# Lecture 10

# Multi-Dimensional Optimization: Newton-Based Algorithms

## 10.1 Lecture Objectives

- Understand the fundamentals of Newton's method
- Understand the major limitations of Newton's method, which lead to both quasi-Newton methods as well as the Levenberg-Marquardt modification
- Understand the variation of Newton's method commonly used to solve nonlinear least-squares optimization problems.

## 10.2 Newton's Method

Similar to the version used for one-dimensional line search, Newton's method as applied to multi-dimensional optimization problems works at each step $k$ by approximating the desired cost function $f(\mathbf{x})$ as a quadratic function around the current estimate $\mathbf{x}^{(k)}$. See figure 10.1 for a pictorial representation in the one-dimensional case.

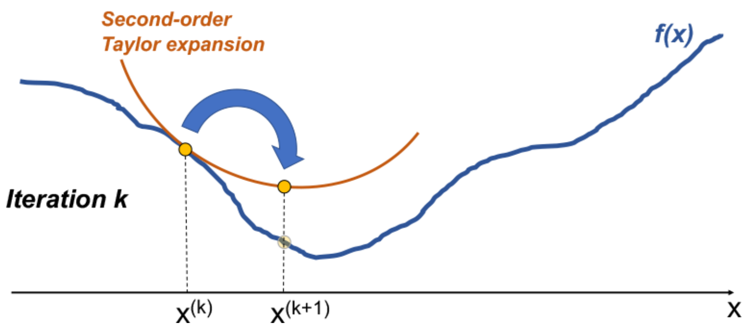

Figure 10.1: Pictorial depiction of the Newton method.

In the multi-dimensional version of Newton's method, we approximate the function $f(\mathbf{x})$ as a quadratic function around the current estimate $\mathbf{x}^{(k)}$:

$$f(\mathbf{x}) \approx f(\mathbf{x}^{(k)}) + (\mathbf{x} - \mathbf{x}^{(k)})^T \nabla f(\mathbf{x}^{(k)}) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^{(k)})^T \nabla^2 f(\mathbf{x}^{(k)}) (\mathbf{x} - \mathbf{x}^{(k)}) \tag{10.1}$$

which leads to the following iteration update:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - (\nabla^2 f(\mathbf{x}^{(k)}))^{-1} \nabla f(\mathbf{x}^{(k)}) \tag{10.2}$$

which is analogous to the one-dimensional Newton's method studied a couple of lectures ago.

### 10.2.1 Limitations of Newton's method

Similarly to the one-dimensional case (or even more so), the multi-dimensional version of Newton's method has two main limitations:

i Computational cost of evaluating and inverting the matrix of second derivatives (Hessian) at each iteration. Sometimes the second derivative cannot be evaluated analytically.

ii Potential for the Hessian to not be positive definite, which may lead to a non-invertible matrix (analogous to the 1D case when the second derivative is zero), or may lead to a non-descent direction (analogous to the 1D case where the second derivative is negative).

### 10.2.2 Quasi-Newton methods

The first challenge above may be addressed through a variety of approximated Newton techniques that do not require evaluation of the full Hessian matrix. These methods, termed 'quasi-Newton', effectively solve the following linear problem at each iteration:

$$\mathbf{B}^{(k)} \Delta \mathbf{x} = -\nabla f(\mathbf{x}^{(k)}) \tag{10.3}$$

for a matrix $\mathbf{B}^{(k)}$ that may be updated at each iteration. In general, a number of different methods have this same structure, where they seek a choice of matrix $\mathbf{B}^{(k)}$ that approximates the Hessian, can be calculated using the gradients of previous iterations, and is easy to invert. The secant method described in one-dimensional line search lecture provides an illustration for this kind of approach. A popular choice for quasi-Newton method is based on the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update[1].

[1]For more details on BFGS and other quasi-Newton algorithms, there are various sources including http://www.stat.cmu.edu/ ryantibs/convexopt-F13/lectures/11-QuasiNewton.pdf

### 10.2.3 Levenberg-Marquardt algorithm

To address the second challenge of Newton's method (ie: cases where the Hessian is not always positive definite), the Levenberg-Marquardt (LM) algorithm uses a "modified" Hessian and iterates as follows:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - (\nabla^2 f(\mathbf{x}^{(k)}) + \mu_k \mathbf{I})^{-1} \nabla f(\mathbf{x}^{(k)}) \tag{10.4}$$

for some $\mu_k > 0$, and where $\mathbf{I}$ is the identity matrix. Note that LM provides a sort of compromise between the Newton direction and the steepest descent direction, as shown in the term $\nabla^2 f(\mathbf{x}^{(k)}) + \mu_k \mathbf{I}$. For very small values of $\mu_k$, the $\nabla^2 f(\mathbf{x}^{(k)})$ component will dominate and the LM algorithm will behave similarly to Newton's method. For large values of $\mu_k$, the $\mu_k \mathbf{I}$ component will dominate and the direction chosen will be the negative gradient (as in the steepest descent method).

### 10.2.4 Newton's method for nonlinear least-squares problems

A very common type of optimization problem consists of fitting a nonlinear signal model to a set of acquired data using least-squares fitting. In this type of problem, our cost function $f(\mathbf{x})$ can be written as follows:

$$f(\mathbf{x}) = \sum_{k=1}^{K} r_k(\mathbf{x})^2 = \mathbf{r}(\mathbf{x})^T \mathbf{r}(\mathbf{x}) \tag{10.5}$$

where $\mathbf{r}(\mathbf{x}) = [r_1(\mathbf{x}), \cdots, r_K(\mathbf{x})]^T$. Note that in this problem we are seeking to fit $K$ data points (ie: drive each of the $r_k$ close to zero) using a set of $N$ variables (each of the entries $x_n$ in our $\mathbf{x}$ vector). Importantly, this type of structure in our objective function allows us to efficiently calculate the gradient and the Hessian at arbitrary points. An important tool to help us calculate these quantities is the Jacobian matrix.

The Jacobian matrix for a problem where we try to fit $K$ data points using a set of $N$ variables is the $K \times N$ matrix given by:

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial r_1}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial r_1}{\partial x_N}(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial r_K}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial r_K}{\partial x_N}(\mathbf{x}) \end{bmatrix} \tag{10.6}$$

Importantly, the gradient for a nonlinear least-squares problem can be written as:

$$\nabla f(\mathbf{x}) = 2\mathbf{J}(\mathbf{x})^T \mathbf{r}(\mathbf{x}) \tag{10.7}$$

and the Hessian can be written as:

$$\nabla^2 f(\mathbf{x}) = 2\left(\mathbf{J}(\mathbf{x})^T \mathbf{J}(\mathbf{x}) + \mathbf{S}(\mathbf{x})\right) \tag{10.8}$$

where $\mathbf{S}(\mathbf{x})$ is the $N \times N$ matrix defined as:

$$\mathbf{S}(\mathbf{x}) = \sum_{k=1}^{K} r_k(\mathbf{x})\, \nabla^2 r_k(\mathbf{x}) \tag{10.9}$$

Oftentimes, the second component of the Hessian ($\mathbf{S}(\mathbf{x})$) is small compared to the first component ($\mathbf{J}(\mathbf{x})^T\, \mathbf{J}(\mathbf{x})$). In such situations, it is warranted to drop this second component leading to an approximated version of the Hessian that depends solely on the Jacobian matrix:

$$\nabla^2 f(\mathbf{x}) \approx 2\mathbf{J}(\mathbf{x})^T\, \mathbf{J}(\mathbf{x}) \tag{10.10}$$

Using this approximation for non-linear least-squares optimization problems leads to the so-called Gauss-Newton algorithm, with the following iteration update:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - (\mathbf{J}(\mathbf{x})^T\, \mathbf{J}(\mathbf{x}))^{-1}\, \mathbf{J}(\mathbf{x})^T\, \mathbf{r}(\mathbf{x}) \tag{10.11}$$

which does not require the calculation of second derivatives, and yet provides good performance (rapid convergence) for a wide variety of nonlinear least-squares fitting algorithms.

Note that, similarly to the original Newton's method, the approximated Hessian matrix used in Gauss-Newton ($\mathbf{J}(\mathbf{x})^T\, \mathbf{J}(\mathbf{x})$) need not be positive definite. In such cases, the Levenberg-Marquardt modification can also be applied to Gauss-Newton, leading to the following update:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - (\mathbf{J}(\mathbf{x})^T\, \mathbf{J}(\mathbf{x}) + \mu_k \mathbf{I})^{-1}\, \mathbf{J}(\mathbf{x})^T\, \mathbf{r}(\mathbf{x}) \tag{10.12}$$

## 10.3 Trade-offs in Iterative Algorithms

Iterative optimization algorithms (including both gradient-based and Newton-based methods) generally present a trade-off between:

i The convergence rate, ie: how much closer to the minimizer does each additional iteration get us. This determines the number of iterations required before stopping.

ii The complexity of each iteration.

Usually, algorithms where each individual iteration is very simple require a lot of iterations to converge. Conversely, algorithms where a single iteration gets us a lot closer to the minimizer contain substantial complexity within each iteration.

## 10.4 Dealing with Constraints

Methods for dealing with constraints in Newton-type algorithms are analogous to those employed in gradient-based algorithms (see previous lecture).