Computational statistics

Chapter 1: Continuous Optimization

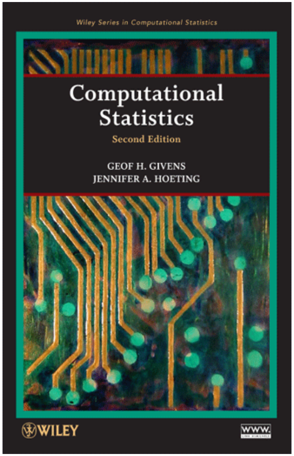
Thierry Denœux

June 2019

Computational statistics

- Modern methods in statistics and econometrics rely heavily on computational methods, for instance,
  - Nonlinear optimization
  - Monte Carlo simulation
  - Resampling techniques (bootstrap, cross-validation)
  - Nonparametric density estimation and smoothing
  - Machine Learning, data mining, big data analysis, etc.
- Computational statistics is a branch of Statistics at the intersection with Computer Science. It concerns the study of efficient procedures for solving statistical problems with computers.

Contents of this course



- Two parts:
  1. Part I: optimization
  2. Part II: simulation
- We will use the R programming language (free, flexible, large collection of available statistical methods).
- Recommended textbook: G. H. Givens and J. A. Hoeting, *Computational Statistics*, Wiley.

Part I: Optimization

- Many problems in statistics can be seen as optimizing (i.e., minimizing or maximizing) some function, for instance:
  - maximizing the likelihood
  - finding the mode of the posterior density, or highest posterior density intervals
  - minimizing risk in Bayesian decision problems
  - minimizing empirical risk (error) in machine learning problems, etc.
- For the simplest models (e.g. least-squares linear regression), a closed-form expression of the solution can be found. In most cases, we have to resort to iterative procedures.

Categories of optimization problems

- Continuous vs. combinatorial optimization
- Univariate vs. multivariate
- Constrained vs. unconstrained

Contents of this course (Part I)

1. Optimizing smooth univariate functions: Bisection, Newton's method, Fisher scoring, secant method
2. Optimizing smooth multivariate functions: nonlinear Gauss-Seidel iteration, Newton's method, Fisher scoring, Gauss-Newton method, ascent algorithms, discrete Newton method, quasi-Newton methods
3. Combinatorial optimization: local search, ascent algorithms, simulated annealing, genetic algorithms
4. Expectation-Maximization (EM) algorithm for maximizing the likelihood or posterior density

Contents of this course (Part II: simulation)

1. Simulation of probability distributions: probability integral transform, rejection sampling, sampling importance resampling
2. Markov chain Monte Carlo methods: Metropolis-Hastings algorithm, Gibbs sampling, application to Bayesian inference
3. Bootstrap

Overview

Introduction to optimization

- In this first part, the real-valued function $g : \mathbb{R}^n \rightarrow \mathbb{R}$ to be maximized or minimized will be assumed to be smooth (at least differentiable).
- It may be a likelihood, a profile likelihood, a Bayesian posterior, or any other function
- Minimizing g is equivalent to maximizing $-g$.
- Unless otherwise specified, we will consider maximization problems, without loss of generality.

Introduction to optimization (continued)

- For maximum likelihood estimation, g is the log likelihood function ℓ, and $\mathbf{x}$ is the corresponding parameter vector $\boldsymbol{\theta}$. If $\hat{\boldsymbol{\theta}}$ is a MLE, it maximizes the log likelihood. Therefore $\hat{\boldsymbol{\theta}}$ is a solution to the score equation

$$\ell'(\boldsymbol{\theta}) = \mathbf{0},$$

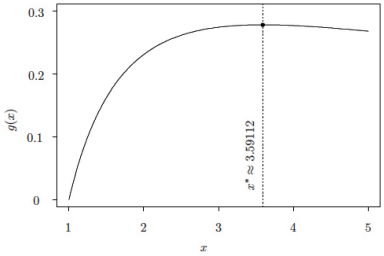
  where $\ell'(\boldsymbol{\theta}) = \left(\frac{\partial \ell(\boldsymbol{\theta})}{\partial \theta_1}, \ldots, \frac{\partial \ell(\boldsymbol{\theta})}{\partial \theta_n}\right)^T$ and $\mathbf{0}$ is a column vector of zeros.
- We see that optimization is intimately linked with solving nonlinear equations. Finding a MLE amounts to finding a root of the score equation.
- In general, the maximum of g is a solution to $g'(\mathbf{x}) = \mathbf{0}$.

Univariate Optimization for Smooth g

- Example 1: Maximize

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

  with respect to x.
- We cannot find the root of $g'(x) = \frac{1+1/x-\log x}{(1+x)^2}$ analytically.



- The maximum of $g(x) = \frac{\log(x)}{1+x}$ occurs at $x^* \approx 3.59112$, indicated by the vertical line.

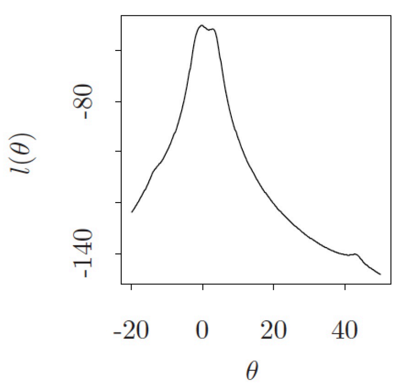
Example 2

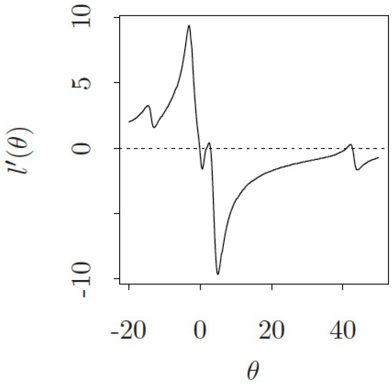
- The following data are an i.i.d. sample from a Cauchy$(\theta, 1)$ distribution:
  1.77, −0.23, 2.76, 3.80, 3.47, 56.75, −1.34, 4.24, −2.44, 3.29, 3.71, −2.40, 4.53, −0.07, −1.05, −13.87, −2.53, −1.75, 0.27, 43.21.
- The likelihood function is

$$L(\theta) = \prod_{i=1}^{20} \frac{1}{\pi\left(1 + (x_i - \theta)^2\right)}$$

  Find the MLE for θ.
- The score function $\ell'(\theta)$ has multiple roots requiring numerical solution.

Log likelihood and score function for the Cauchy data





Remark: in this example, the roots of equation $\ell'(\theta) = 0$ correspond to minima and maxima. The maxima satisfy the additional condition $\ell''(\theta) < 0$.

Local vs. global maximum

- A vector $\mathbf{x}_0$ is a local maximum of g if $\exists \epsilon > 0$ such that, for all $\mathbf{x} \in \mathbb{R}^n$,
$$\|\mathbf{x} - \mathbf{x}_0\| \leq \epsilon \Rightarrow g(\mathbf{x}_0) \geq g(\mathbf{x})$$
- A vector $\mathbf{x}_0$ is a global maximum of g if, for all $\mathbf{x} \in \mathbb{R}^n$,
$$g(\mathbf{x}_0) \geq g(\mathbf{x})$$
- We usually want to find a global maximum, but optimization algorithms can only be guaranteed to converge to a local maximum.
- Solution: restart the algorithm from different initial conditions, but we can never be sure to have reached a global maximum.

Iterative Methods

- Recall the simple example where we seek to maximize

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

  with respect to x.
- We will rely on successive approximations of the solution.
- If we know that the maximum is around 3, it might be reasonable to use $x^{(0)} = 3.0$ as an initial guess, or starting value.
- An updating equation will be used to produce an improved guess, $x^{(t+1)}$, from the most recent value $x^{(t)}$, for $t = 0, 1, 2, \ldots$ until iterations are stopped.

Overview

Overview

Bisection Method

- In the rest of this chapter we assume that $g : \mathbb{R} \to \mathbb{R}$ is a univariate function.
- If g' is continuous on $[a_0, b_0]$ and $g'(a_0)g'(b_0) \leq 0$ then the intermediate value theorem implies that there exists at least one $x^* \in [a_0, b_0]$ for which $g'(x^*) = 0$ and hence x^* is a local optimum of g.
- To find such a root, the bisection method systematically shrinks the interval from $[a_0, b_0]$ to $[a_1, b_1]$ to $[a_2, b_2]$ and so on, where $[a_0, b_0] \supset [a_1, b_1] \supset [a_2, b_2] \supset \cdots$ are nested intervals.
- If these intervals are chosen to retain $g'(a_i)g'(b_i) \leq 0$, then the ith interval contains a root.

Bisection Method

- Let $x^{(0)} = (a_0 + b_0)/2$ be the starting value.
- The updating equations are

$$[a_{t+1}, b_{t+1}] = \begin{cases} [a_t, x^{(t)}] & \text{if } g'(a_t)g'(x^{(t)}) \leq 0 \\ [x^{(t)}, b_t] & \text{if } g'(a_t)g'(x^{(t)}) > 0 \end{cases}$$

and

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

- If g has more than one root in the starting interval, it is easy to see that bisection will find one of them, but will not find the rest.

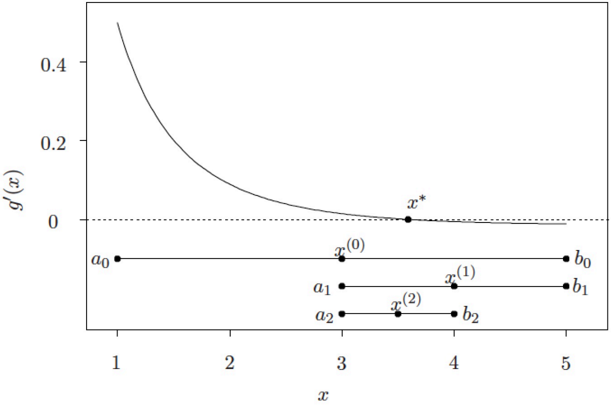
Example

- To find the value of x maximizing

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

  we might take $a_0 = 1$, $b_0 = 5$, and $x^{(0)} = 3$.
- The following figure illustrates the first few steps of the bisection algorithm.
- For continuous smooth functions, bisection is guaranteed to converge to a root because a root is always in the interval and the length of the interval halves at each iteration.
- However, the method is slow.

Example



The top portion of this graph shows $g'(x)$ and its root at x^*. The bottom portion shows the first three intervals obtained using the bisection method with $(a_0, b_0) = (1,\ 5)$. The t-th estimate of the root is at the center of the t-th interval.

Stopping Criteria

- Near the root $g'(x^{(t+1)}) \approx 0$. However, relatively large changes from $x^{(t)}$ to $x^{(t+1)}$ are often seen even when $g'(x^{(t+1)})$ is roughly zero, therefore a stopping rule based directly on $g'(x^{(t+1)})$ is not very reliable.
- On the other hand, a small change from $x^{(t)}$ to $x^{(t+1)}$ is most frequently associated with $g'(x^{(t+1)})$ near zero. Therefore, we typically assess convergence by monitoring $\left|x^{(t+1)} - x^{(t)}\right|$ and use $g'(x^{(t+1)})$ as a backup check.
- The absolute convergence criterion mandates stopping when

$$\left|x^{(t+1)} - x^{(t)}\right| < \epsilon,$$

where ϵ is a constant chosen to indicate tolerable imprecision.

Stopping Criteria (continued)

- The relative convergence criterion mandates stopping when iterations have reached a point for which
$$\frac{\left|x^{(t+1)} - x^{(t)}\right|}{\left|x^{(t)}\right|} < \epsilon. \tag{1}$$
- This criterion enables the specification of a target precision (e.g., 'within 1%') without worrying about the units of x.
- Preference between the absolute and relative convergence criteria depends on the problem at hand:
  - If the scale of x is huge (or tiny) relative to ϵ, an absolute convergence criterion may stop iterations too reluctantly (or too soon).
  - The relative convergence criterion corrects for the scale of x, but can become unstable if $x^{(t)}$ values (or the true solution) lie too close to zero.
- In this latter case, another option is to monitor relative convergence by stopping when $\frac{\left|x^{(t+1)} - x^{(t)}\right|}{\left|x^{(t)}\right| + \epsilon} < \epsilon$.

Convergence diagnostics

- Also important to include stopping rules that flag a failure to converge:
  - Stop after N iterations, regardless of convergence. Do not devote all affordable iterations to one attempt! Budget time for many smaller attempts, anticipating convergence failures, data corrections, multiple starting values, etc.
  - Could stop if any convergence measure fails to decrease or cycle over several iterations.
  - It is also sensible to stop if the procedure appears to be converging to a point at which $g(x)$ is inferior to another value you have already found (i.e., a known false peak or local maximum).
- Regardless of which such stopping rules you employ, any indication of poor convergence behavior means that $x^{(t+1)}$ must be discarded and the procedure somehow restarted in a manner more likely to yield successful convergence.

Overview

Newton's Method

- Suppose that g' is continuously differentiable and that $g''(x^*) \neq 0$.
- At iteration t, the approach approximates $g'(x^*)$ by the linear Taylor series expansion:

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

- Since g' is approximated by its tangent line at $x^{(t)}$, it seems sensible to approximate the root of g' by the root of the tangent line. Thus, solving for the root,

$$x^* \equiv x^{(t+1)} = x^{(t)} - \frac{g'(x^{(t)})}{g''(x^{(t)})} = x^{(t)} + h^{(t)}$$

- When the optimization of g corresponds to a MLE problem where $\hat{\theta}$ is a solution to $\ell'(\theta) = 0$, the updating equation for Newton's method is

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

Example

- For the simple function of Example 1,

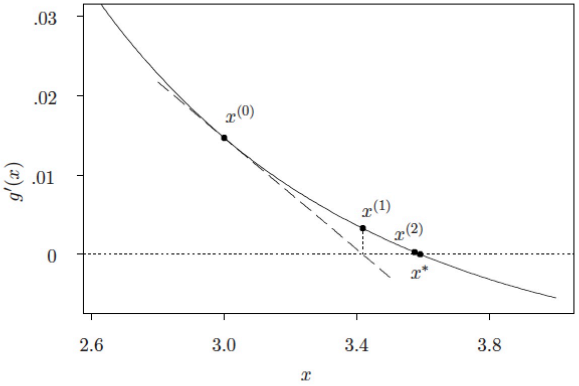
$$g(x) = \frac{\log(x)}{1+x},$$

we have

$$h^{(t)} = \frac{(x^{(t)}+1)\left[1+1/x^{(t)} - \log(x^{(t)})\right]}{3+4/x^{(t)}+1/(x^{(t)})^2 - 2\log(x^{(t)})}.$$

- The following figure illustrates the first several iterations. Starting from $x^{(0)} = 3.0$, Newton's method quickly finds $x^{(4)} \approx 3.59112$. For comparison, the first five decimal places of x^* are not correctly determined by the bisection method until iteration 19.

Example (continued)



At the first step, Newton's method approximates g' by its tangent line at $x^{(0)}$ whose root, $x^{(1)}$, serves as the next approximation of the true root, x^*. The next step similarly yields $x^{(2)}$, which is already quite close to the root at x^*.

Convergence rate

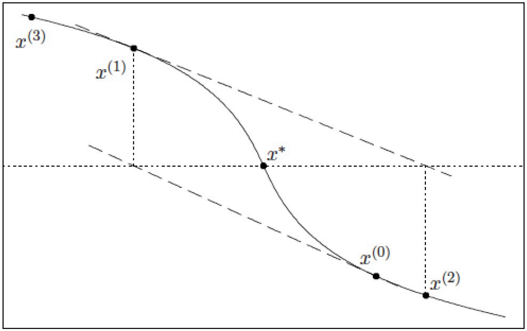
- Define the approximation error at iteration t, $\epsilon^{(t)} = x^{(t)} - x^*$
- A method has convergence of order β if $\lim_{t\to\infty} \epsilon^{(t)} = 0$ and

$$\lim_{t\to\infty} \frac{\left|\epsilon^{(t+1)}\right|}{\left|\epsilon^{(t)}\right|^{\beta}} = c$$

  for some constants $c \neq 0$ and $\beta > 0$.
- Higher orders of convergence are better in the sense that precise approximation of the true solution is more quickly achieved.
- Newton's method has quadratic convergence order, $\beta = 2$
- Unfortunately, high orders are sometimes achieved at the expense of robustness: some slow algorithms are more foolproof than their faster counterparts.

Convergence of Newton's method

Newton's method may fail to converge. For instance



Starting from $x^{(0)}$, Newton's method diverges by taking steps that are increasingly distant from the true root, x^*. In contrast, the bisection method would converge in this case.

When does Newton's method converge?

Theorem

If g' has two continuous derivatives and $g''(x^*) \neq 0$, then there exists a neighborhood of x^* for which NM converges to x^* when started from some $x^{(0)}$ in that neighborhood.

Definition

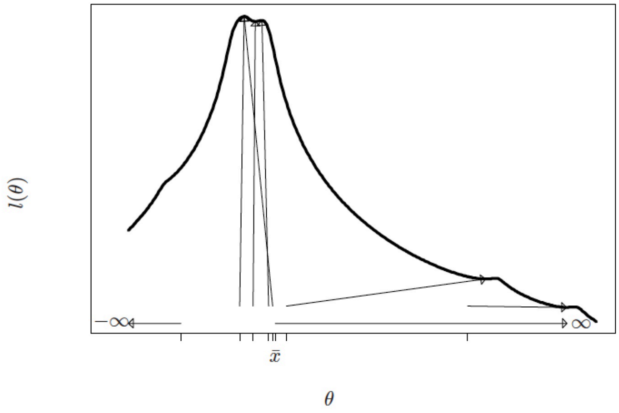
A real-valued function f defined on an interval I is convex if the line segment between any two points on the graph of the function lies above or on the graph,

$$\forall x, y \in I, \forall \alpha \in [0,1], \; f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y)$$

Theorem

If g' is twice continuously differentiable, is convex and has a root, then NM converges to that root from any starting point.

Importance of the starting point



Log-likelihood for the Cauchy data. Arrows show convergence of Newton's method from several starting values

Fisher Scoring

- Fisher information (for scalar parameter) is

$$I(\theta) = \mathbb{E}\left[\ell'(\theta)^2\right] =^* -\mathbb{E}\left[\ell''(\theta)\right]$$

  *under regularity conditions.

- Reminder: for large iid samples, it holds approximately that $\hat{\theta} \sim \mathcal{N}\left(\theta, I(\theta)^{-1}\right)$.
- Let $J(\hat{\theta}) = -\ell''(\hat{\theta})$ (observed information)
- Usually $I(\hat{\theta}) \approx J(\hat{\theta})$
- This suggests using the increment $h^{(t)} = \ell'(\theta^{(t)})/I(\theta^{(t)})$ where $I(\theta^{(t)})$ is the Fisher information evaluated at $\theta^{(t)}$.
- This yields

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

Fisher Scoring vs. Newton's method

- Fisher scoring (FS) and Newton's method (NM) share the same asymptotic properties; either may be easier for a particular problem.
- In particular, $I(\theta)$ may be easier to compute. In the case of iid data, $I_n(\theta) = nI_1(\theta)$.
- The observed information $-\ell''(\theta)$ may be negative (resulting in divergence), specially far from the solution, whereas $I(\theta)$ is always positive.
- Generally, FS makes rapid improvements initially, while NM gives better refinements near the end.
- Case of the linear canonical one-parameter exponential family:

$$f(x;\theta) = b(x) \exp\left[\theta t(x) - c(\theta)\right]$$

We have $-\ell''(\theta) = c''(\theta) = I(\theta)$: FS and NM coincide.

Overview

Secant Method

- When differentiating g' is difficult, we can replace the derivative by the discrete differenced approximation,

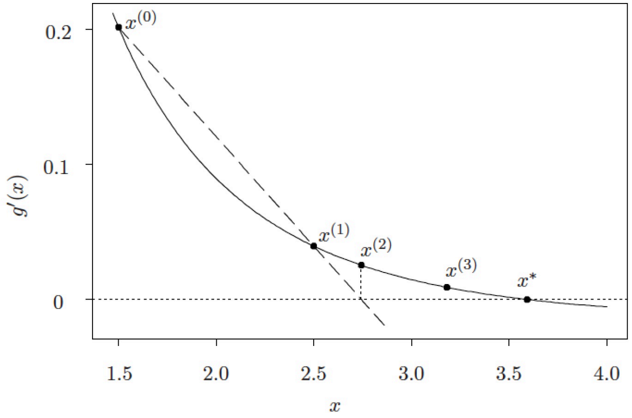
$$g''(x^{(t)}) \approx \frac{g'(x^{(t)}) - g'(x^{(t-1)})}{x^{(t)} - x^{(t-1)}}$$

- This yields the update

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

  for $t \geq 1$.
- Requires two starting points, $x^{(0)}$ and $x^{(1)}$.
- The following figure illustrates the first steps of the method for maximizing the simple function of Example 1.
- The order of convergence of the secant method is superlinear: $\beta \approx 1.62$

Example



The secant method locally approximates g' using the secant line between $x^{(0)}$ and $x^{(1)}$. The corresponding estimated root, $x^{(2)}$, is used with $x^{(1)}$ to generate the next approximation

Overview

Multivariate optimization for smooth g

- Let $g : \boldsymbol{x} \in \mathbb{R}^p \rightarrow \mathbb{R}$
- Can use analogous stopping criteria:

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

or

$$\frac{D(\mathbf{x}^{(t+1)}, \mathbf{x}^{(t)})}{D(\mathbf{x}^{(t)}, \mathbf{0}) + \epsilon} < \epsilon.$$

for $D(\mathbf{u}, \mathbf{v}) = \sum_{i=1}^p |u_i - v_i|$ or $D(\mathbf{u}, \mathbf{v}) = \sqrt{\sum_{i=1}^p (u_i - v_i)^2}$.

- Same strategy of iterative approximation. We will extend previous methods and introduce new options.

Overview

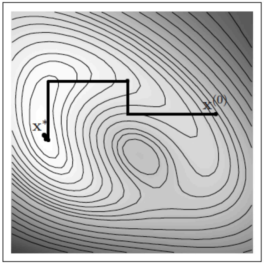
Cyclic coordinate ascent

- Also called backfitting or Gauss-Seidel iteration. One key application is for fitting additive models, GAMs, etc.
- Idea: transform a p-dimensional optimization problem into p univariate optimization problems. How?

Cyclic coordinate ascent

- Also called backfitting or Gauss-Seidel iteration. One key application is for fitting additive models, GAMs, etc.
- Idea: transform a p-dimensional optimization problem into p univariate optimization problems. How?
- Approach: optimize g with respect to each component of $\mathbf{x}$ successively, fixing all other components to their last obtained value.

Algorithm



Case $p = 2$:

- Initialize $x_1 = x_1^{(0)}$
- Find $x_2^{(1)} = \arg\max_{x_2} g(x_1^{(0)}, x_2)$
- Find $x_1^{(1)} = \arg\max_{x_1} g(x_1, x_2^{(1)})$
- Find $x_2^{(2)} = \arg\max_{x_2} g(x_1^{(1)}, x_2)$
- $\vdots$

Cyclic coordinate ascent: pros and cons

- Advantages:
  1. Simplifies a potentially difficult problem
  2. Solution of each univariate problem is easier and more stable
- Drawbacks
  1. Convergence is not guaranteed
  2. Can be slow
- For hard problems (high dimension, complex function shape), we need more sophisticated optimization procedures.

Overview

Gradient ascent

- Gradient methods are based on the gradient

$$\mathbf{g}'(\mathbf{x}) = \left(\frac{\partial g(\mathbf{x})}{\partial x_1}, \ldots, \frac{\partial g(\mathbf{x})}{\partial x_p}\right)^T,$$

  which indicates the direction of steepest ascent of function g at $\mathbf{x}$.
- The steepest ascent method uses the update equation

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

  where $\alpha^{(t)}$ is the step size at iteration t.
- How to determine the step size?

Ascent property

- For small enough $\alpha^{(t)}$, we have $g(\mathbf{x}^{(t+1)}) > g(\mathbf{x}^{(t)})$.
- Proof: we have

$$\begin{aligned} g(\mathbf{x}^{(t+1)}) - g(\mathbf{x}^{(t)}) &= g(\mathbf{x}^{(t)} + \alpha^{(t)}\mathbf{g}'(\mathbf{x}^{(t)})) - g(\mathbf{x}^{(t)}) \\ &= \alpha^{(t)}\mathbf{g}'(\mathbf{x}^{(t)})^T\mathbf{g}'(\mathbf{x}^{(t)}) + o(\alpha^{(t)}), \end{aligned}$$

  where the second equality follows from the linear Taylor expansion

$$g(\mathbf{x}^{(t)} + \alpha^{(t)}\mathbf{g}'(\mathbf{x}^{(t)})) = g(\mathbf{x}^{(t)}) + \alpha^{(t)}\mathbf{g}'(\mathbf{x}^{(t)})^T\mathbf{g}'(\mathbf{x}^{(t)}) + o(\alpha^{(t)}).$$

- Therefore, ascent can be ensured by choosing $\alpha^{(t)}$ sufficiently small, yielding

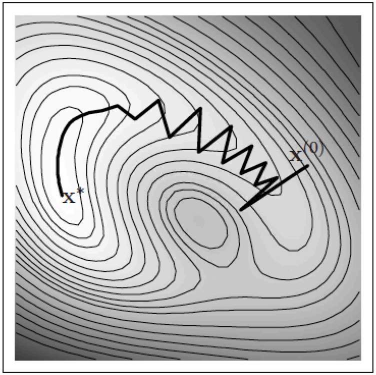
$$g(\mathbf{x}^{(t+1)}) - g(\mathbf{x}^{(t)}) > 0$$

  from (??) since $o(\alpha^{(t)})/\alpha^{(t)} \to 0$ as $\alpha^{(t)} \to 0$.

Determining $\alpha^{(t)}$

- Choosing $\alpha^{(t)}$ very small guarantees ascent, but can result in very slow convergence.
- We need a strategy to adapt $\alpha^{(t)}$, making it as large as possible, while ensuring uphill steps.
- Backtracking: attempt a step for, say, $\alpha^{(t)} = 1$;
    - If it is downhill, backtrack and reduce (e.g.,halve) $\alpha^{(t)}$.
    - If the step is still downhill, continue halving $\alpha^{(t)}$ until a sufficiently small step is found to be uphill.
- Step adaptation: attempt a step with the current value $\alpha^{(t)}$;
    - If it is downhill, backtrack and set $\alpha^{(t+1)} = b\alpha^{(t)}$ with $b < 1$.
    - If it is uphill, keep the last move and set $\alpha^{(t+1)} = a\alpha^{(t)}$ with $a > 1$

Example



Steepest ascent with backtracking, using $\alpha = 0.25$ initially at each step

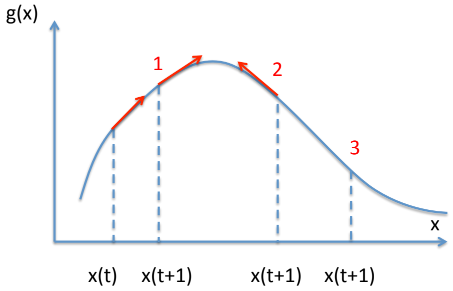
The steepest ascent direction is not necessarily the wisest, and backtracking doesn't prevent oversteps

Silva-Almeida algorithm

- Update rule:

$$x_j^{(t+1)} = x_j^{(t)} + \alpha_j^{(t)} \frac{\partial g(\mathbf{x}^{(t)})}{\partial \mathbf{x}_j}$$

- A learning rate $\alpha_j^{(t)}$ is adapted separately for each variable x_j.



1. Case 1: accept the change and set $\alpha_j^{(t+1)} = a\,\alpha_j^{(t)}$ with $a > 1$;
2. Case 2: accept the change and set $\alpha_j^{(t+1)} = b\,\alpha_j^{(t)}$ with $b < 1$;
3. Case 3: backtrack and $\alpha_j^{(t+1)} = c\,\alpha_j^{(t)}$ with $c < 1$ for all j.

- Typically, $a = 1.2$, $b = 0.8$, $c = 0.5$.

Overview

Multivariate Newton's method

- In the Silva-Almeida method, the step at each iteration is no longer in the direction of the gradient.
- Indeed, the gradient direction is not always the best. For instance, if g is quadratic,

$$g(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x} + c$$

  with A negative definite, the unique maximum can be found in one step from any starting point $\mathbf{x}^{(0)}$ by

$$x^* = -\mathbf{A}^{-1}\mathbf{b} = \mathbf{x}^{(0)} - \mathbf{g}''(\mathbf{x}^{(0)})^{-1}\mathbf{g}'(\mathbf{x}^{(0)}) \tag{2}$$

  where $\mathbf{g}''(\mathbf{x}) = \left(\frac{\partial^2 g(\mathbf{x})}{\partial x_i \partial x_j}\right)$ is the $p \times p$ Hessian matrix of g at $\mathbf{x}$
- Newton's method: at each time step, approximate $g(\mathbf{x})$ around $\mathbf{x}^{(t)}$ by a second-order Taylor series expansion, and use update equation (2).

Multivariate Newton's method and Fisher scoring

- 2nd order approximation of $g(\mathbf{x})$ around $\mathbf{x}^{(t)}$:

$$g(\mathbf{x}) \approx g(\mathbf{x}^{(t)}) + (\mathbf{x}-\mathbf{x}^{(t)})^T\mathbf{g}'(\mathbf{x}^{(t)}) + \frac{1}{2}(\mathbf{x}-\mathbf{x}^{(t)})^T\mathbf{g}''(\mathbf{x}^{(t)})(\mathbf{x}-\mathbf{x}^{(t)}).$$

- Setting $\mathbf{g}'(\mathbf{x}) = \mathbf{0}$, we get the update equation

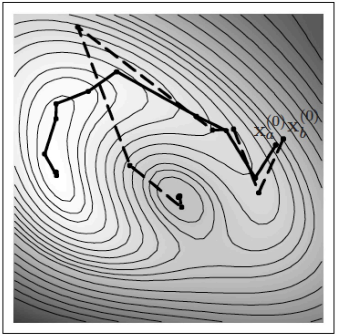
$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \mathbf{g}''(\mathbf{x}^{(t)})^{-1}\mathbf{g}'(\mathbf{x}^{(t)}).$$

- Fisher scoring:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \mathbf{I}(\boldsymbol{\theta}^{(t)})^{-1}\ell'(\boldsymbol{\theta}^{(t)}),$$

where $\mathbf{I}(\boldsymbol{\theta}) = -\mathbb{E}\{\ell''(\boldsymbol{\theta})\}$ is the Fisher information matrix at $\boldsymbol{\theta}$.

Example



Two runs starting from $\mathbf{x}_a^{(0)}$ and $\mathbf{x}_b^{(0)}$ are shown. These converge to the true maximum and to a local minimum, respectively.
Newton's method is not guaranteed to walk uphill. It is not guaranteed to find a local maximum. Step length matters even when step direction is good.

Newton-like methods

- Some very effective methods rely on updating equations of the form

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

  where $\mathbf{M}^{(t)}$ is a $p \times p$ matrix approximating the Hessian, $\mathbf{g}''(\mathbf{x}^{(t)})$.
- Two issues:
  - We want to avoid calculating Hessian if it is computationally expensive or analytically difficult
  - We want to guarantee uphill steps

Ascent algorithms

- If we use the updating increment

$$\mathbf{h}^{(t)} = -\alpha^{(t)} \left[\mathbf{M}^{(t)}\right]^{-1} \mathbf{g}'(\mathbf{x}^{(t)}).$$

  then any positive definite matrix $-\mathbf{M}^{(t)}$ will ensure ascent for a sufficiently small $\alpha^{(t)}$
- Backtracking can be used, as in the steepest ascent method.
- Steepest ascent is recovered as a special case, with $\mathbf{M}^{(t)} = -\mathbf{I}$.
- Fisher scoring is another special case with $-\mathbf{M}^{(t)} = \mathbf{I}(\boldsymbol{\theta}^{(t)})$. Since $\mathbf{I}(\boldsymbol{\theta}^{(t)})$ is positive semi-definite, backtracking with Fisher scoring avoids stepping downhill.

Discrete Newton method

- To avoid calculating the Hessian, one could resort to an analogue of the 1-dimensional secant method.
- For example,

$$\mathbf{M}_{ij}^{(t)} = \frac{g_i'\left(\mathbf{x}^{(t)} + h_{ij}^{(t)}\mathbf{e}_j\right) - g_i'\left(\mathbf{x}^{(t)}\right)}{h_{ij}^{(t)}}$$

  where $g_i'(\mathbf{x}) = dg(\mathbf{x})/dx_i$ is the ith element of $\mathbf{g}'(\mathbf{x})$, $\mathbf{e}_j$ is the p-vector with a 1 in the jth position and zeros elsewhere, and $h_{ij}^{(t)}$ are some constants.
- $h_{ij}^{(t)} = h$ for all (i,j) and t leads to linear convergence order: $\beta = 1$.
- Alternatively, $h_{ij}^{(t)} = x_j^{(t)} - x_j^{(t-1)}$ for all i gives superlinear convergence.

Quasi-Newton methods

- The discrete Newton method strategy is computationally burdensome because $\mathbf{M}^{(t)}$ is wholly updated at every step.
- A more efficient approach can be designed based on the direction of the most recent step. From a first order Taylor series expansion of $\mathbf{g}'$ at $\mathbf{x}^{(t)}$, we have

$$\mathbf{g}'(\mathbf{x}^{(t+1)}) - \mathbf{g}'(\mathbf{x}^{(t)}) \approx \mathbf{g}''(\mathbf{x}^{(t)})(\mathbf{x}^{(t+1)} - \mathbf{x}^{(t)})$$

- $\mathbf{M}^{(t+1)}$ satisfies the secant condition if

$$\mathbf{g}'(\mathbf{x}^{(t+1)}) - \mathbf{g}'(\mathbf{x}^{(t)}) = \mathbf{M}^{(t+1)}(\mathbf{x}^{(t+1)} - \mathbf{x}^{(t)}). \tag{3}$$

- Goal: generate $\mathbf{M}^{(t+1)}$ from $\mathbf{M}^{(t)}$ in a manner that requires few calculations and satisfies (3), while learning about the curvature of $\mathbf{g}'$ in the direction of the most recent step.

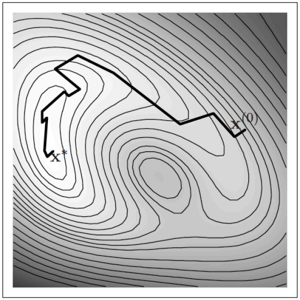
BFGS method

- The widely used BFGS method updates matrix $\mathbf{M}^{(t+1)}$ so as to satisfy the secant condition. It is defined by the following update equation

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

  where $\mathbf{z}^{(t)} = \mathbf{x}^{(t+1)} - \mathbf{x}^{(t)}$ and $\mathbf{y}^{(t)} = \mathbf{g}'(\mathbf{x}^{(t+1)}) - \mathbf{g}'(\mathbf{x}^{(t)})$.
- The BFGS update confers hereditary positive definiteness: if $-\mathbf{M}^{(t)}$ is positive definite, so is $-\mathbf{M}^{(t+1)}$.
- Backtracking is normally used.

Example



Quasi-Newton optimization with the BFGS update and backtracking to ensure ascent.

Convergence of quasi-Newton methods is generally superlinear, but not quadratic. These are powerful and popular methods, available, for example, in the R function `optim()`.

Overview

Gauss-Newton method

Basic idea

- For nonlinear least squares problems with observed data $(y_i, \mathbf{z}_i)$ for $i = 1, \ldots, n$ and model
$$Y_i = f(\mathbf{z}_i, \boldsymbol{\theta}) + \epsilon_i$$
for some non-linear function, f, and random error, ϵ_i.
- We seek to estimate $\boldsymbol{\theta}$ by maximizing an objective function
$$g(\boldsymbol{\theta}) = -\sum_{i=1}^{n} (y_i - f(\mathbf{z}_i, \boldsymbol{\theta}))^2 .$$
- Newton's method approximates g via Taylor series. The Gauss-Newton approach approximates f itself by its linear Taylor series expansion about $\boldsymbol{\theta}^{(t)}$.

Gauss-Newton method

Linearized reformulation

- We have
$$f(\mathbf{z}_i, \boldsymbol{\theta}) \approx f(\mathbf{z}_i, \boldsymbol{\theta}^{(t)}) + (\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)})^T \mathbf{f}'(\mathbf{z}_i, \boldsymbol{\theta}^{(t)})$$
where for each i, $\mathbf{f}'(\mathbf{z}_i, \boldsymbol{\theta}^{(t)})$ is the column vector of partial derivatives of f with respect to $\theta_j^{(t)}$, for $j = 1, \ldots, p$, evaluated at $(\mathbf{z}_i, \boldsymbol{\theta}^{(t)})$.
- Now, instead of $g(\boldsymbol{\theta})$, we maximize
$$\begin{aligned}\tilde{g}(\boldsymbol{\theta}) &= -\sum_{i=1}^{n} \left(y_i - f(\mathbf{z}_i, \boldsymbol{\theta}^{(t)}) - (\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)})^T \mathbf{f}'(\mathbf{z}_i, \boldsymbol{\theta}^{(t)})\right)^2 \\ &= -\sum_{i=1}^{n} \left(x_i^{(t)} - (\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)})^T \mathbf{a}_i^{(t)}\right)^2\end{aligned}$$
with respect to $\boldsymbol{\theta}$, with $x_i^{(t)} = y_i - f(\mathbf{z}_i, \boldsymbol{\theta}^{(t)})$, and define $\mathbf{a}_i^{(t)} = \mathbf{f}'(\mathbf{z}_i, \boldsymbol{\theta}^{(t)})$.

Gauss-Newton method

Update equation

- Then the approximated problem can be re-expressed as minimizing the squared residuals of the linear regression model

$$\mathbf{X}^{(t)} = \mathbf{A}^{(t)}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) + \boldsymbol{\epsilon}$$

  where $\mathbf{X}^{(t)}$ and $\boldsymbol{\epsilon}$ are column vectors whose ith elements consist of $X_i^{(t)}$ and ϵ_i, respectively. Similarly, $\mathbf{A}^{(t)}$ is a matrix whose ith row is $(\mathbf{a}_i^{(t)})^T$.

- This is a linear regression problem! Thus,

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \left((\mathbf{A}^{(t)})^T \mathbf{A}^{(t)}\right)^{-1} (\mathbf{A}^{(t)})^T \mathbf{x}^{(t)}.$$

- Requires no computation of Hessian.
- Works best when the model fits fairly well and f is not severely nonlinear.

Overview

Nelder-Mead algorithm

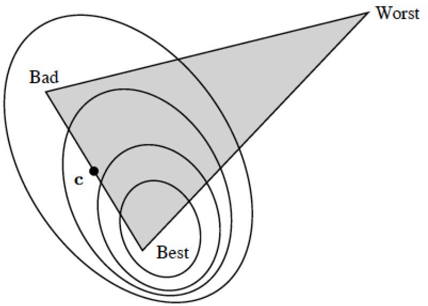
Main idea

- An algorithm that does not require the calculation of $g(\mathbf{x})$ or $g''(\mathbf{x})$.
- Idea: evaluation g at $p+1$ points $\mathbf{x}_1, \ldots, \mathbf{x}_{p+1}$ forming a simplex*.
- This simplex defines a region, which is iteratively reshaped by replacing the worst point (vertex) by a better one.

* Definition: a k-simplex is a k-dimensional polytope which is the convex hull of $k+1$ points (vertices). A a 2-simplex is a triangle.

Nelder-Mead algorithm

Definitions



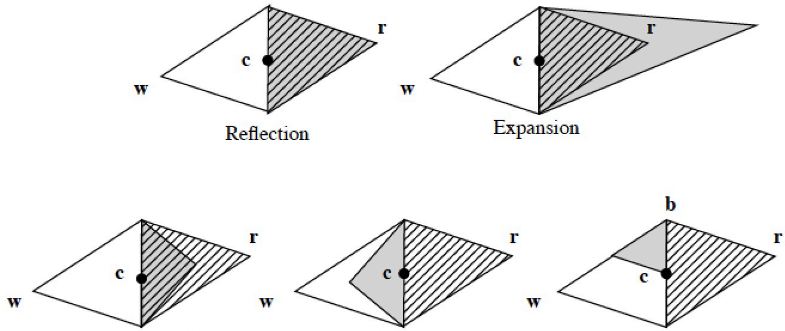
Let

- $\mathbf{x}_{best}$: vertex with highest value of g
- $\mathbf{x}_{worst}$: vertex with lowest value of g
- $\mathbf{x}_{bad}$: 2nd worst vertex
- Best face: face opposite to $\mathbf{x}_{worst}$, $\mathbf{c}$ its centroid.

Nelder-Mead algorithm

Transformations of a vertex

Five possible transformations of a vertex:



Reflection

Expansion

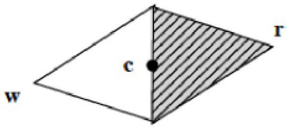
Outer Contraction

Inner Contraction

Shrinkage

Nelder-Mead algorithm

Basic algorithm

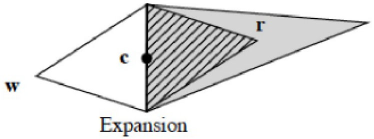


Reflection

- The location of the new vertex (replacing $\mathbf{x}_{worst}$) is based on the reflection vertex $\mathbf{x}_r = \mathbf{c} + \alpha_r(\mathbf{c} - \mathbf{x}_{worst})$, usually $\alpha_r = 1$
- If $g(\mathbf{x}_{bad}) < g(\mathbf{x}_r) < g(\mathbf{x}_{best})$: keep $\mathbf{x}_r$ as the new vertex
- If $g(\mathbf{x}_r) > g(\mathbf{x}_{best})$: try an expansion step
- If $g(\mathbf{x}_r) < g(\mathbf{x}_{bad})$: try a contraction step

Nelder-Mead algorithm

Expansion

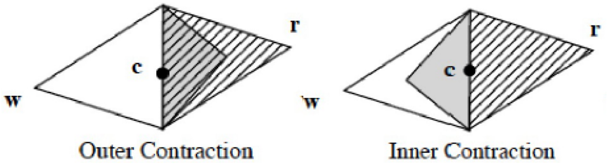


If $g(\mathbf{x}_r) > g(\mathbf{x}_{best})$: Expansion.
Let $\mathbf{x}_e = \mathbf{c} + \alpha_e(\mathbf{x}_r - \mathbf{c})$, usually $\alpha_e = 2$

- If $g(\mathbf{x}_e) > g(\mathbf{x}_r)$: set $\mathbf{x}_e$ as the new vertex
- Otherwise, keep $\mathbf{x}_r$

Nelder-Mead algorithm

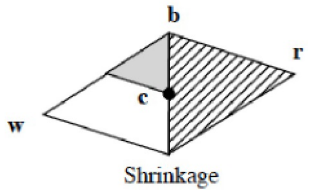
Contraction



If $g(\mathbf{x}_r) < g(\mathbf{x}_{bad})$: Contraction.

- If $g(\mathbf{x}_r) > g(\mathbf{x}_{worst})$: outer contraction. Let $\mathbf{x}_o = \mathbf{c} + \alpha_c(\mathbf{x}_r - \mathbf{c})$, usually $\alpha_c = 0.5$.
  - If $g(\mathbf{x}_o) > g(\mathbf{x}_r)$: keep $\mathbf{x}_o$
  - Otherwise: perform a shrink transformation
- If $g(\mathbf{x}_r) \leq g(\mathbf{x}_{worst})$: inner contraction. Let $\mathbf{x}_i = \mathbf{c} + \alpha_c(\mathbf{x}_{worst} - \mathbf{c})$.
  - If $g(\mathbf{x}_i) > g(\mathbf{x}_{worst})$: keep $\mathbf{x}_i$
  - Otherwise: perform a shrink transformation

Nelder-Mead algorithm
Shrinking



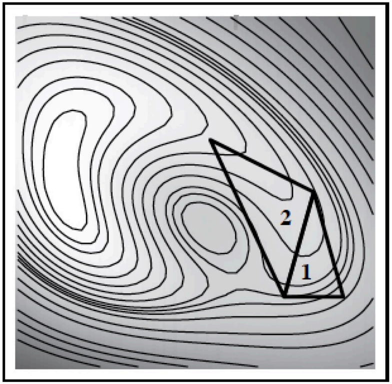
Shrink transformation: all vertices except $\mathbf{x}_{best}$ are shrunk toward $\mathbf{x}_{best}$:

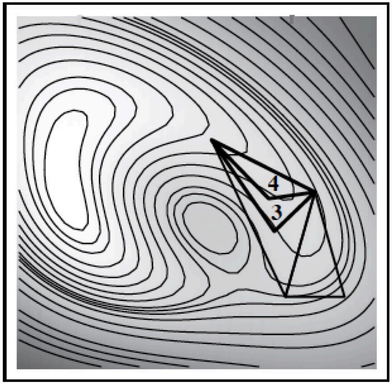
$$\mathbf{x}'_j = \mathbf{x}_{best} + \alpha_s(\mathbf{x}_j - \mathbf{x}_{best}),$$

usually $\alpha_s = 0.5$.

Nelder-Mead algorithm

Example





Nelder-Mead algorithm

Example