CME 108/MATH 114
Introduction to Scientific Computing
Summer 2020

Newton's Method (Optimization) and Steepest Gradient Descent (GD)

As motivation for our first method, first recall the basic procedure for locating extrema of functions of a single variable, which you learned in an introductory Calculus course.

Question: How can we find the extrema of a smooth function $f : \mathbb{R} \to \mathbb{R}$?

Answer: The critical points of f are the candidates for local extrema. If f is smooth the only possible option are the roots of f'.

It turns out this is true more generally:

Theorem 0.0.1. *(Fermat) If $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable and f has a local extremum at $x = x^*$, then*

$$\nabla f(x^*) = 0$$

Fermat's theorem implies that the candidate extrema of a smooth function f are roots of the (possibly non-linear) system

$$\frac{\partial f}{\partial x_i}(x) = 0, \quad i = 1, \ldots, n.$$

Optimization becomes a root finding problem, and we can apply the techniques we've discussed before!

In particular, notice that solving $\nabla f(x) = 0$ is equivalent to solving the (possibly nonlinear) system of equations

$$\frac{\partial f}{\partial x_i} = 0, \qquad i = 1, \ldots, n.$$

Newton's method Our first approach is therefore to use Newton's method, applied to

$$g(x) = \nabla f(x)^T = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \tag{1}$$

Recall this is an iterative process which prescribes

$$x^{(k+1)} \leftarrow x^{(k)} - J_g^{-1}(x^{(k)})g(x^{(k)})$$

In this case, the Jacobian matrix of g contains the second partials of f:

$$\begin{aligned} J_g(x^{(k)}) &= \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g_n}{\partial x_1} & \frac{\partial g_n}{\partial x_2} & \cdots & \frac{\partial g_n}{\partial x_n} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \frac{\partial^2 f}{\partial x_2 \partial x_n} & \cdots & \frac{\partial f^2}{\partial x_n \partial x_n} \end{bmatrix} \\ &= H_f(x^{(k)}) \end{aligned}$$

The matrix H_f of second partials of f is called the "Hessian". Notice that if f is sufficiently smooth, $\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i}$, so H_f is symmetric at every point.

Therefore, in terms of f Newton's method for optimization suggest:

Algorithm 1: Newton's method (Optimization)

initialize $x^{(0)}$
for k **in** *1, to max-iter,* **do**
  $p^{(k)} = -H_f(x^{(k)}) \backslash \nabla f(x^{(k)})^T$ `// since we interpret the gradient as a row vector, this is a column`
  if $||p^{(k)}|| < \epsilon_{\text{tol}}$ **then**
    break
  end
  $x^{(k+1)} \leftarrow x^{(k)} + p(k)$
end

Therefore, to implement this method, we need to compute both ∇f and H_f

Remark 0.0.2. Again recall that the convergence of Newton's method is highly dependent on the starting point, so it is not necessarily "easy" to locate the extrema of a smooth function.

Note As in the one-dimensional case, there is an analogous Second Derivative Test which allows us to classify the extrema of a function. This test depends on a

generalized notion of the "sign" of H_f. For instance, if f has a critical point at $x = x^*$ and $H_f(x^*)$ is SPD, the f has a local minimum at x^*.

If symmetric matrices are compared to real numbers, then SPD matrices are the analogue of positive numbers. An SPD Hessian $y^T H_f(x^*)y > 0$ for all $y \in \mathbb{R}^n$ classifies the critical point $x = x^*$ as a local minimum, much like a positive second derivative $f''(x^*) > 0$ classifies a local minimum in the one-dimensional case.

Let's take a step backwards and think about the picture associated to this version of Newton's method. For simplicity, consider the one-dimensional case. At each step, we find the zero of the line tangent to f' and move to it. If we think about what goes on at the level of f (think about integrating everything), we see that each step we draw a quadratic which is tangent to f (if both touch f at $x = x^{(k)}$), and it has the same slope as f there) and we move to its exact minimum/maximum.

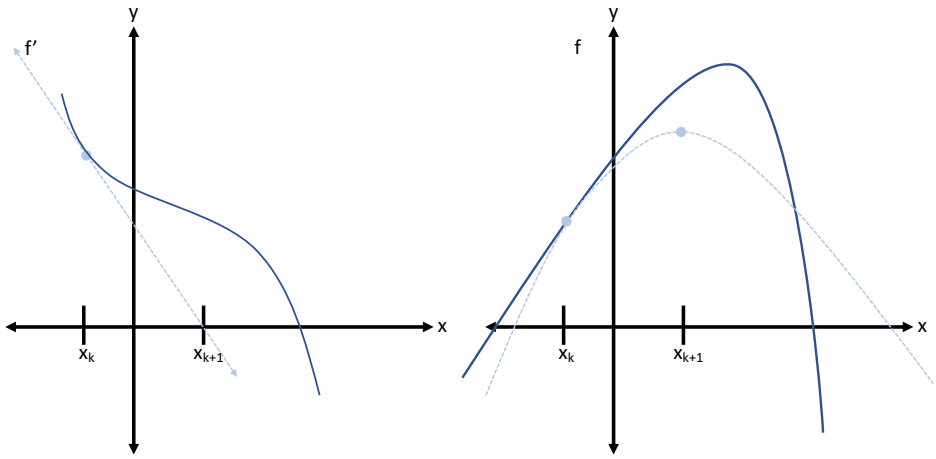

Figure 1: Geometric intuition of the MVT

Aside from Newton's method, there are many other optimization methods. We will discuss one other basic idea here.

Gradient descent The steepest descent, gradient descent, or steepest gradient descent method is perhaps the most intuitive method of all.

Remark 0.0.3. In modern literature and codebases, the abbreviation "SGD" might refer to the *stochastic* gradient descent method, which is a variant on the method we discuss here.

Question: What's the quickest way down a mountain?

Answer: At each step, look for the steepest downward direction and step in that direction.

Recall that the directional derivative $D_u f(x) = u \cdot \nabla f(x)$ gives the change in the surface height **in the direction** of the (unit) vector u. The Cauchy-Schwarz inequality implies

$$D_u f(x) \geq -||u||||\nabla f(x)|| = -||\nabla f(x)||,$$

with equality only when $u = \frac{-\nabla f(x)}{||\nabla f(x)||}$

Thus at each iteration of the steepest descent method we step in the direction of $-\nabla f(x)$:

$$x^{(k+1)} \leftarrow x^{(k)} - \alpha \nabla f(x^{(k)})$$

where $\alpha > 0$ is a chosen step size, also known as the "learning rate" in some contexts.

Ideally, we'd step the perfect amount every time: α is chosen so that $\Phi(\alpha) = f(x^{(k)} - \alpha \nabla f(x^{(k)}))$ is as small as possible by moving **in the direction of** $-\nabla f(x^{(k)})$.

Thus our pseudocode is given by:

Algorithm 2: Steepest gradient descent with exact line search

```
initialize x^(0)
for k in 1, to max-iter, do
    α* = argmin_α Φ(α)   // minimum along path on f in the direction
      of −∇f(x^(k))
    x^(k+1) ← x^(k) − α*∇f(x^(k))
    if ||x^(k+1) − x^(k)|| < ε_tol then
        break
    end
end
```

The optimization problem $\min_\alpha \Phi(\alpha)$ is a one-dimensional problem that implementations of gradient descent typically solve using a **zeroth-order** method (a method that doesn't require information about derivatives of Φ), like golden section search.

This is called "exact line search".

Remark 0.0.4. Finding the optimal step size typically adds quite a bit of computational overheard so in practice we simply approximate it.

The simplest approximation strategy uses a constant learning state throughout the procedure.

Alternatively, we can use something like the (inexact) backtracking line search.

Example 0.0.5.

$$f(x) = 3x_1^2 + 4x_2^2$$

Clearly the minimum is at $(0, 0)$*, since* $f(x) \geq 0$*. Steepest descent method, starting at* $x^{(0)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$*, gives:*

$$\begin{aligned} \Phi(\alpha) &= f(x^{(0)} - \alpha \nabla f(x^{(0)})) && (2) \\ &= f\left(\begin{bmatrix} 1 \\ 1 \end{bmatrix} - \alpha \begin{bmatrix} 6 \\ 8 \end{bmatrix} \right) && (3) \\ &= 3(1 - 6\alpha)^2 + 4(1 - 8\alpha)^2 && (4) \\ &= 7 - 100\alpha + 133\alpha^2, && (5) \end{aligned}$$

and $\alpha^* = \arg\min_\alpha \Phi(\alpha) = \frac{-(-100)}{2(133)} = \frac{100}{266}$*, so*

$$\begin{aligned} x^{(1)} &= x^{(0)} - \alpha^* \nabla f(^{(0)}) && (6) \\ &= \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \frac{100}{266} \begin{bmatrix} 6 \\ 8 \end{bmatrix} && (7) \end{aligned}$$