Optimization and the BFGS Algorithm

Jake Spiteri

December 06, 2019

Table of Contents

1. Introduction
   - Unconstrained Optimization
   - Motivating Example
   - Intuition

2. Unconstrained Optimization
   - Algorithms
     - Second-Order Methods — Newton's Method
     - Quasi-Newton Methods — The BFGS Method

The Optimization Problem

An optimization problem has the general form

$$\begin{array}{ll} \text{minimize} & f_0(\boldsymbol{x}) \\ \text{subject to} & f_i(\boldsymbol{x}) \leq b_i, \quad i = 1, \ldots, m. \end{array}$$

We say that

- $\boldsymbol{x} = (x_1, \ldots, x_n)$ are the *optimization variables*,
- $f_0 : \mathbb{R}^n \to \mathbb{R}$ is the *objective function*,
- $f_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, m$ are the *inequality constraint functions*.

The Optimization Problem

We say that a vector $\boldsymbol{x}^*$ is optimal if it minimizes the objective function $f_0(\boldsymbol{x})$ and satisfies the constraint functions f_i, $i = 1, \ldots, m$.

Unconstrained Problem

We minimize an objective function which is a function of real variables. We place no constraints on these values. The general problem can be written

$$\min\{f(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^n\}$$

A Motivating Example — MLE

Suppose we observe data $\{x_i, y_i\}_{i=1}^n$, where our y_i is count data. Then we can think of Y_i as following a Poisson distribution. I.e. we can form the Poisson regression model:

$$Y_i \sim \text{Poisson}(\mu_i), \quad \mu_i = \mathbb{E}(Y_i), \quad log(\mu_i) = \beta_0 + \beta_1 x_i$$

We want to estimate the parameters $\boldsymbol{\beta} = (\beta_0, \beta_1)$ by maximum likelihood estimation.
We can write down the conditional probability and thus the log likelihood:

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{i=1}^{n} e^{y_i \boldsymbol{\beta} \boldsymbol{x}_i} e^{-e^{\boldsymbol{\beta} \boldsymbol{x}_i}} / y_i!$$

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} y_i \boldsymbol{\beta} \boldsymbol{x}_i - e^{\boldsymbol{\beta} \boldsymbol{x}_i} - \log(y_i!)$$

We cannot write down a closed form for $\boldsymbol{\beta}$! We require numerical optimization methods.

Intuition for Optimization Algorithms

The optimization algorithms we present are iterative processes that allow us to find the minimum of a function. Often we can only promise a local minimum.

At time step k, we have current iterate $\boldsymbol{x}_k$ and we want to find the next iterate $\boldsymbol{x}_{k+1}$ such that $f(\boldsymbol{x}_k) > f(\boldsymbol{x}_{k+1})$.

Our algorithms have the general form

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{d}_k,$$

where $\boldsymbol{d}_k$ is a descent direction.

Intuition for Optimization Algorithms

First-order methods: Use the first derivative.

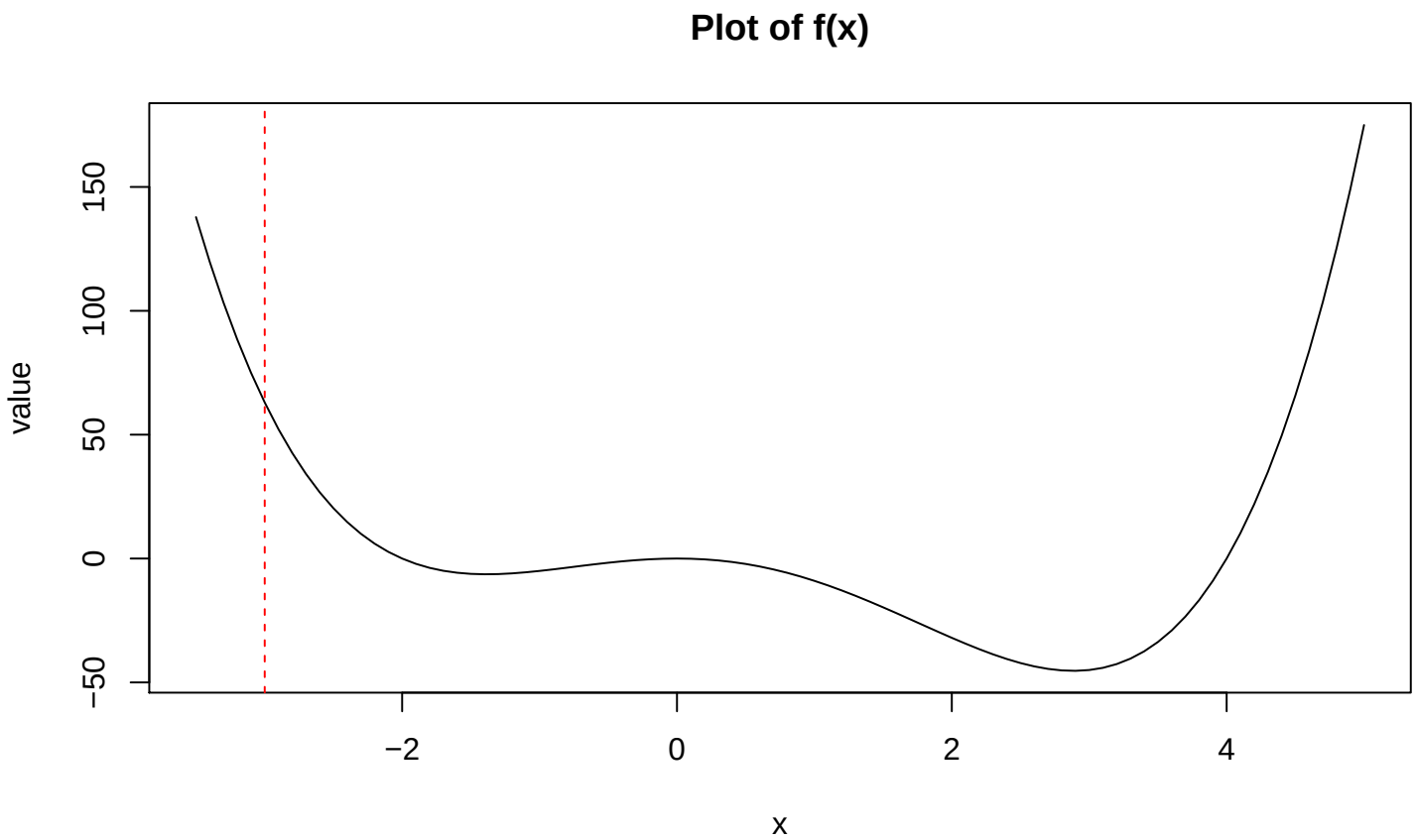
Steepest descent is a very popular first-order method with the following update step:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k).$$

Suppose we want to minimize the function $f(x) = x^4 - 2x^3 - 8x^2$.
At time step k we have $x_k = -3$. How do we choose x_{k+1}?

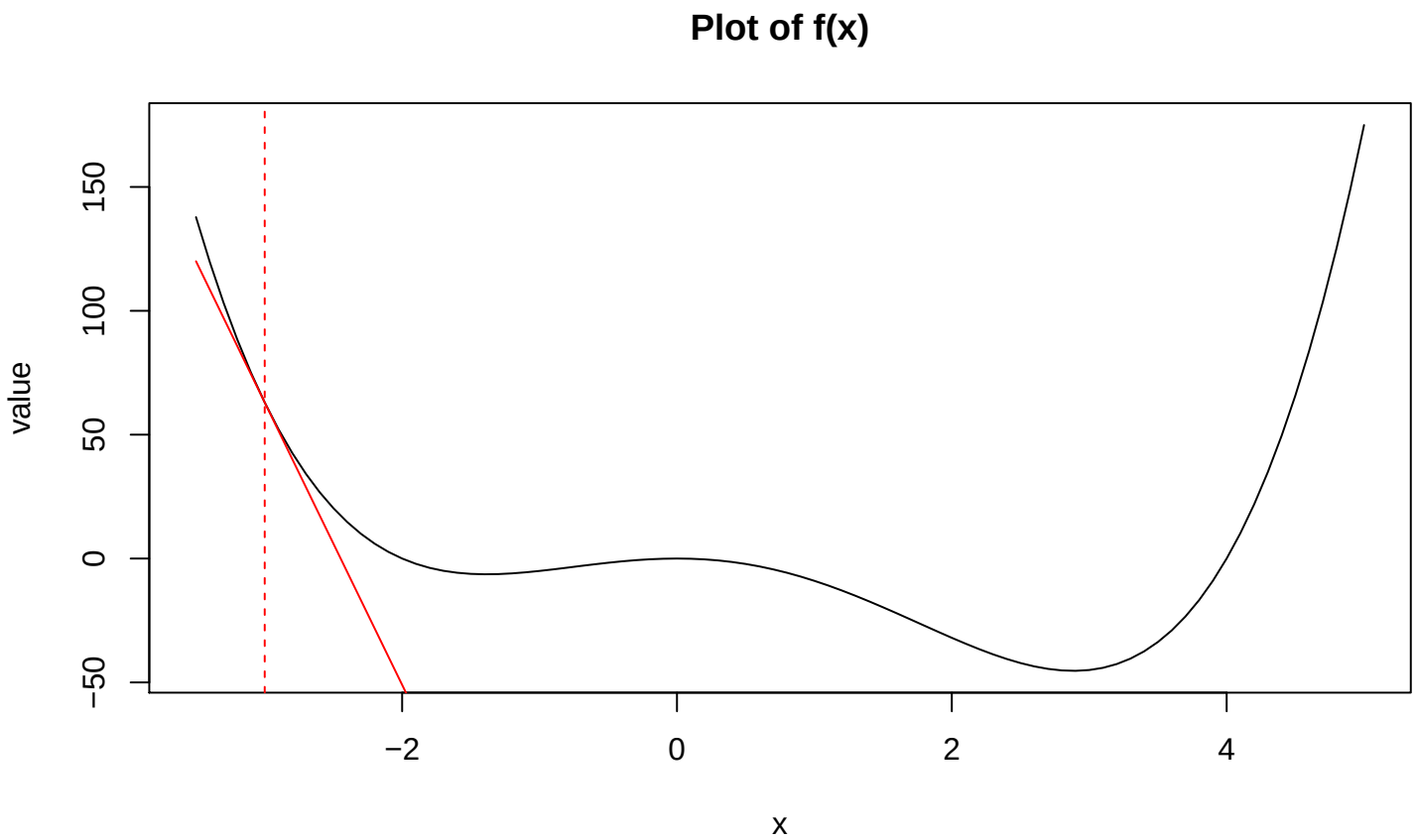
Intuition for Optimization Algorithms

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)$$



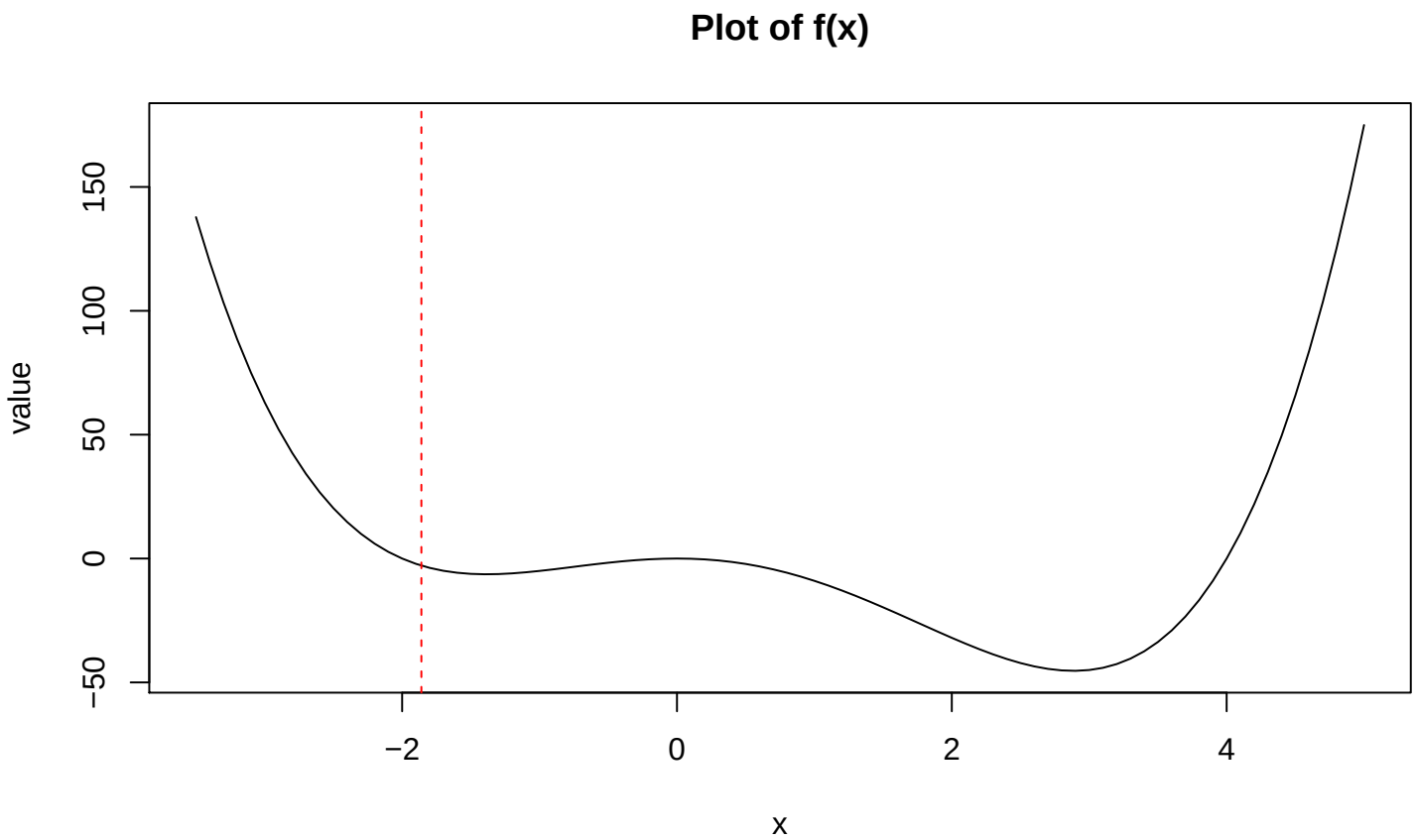
Intuition for Optimization Algorithms

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)$$



Intuition for Optimization Algorithms

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)$$



Intuition for Optimization Algorithms

Second-order methods: Use the first derivative and the Hessian (the matrix of second derivatives).

Newton's method is a very popular second-order method with the following update step:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - (\nabla^2 f(\boldsymbol{x}_k))^{-1} \nabla f(\boldsymbol{x}_k).$$

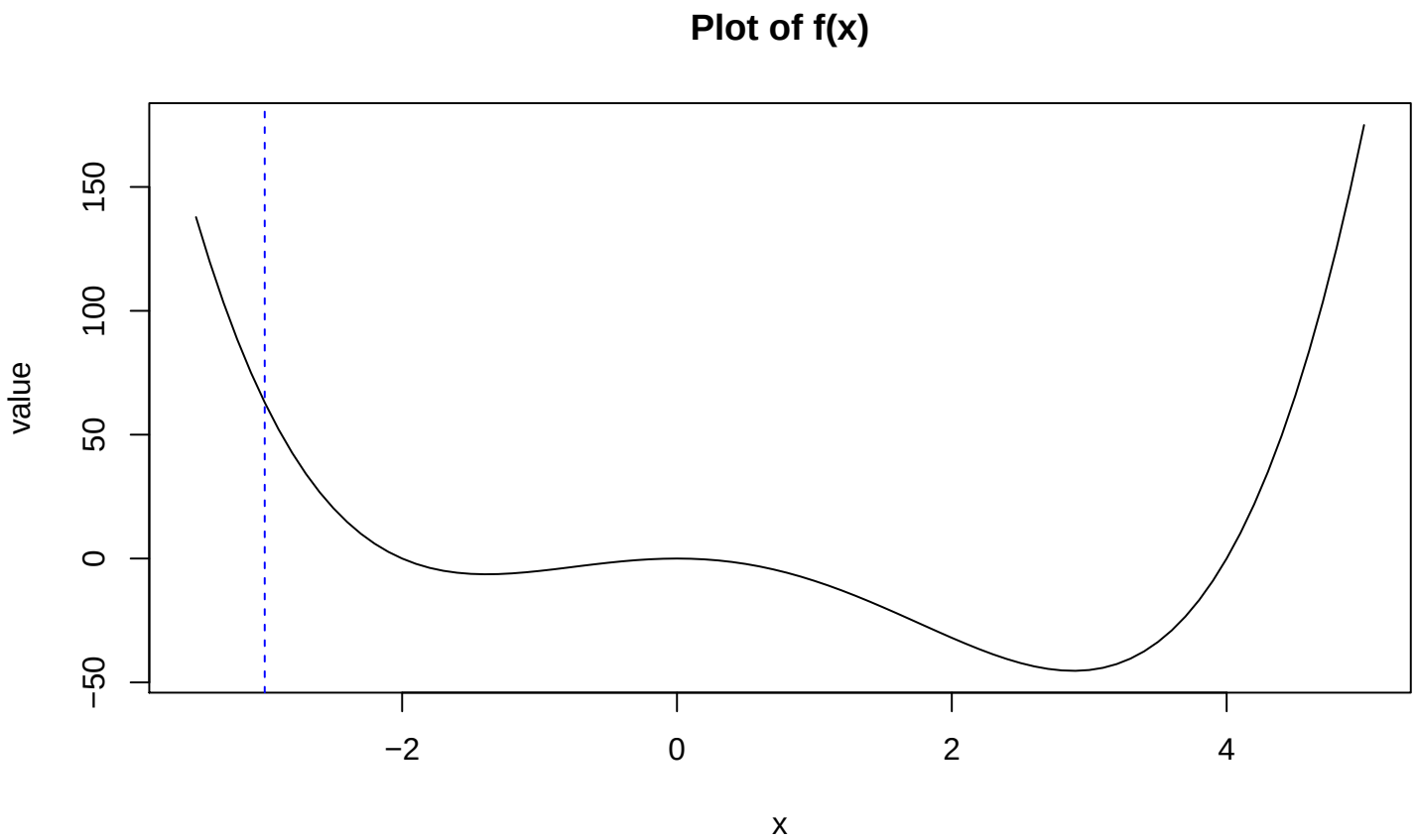
Key intuition

Newton's method minimizes the second-order approximation of the function.

Suppose we want to minimize the function $f(x) = x^4 - 2x^3 - 8x^2$.
At time step k we have $x_k = -3$. How do we choose x_{k+1}?

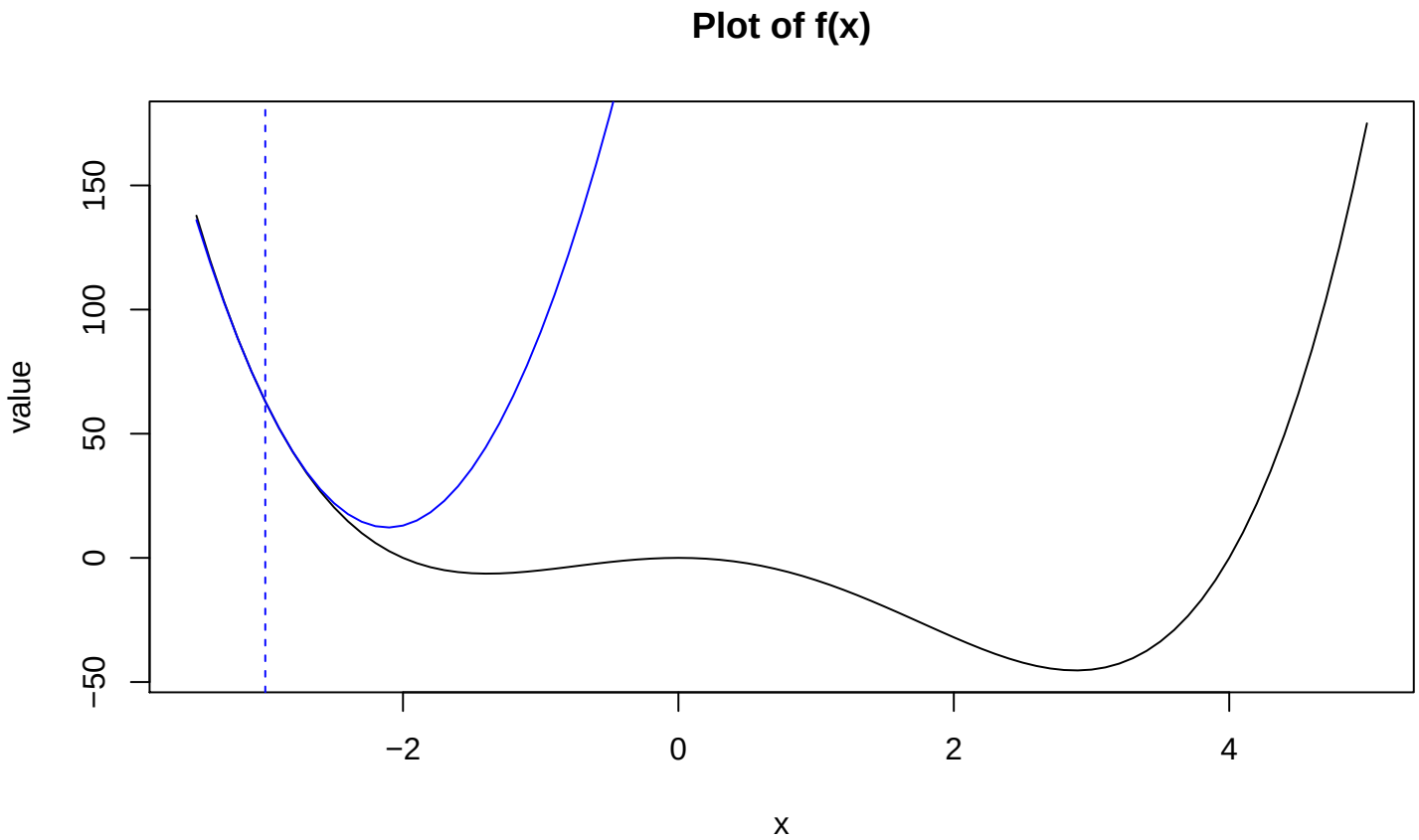
Intuition for Optimization Algorithms

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - (\nabla^2 f(\boldsymbol{x}_k))^{-1} \nabla f(\boldsymbol{x}_k)$$



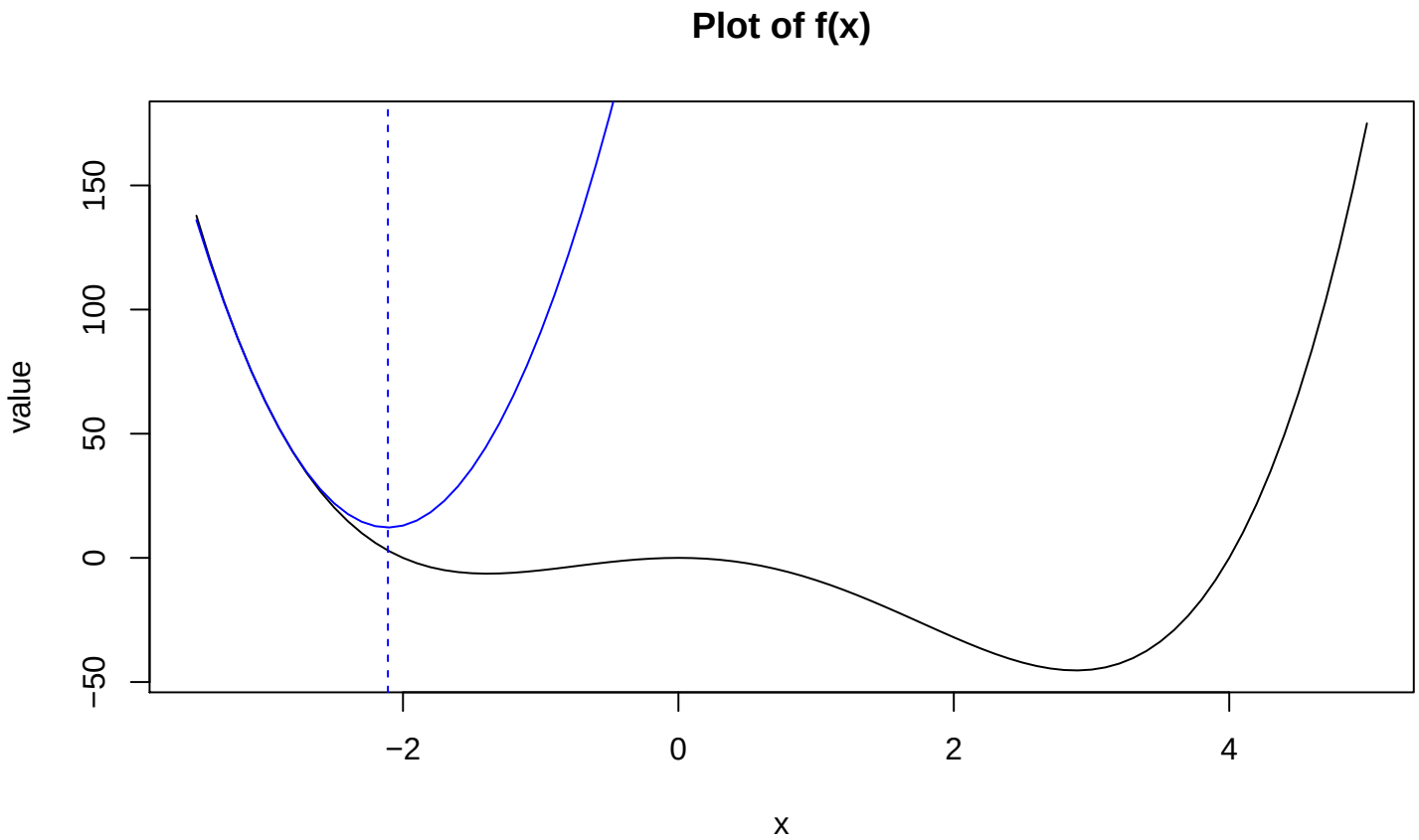
Intuition for Optimization Algorithms

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - (\nabla^2 f(\boldsymbol{x}_k))^{-1} \nabla f(\boldsymbol{x}_k)$$



Intuition for Optimization Algorithms

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - (\nabla^2 f(\boldsymbol{x}_k))^{-1} \nabla f(\boldsymbol{x}_k)$$



Second-Order Methods — Approximating a function

Theorem (quadratic approximation theorem)

Let $f : U \to \mathbb{R}$ be a twice continuously differentiable function over an open set $U \subset \mathbb{R}^n$, and let $\boldsymbol{x} \in U, r > 0$ satisfy $B(\boldsymbol{x}, r) \subseteq U$. Then for any $\boldsymbol{y} \in B(\boldsymbol{x}, r)$

$$f(\boldsymbol{y}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T(\boldsymbol{y} - \boldsymbol{x}) + \frac{1}{2}(\boldsymbol{y} - \boldsymbol{x})^T \nabla^2 f(\boldsymbol{x})(\boldsymbol{y} - \boldsymbol{x}) + o(\|\boldsymbol{y} - \boldsymbol{x}\|^2).$$

Second-Order Methods

- A second-order method uses evaluations of the Hessian.
- Newton's method is an iterative method in which each update is chosen to minimize the quadratic approximation of the objective function around $\boldsymbol{x}_k$.

Quadratic approximation of $f(\boldsymbol{x})$ around $\boldsymbol{x}_k$:

$$Q(\boldsymbol{x}) = f(\boldsymbol{x}_k)+\nabla f(\boldsymbol{x}_k)^T(\boldsymbol{x}-\boldsymbol{x}_k)+\frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}_k)^T\nabla^2 f(\boldsymbol{x}_k)(\boldsymbol{x}-\boldsymbol{x}_k)\ \boldsymbol{x}_{k+1} = \underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{argmin}}$$

$$\nabla f(\boldsymbol{x}_k) + \nabla^2 f(\boldsymbol{x}_k)(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) = \boldsymbol{0},$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - (\nabla^2 f(\boldsymbol{x}_k))^{-1}\nabla f(\boldsymbol{x}_k)$$

Comments on Newton's Method

Newton's method converges quadratically.

Pure Newton's method may diverge if our initialization is bad. We can avoid this by modifying the Hessian at each step to make it positive (semi) definite if it is not, and ensuring that the step size is chosen such that the function decreases.

It may be difficult to evaluate the Hessian, particularly for high-dimensional objective functions.

Quasi-Newton Methods

Quasi-Newton methods use an approximation of the second derivative.

Avoids evaluating the Hessian $\nabla^2 f(\boldsymbol{x})$ at each iteration — an error-prone and expensive process.

Attains a **superlinear** rate of convergence, and lower computational cost ($\mathcal{O}(n^2)$ as opposed to $\mathcal{O}(n^3)$ in Newton's method).

The BFGS Method

Recall the second-order Taylor series approximation around $\boldsymbol{x}_k$,

$$f(\boldsymbol{x}_k + \boldsymbol{d}) = f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T \nabla^2 f(\boldsymbol{x}_k)\boldsymbol{d} + o(\|\boldsymbol{d}\|^2)$$

We will define the approximation of the objective function at $\boldsymbol{x}_k$ as

$$m_k(\boldsymbol{d}) = f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T \boldsymbol{B}_k \boldsymbol{d}$$

$\boldsymbol{B}_k$ is an $n \times n$ symmetric positive definite matrix that will be updated at every iteration.

The BFGS Method

The minimizer of the convex quadratic $m_k(\boldsymbol{d})$, which we can write as

$$\boldsymbol{d} = \boldsymbol{B}_k^{-1} \nabla f(\boldsymbol{x}_k),$$

is the *search direction*. Hence, the new iterate is

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{B}_k^{-1} \nabla f(\boldsymbol{x}_k),$$

where α_k is our step size.

This iterative update is the same as Newton's method except it uses the approximate Hessian.

Instead of recomputing $\boldsymbol{B}_k$ at every iteration, we iteratively update it by accounting for the curvature measured in the most recent step.

Suppose we generate a new iterate $\boldsymbol{x}_{k+1}$, then we construct the new quadratic model

$$m_{k+1}(\boldsymbol{d}) = f(\boldsymbol{x}_{k+1}) + \nabla f(\boldsymbol{x}_{k+1})^T \boldsymbol{d} + \boldsymbol{d}^T \boldsymbol{B}_{k+1} \boldsymbol{d}$$

In order to determine the new $\boldsymbol{B}_{k+1}$ we place restrictions on $m_{k+1}(\boldsymbol{d})$.

We require that the gradient of m_{k+1} be equal to the gradient of f evaluated at the two most recent iterates $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k+1}$.

1. $\nabla m_{k+1}(\boldsymbol{0}) = \nabla f(\boldsymbol{x}_{k+1})$,
2. $\nabla m_{k+1}(-\alpha_k \boldsymbol{d}_k) = \nabla f(\boldsymbol{x}_{k+1)} - \alpha_k \boldsymbol{B}_{k+1}\boldsymbol{d}_k = \nabla f(\boldsymbol{x}_k)$.

Rearranging 2 gives us

$$\alpha_k \boldsymbol{B}_{k+1}\boldsymbol{d}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$$

We introduce the following notation:

- $\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k \quad (= \alpha_k \boldsymbol{d}_k)$,
- $\boldsymbol{y}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$,

Recall from the last slide

$$\alpha_k \boldsymbol{B}_{k+1} \boldsymbol{d}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k),$$

then we can write the secant equation

$$\boldsymbol{B}_{k+1} \boldsymbol{s}_k = \boldsymbol{y}_k.$$

- Secant equation:

$$\boldsymbol{B}_{k+1}\boldsymbol{s}_k = \boldsymbol{y}_k$$

- BFGS update:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{B}_k^{-1} \nabla f(\boldsymbol{x}_k)$$

- BFGS places restrictions on the inverse of the Hessian approximation

$$\boldsymbol{H}_k = \boldsymbol{B}_k^{-1}$$

- Secant equation

$$\boldsymbol{H}_{k+1}\boldsymbol{y}_k = \boldsymbol{s}_k$$

Secant equation:

$$\boldsymbol{H}_{k+1}\boldsymbol{y}_k = \boldsymbol{s}_k$$

We require that the symmetric positive definite matrix $\boldsymbol{H}_{k+1}$ maps $\boldsymbol{y}_k$ to $\boldsymbol{s}_k$. This is only possible if $\boldsymbol{y}_k$ and $\boldsymbol{s}_k$ satisfy the curvature condition

$$\boldsymbol{y}_k^T \boldsymbol{s}_k > 0.$$

If f is strongly convex then the curvature condition is satisfied for any $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k+1}$.

Remark

For non-convex f we must impose this restriction explicitly. We impose the Wolfe or strong Wolfe conditions on the line search. The Wolfe conditions ensure that we take a step which produces a sufficient decrease in f, and doesn't take an unacceptably small step.

The secant equation places n restrictions on $\boldsymbol{H}_{k+1}$.

As $\boldsymbol{H}_{k+1}$ is symmetric positive definite, it has $n(n+1)/2$ parameters.

$\boldsymbol{H}_{k+1}$ has infinitely many solutions.

In order to determine $\boldsymbol{H}_{k+1}$ uniquely, we impose the condition that among all symmetric matrices that satisfy the secant equation, $\boldsymbol{H}_{k+1}$ is in some sense closest to $\boldsymbol{H}_k$.

We form the optimization problem
where $\boldsymbol{s}_k$ and $\boldsymbol{y}_k$ satisfy the curvature condition and $\boldsymbol{H}_k$ is symmetric and positive definite.

Different matrix norms result in different quasi-Newton methods. Hence the following choices are arbitrary but they provide a nice solution and produce the BFGS algorithm update.

A norm that provides an easy solution to the optimization problem which is scale-invariant is the weighted Frobenius norm

$$\|A\|_{\boldsymbol{W}} \equiv \|\boldsymbol{W}^{1/2}\boldsymbol{A}\boldsymbol{W}^{1/2}\|_F,$$

where $\|\cdot\|_F$ is defined by $\|\boldsymbol{A}\|_F^2 = \sum_{i=1}^n \sum_{j=1}^n a_{ij}^2$.

The matrix $\boldsymbol{W}$ can be chosen to be any matrix satisfying $\boldsymbol{W}\boldsymbol{s}_k = \boldsymbol{y}_k$. One possible $\boldsymbol{W}$ is the inverse of the average Hessian. That is,

$$\boldsymbol{W} = \bar{\boldsymbol{G}}_k^{-1}, \qquad \text{where } \bar{\boldsymbol{G}}_k = \int_0^1 \nabla^2 f(\boldsymbol{x}_k + t\alpha_k \boldsymbol{d}_k)dt$$

Using the Taylor series approximation of $\nabla f(\boldsymbol{x}_{k+1})$ around $\boldsymbol{x}_k$

$$\nabla f(\boldsymbol{x}_{k+1}) = \nabla f(\boldsymbol{x}_k) + \int_0^1 \nabla^2 f(\boldsymbol{x}_k + t\alpha_k \boldsymbol{d}_k)\alpha_k \boldsymbol{d}_k dt,$$

we can show that $\bar{\boldsymbol{G}}_k$ satisfies the secant equations:

$$\boldsymbol{y}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k) = \bar{\boldsymbol{G}}_k \alpha_k \boldsymbol{d}_k = \bar{\boldsymbol{G}}_k \boldsymbol{s}_k,$$

The unique solution to this optimization problem is given by

$$\boldsymbol{H}_{k+1} = (\boldsymbol{I} - \rho_k \boldsymbol{s}_k \boldsymbol{y}_k^T)\boldsymbol{H}_k(\boldsymbol{I} - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^T) + \rho_k \boldsymbol{s}_k \boldsymbol{s}_k^T,$$

where $\rho_k = \frac{1}{\boldsymbol{y}_k^T \boldsymbol{s}_k}$.

[H] Given a starting point x_0, convergence tolerance $\epsilon > 0$, and inverse Hessian approximation $\boldsymbol{H}_0$ $k \leftarrow 0$ $\|\nabla f(\boldsymbol{x}_k)\| > \epsilon$ Compute the search direction

$$\boldsymbol{d}_k = -\boldsymbol{H}_k \nabla f(\boldsymbol{x}_k)$$

Set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k$, where α_k is computed via a line search procedure to satisfy the Wolfe conditions Define $\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$ and $\boldsymbol{y}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$ Compute $\boldsymbol{H}_{k+1}$ using $\boldsymbol{H}_{k+1} = (\boldsymbol{I} - \rho_k \boldsymbol{s}_k \boldsymbol{y}_k^T)\boldsymbol{H}_k(\boldsymbol{I} - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^T) + \rho_k \boldsymbol{s}_k \boldsymbol{s}_k^T$ $k \leftarrow k + 1$ The BFGS Algorithm

Thank you!

Appendix — Optimizing the Step Size

For general functions, BFGS converges globally if the Wolfe conditions are imposed on the line search.

Definition (Wolfe Conditions)

A step length α_k satisfies the Wolfe conditions in direction $\boldsymbol{d}_k$ if the following inequalities hold:

1. $f(\boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k) \leq f(\boldsymbol{x}_k) + c_1 \alpha_k \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d}_k$,
2. $\nabla f(\boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k)^T \boldsymbol{d}_k \geq c_2 \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d}_k$,

where $0 < c_1 < c_2 < 1$. We may refer to the first condition as the sufficient decrease 1, and the second condition as the curvature condition.

Appendix — Initializing the Hessian

- We often set $\boldsymbol{H}_0 = \beta \boldsymbol{I}$, for some scalar β.
- A common heuristic is to initially set $\boldsymbol{H}_0 = \boldsymbol{I}$, then set $\boldsymbol{H}_0 \leftarrow \frac{\boldsymbol{y}_k^T \boldsymbol{s}_k}{\boldsymbol{y}_k^T \boldsymbol{y}_k} \boldsymbol{I}$.

Heuristic reasoning: Makes the 'size' of $\boldsymbol{H}_0$ similar to $(\nabla^2 f(\boldsymbol{x}_0))^{-1}$. Assuming the average Hessian is positive definite, we can write $\bar{\boldsymbol{G}}_k = \bar{\boldsymbol{G}}_k^{1/2} \bar{\boldsymbol{G}}_k^{1/2}$. Then using $\boldsymbol{y}_k = \bar{\boldsymbol{G}}_k \boldsymbol{s}_k$,

$$\frac{\boldsymbol{y}_k^T \boldsymbol{s}_k}{\boldsymbol{y}_k^T \boldsymbol{y}_k} = \frac{(\bar{\boldsymbol{G}}_k^{1/2} \boldsymbol{s}_k)^T \bar{\boldsymbol{G}}_k^{1/2} \boldsymbol{s}_k}{(\bar{\boldsymbol{G}}_k^{1/2} \boldsymbol{s}_k)^T \bar{\boldsymbol{G}}_k \bar{\boldsymbol{G}}_k^{1/2} \boldsymbol{s}_k} = \frac{\boldsymbol{z}_k^T \boldsymbol{z}_k}{\boldsymbol{z}_k^T \bar{\boldsymbol{G}}_k \boldsymbol{z}_k}$$

Appendix — Solving the Minimization Problem

We have the optimization problem

$$\begin{aligned} \text{minimize}_{\boldsymbol{H}} \quad & \|\boldsymbol{H} - \boldsymbol{H}_k\|, \\ \text{subject to} \quad & \boldsymbol{H} = \boldsymbol{H}^T, \\ & \boldsymbol{H}\boldsymbol{y}_k = \boldsymbol{s}_k, \end{aligned}$$

where $\boldsymbol{s}_k$ and $\boldsymbol{y}_k$ satisfy the curvature condition and $\boldsymbol{H}_k$ is symmetric and positive definite.

Appendix — Solving the Minimization Problem

We introduce the following notation:

$$\begin{aligned}
\widehat{\boldsymbol{H}} &= \boldsymbol{W}^{1/2}\boldsymbol{H}\boldsymbol{W}^{1/2} \\
\widehat{\boldsymbol{H}}_k &= \boldsymbol{W}^{1/2}\boldsymbol{H}_k\boldsymbol{W}^{1/2} \\
\widehat{\boldsymbol{s}}_k &= \boldsymbol{W}^{1/2}\boldsymbol{s}_k \\
\widehat{\boldsymbol{y}}_k &= \boldsymbol{W}^{-1/2}\boldsymbol{y}_k
\end{aligned}$$

Then we have the minimization problem

$$\begin{aligned}
\text{minimize}_{\ \widehat{\boldsymbol{H}}} \quad & \|\widehat{\boldsymbol{H}} - \widehat{\boldsymbol{H}}_k\|_F, \\
\text{subject to} \quad & \widehat{\boldsymbol{H}}\widehat{\boldsymbol{y}}_k = \widehat{\boldsymbol{y}}_k (= \widehat{\boldsymbol{s}}_k),
\end{aligned}$$

Note that $\widehat{\boldsymbol{y}}_k$ is an eigenvector of $\widehat{\boldsymbol{H}}$.

Appendix — Solving the Minimization Problem

We introduce a new orthonormal bases: $\boldsymbol{U} = [\boldsymbol{u} \mid \boldsymbol{u}_\perp]$. Let $\boldsymbol{u}$ be the normalized eigenvector $\widehat{\boldsymbol{y}}_k$.
Multiplications by orthogonal matrices preserve length, and the Frobenius norm is unitary invariant. Hence

$$\|\widehat{\boldsymbol{H}} - \widehat{\boldsymbol{H}}_k\|_F = \|\boldsymbol{U}^T\widehat{\boldsymbol{H}}_k\boldsymbol{U} - \boldsymbol{U}^T\widehat{\boldsymbol{H}}\boldsymbol{U}\|_F = \left\| \begin{bmatrix} * & * \\ * & \boldsymbol{u}_\perp^T\widehat{\boldsymbol{H}}_k\boldsymbol{u}_\perp \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & \boldsymbol{u}_\perp^T\widehat{\boldsymbol{H}}\boldsymbol{u}_\perp \end{bmatrix} \right\|_F$$

We can only modify the matrix on the right, and in this the only element we can change is the lower right element. In order to minimize the norm we must set the lower left elements of the matrices to be equal.

$$\boldsymbol{u}_\perp^T\widehat{\boldsymbol{H}}\boldsymbol{u}_\perp = \boldsymbol{u}_\perp^T\widehat{\boldsymbol{H}}_k\boldsymbol{u}_\perp$$

Appendix — Solving the Minimization Problem

$$\boldsymbol{u}_\perp^T \widehat{\boldsymbol{H}} \boldsymbol{u}_\perp = \boldsymbol{u}_\perp^T \widehat{\boldsymbol{H}}_k \boldsymbol{u}_\perp$$

This gives the optimal solution

$$\begin{aligned}
\widehat{\boldsymbol{H}} &= \boldsymbol{U} \begin{bmatrix} 1 & 0 \\ 0 & \boldsymbol{u}_\perp^T \widehat{\boldsymbol{H}}_k \boldsymbol{u}_\perp \end{bmatrix} \boldsymbol{U}^T \\
&= \begin{bmatrix} \boldsymbol{u} & \boldsymbol{u}_\perp \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \boldsymbol{u}_\perp^T \widehat{\boldsymbol{H}}_k \boldsymbol{u}_\perp \end{bmatrix} \begin{bmatrix} \boldsymbol{u}^T \\ \boldsymbol{u}_\perp^T \end{bmatrix} \\
&= \boldsymbol{u}\boldsymbol{u}^T + \boldsymbol{u}_\perp \boldsymbol{u}_\perp^T \widehat{\boldsymbol{H}}_k \boldsymbol{u}_\perp \boldsymbol{u}_\perp^T \\
&= \boldsymbol{u}\boldsymbol{u}^T + (\boldsymbol{I} - \boldsymbol{u}\boldsymbol{u}^T) \widehat{\boldsymbol{H}}_k (\boldsymbol{I} - \boldsymbol{u}\boldsymbol{u}^T)
\end{aligned}$$