(2,5),(2,6),(3,6),(3,8),$$
$$(4,5),(4,8),(4,9),(5,7),(6,7),(6,8),(7,8)$$

First, choose one of the initial cities of the parents as the initial city of the offspring. At k-th step, if $i_1, \ldots, i_k$ have been chosen as the first k cities on the route, then choose among cities that are linked to i_k *and* are different from $i_1, \ldots, i_{k-1}$ as the $(k+1)$st city of the offspring.