Algorithms for unconstrained minimization

One of the benefits of minimizing convex functions is that we can often use very simple algorithms to find solutions. Specifically, we want to solve

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}),$$

where f is convex. For now we will assume that f is also differentiable.[1] We have just seen that, in this case, a necessary and sufficient condition for $\boldsymbol{x}^\star$ to be a minimizer is that the gradient vanishes:

$$\nabla f(\boldsymbol{x}^\star) = \mathbf{0}.$$

Thus, we can equivalently think of the problem of minimizing $f(\boldsymbol{x})$ as finding an $\boldsymbol{x}^\star$ that $\nabla f(\boldsymbol{x}^\star) = \mathbf{0}$. As noted before, it is not a given that such an $\boldsymbol{x}^\star$ exists, but for now we will assume that f does have (at least one) minimizer.

Every general-purpose optimization algorithm we will look at in this course is **iterative** — they will all have the basic form:

Iterative descent

```
Initialize: $k = 0$, $\boldsymbol{x}_0 =$ initial guess
while not converged do
    calculate a direction to move $\boldsymbol{d}_k$
    calculate a step size $\alpha_k \geq 0$
    $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k$
    $k = k + 1$
end while
```

[1]We will also be interested in cases where f is not differentiable. We will revisit this later in the course.

The central challenge in designing a good algorithm mostly boils down to computing the direction $\boldsymbol{d}_k$. As a preview, here are some choices that we will discuss:

1. **Gradient descent**: We take
$$\boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k).$$
This is the direction of "steepest descent" (where "steepest" is defined relative to the Euclidean norm). Gradient descent iterations are cheap, but many iterations may be required for convergence.

2. **Accelerated gradient descent**: We can sometimes reduce the number of iterations required by gradient descent by incorporating a *momentum* term. Specifically, we first compute
$$\boldsymbol{p}_k = (\boldsymbol{x}_k - \boldsymbol{x}_{k-1})$$
and then take
$$\boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k) + \frac{\beta_k}{\alpha_k}\boldsymbol{p}_k$$
or
$$\boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k + \beta_k \boldsymbol{p}_k) + \frac{\beta_k}{\alpha_k}\boldsymbol{p}_k.$$
The "heavy ball" method and conjugate gradient descent use the former update rule; Nesterov's method uses the latter. We will see later that by incorporating this momentum term, we can sometimes dramatically reduce the number of iterations required for convergence.

3. **Newton's method**: Gradient descent methods are based on building linear approximations to the function at each iteration. We can also build a quadratic model around $\boldsymbol{x}_k$ then compute

the exact minimizer of this quadratic by solving a system of equations. This corresponds to taking

$$\boldsymbol{d}_k = -\left(\nabla^2 f(\boldsymbol{x}_k)\right)^{-1} \nabla f\left(\boldsymbol{x}_k\right),$$

that is, the inverse of the Hessian evaluated at $\boldsymbol{x}_k$ applied to the gradient evaluated at the same point. Newton iterations tend to be expensive (as they require a system solve), but they typically converge in far fewer iterations than gradient descent.

4. **Quasi-Newton methods**: If the dimension N of $\boldsymbol{x}$ is large, Newton's method is not computationally feasible. In this case we can replace the Newton iteration with

$$\boldsymbol{d}_k = -\boldsymbol{Q}_k \nabla f\left(\boldsymbol{x}_k\right)$$

where $\boldsymbol{Q}_k$ is an approximation or estimate of $\left(\nabla^2 f(\boldsymbol{x}_k)\right)^{-1}$. Quasi-Newton methods may require more iterations than a pure Newton approach, but can still be very effective.

Whichever direction we choose, it should be a **descent direction**, i.e., $\boldsymbol{d}_k$ should satisfy

$$\langle \boldsymbol{d}_k, \nabla f\left(\boldsymbol{x}_k\right) \rangle \leq 0.$$

Since f is convex, it is always true that

$$f\left(\boldsymbol{x} + \alpha \boldsymbol{d}\right) \geq f(\boldsymbol{x}) + \alpha \left\langle \boldsymbol{d}, \nabla f(\boldsymbol{x}) \right\rangle,$$

and so to decrease the value of the functional while moving in direction $\boldsymbol{d}$, it is necessary that the inner product above be negative.

Line search

Given a starting point $\boldsymbol{x}_k$ and a direction $\boldsymbol{d}_k$, we still need to decide on α_k, i.e., how far to move. With $\boldsymbol{x}_k$ and $\boldsymbol{d}_k$ fixed, we can think of the remaining problem as a one-dimensional optimization problem where we would like to choose α to minimize (or at least reduce)

$$\phi(\alpha) = f\left(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k\right).$$

Note that we don't necessarily need to find the true minimum – we aren't even sure that we are moving in the right direction at this point – but we would generally still like to make as much progress as possible before calculating a new direction $\boldsymbol{d}_{k+1}$. There are many methods for doing this, here are three:

Exact: Solve the 1D optimization program

$$\underset{\alpha \geq 0}{\text{minimize}} \;\; \phi(\alpha).$$

This is typically not worth the trouble, but there are instances (e.g., least squares and other unconstrained convex quadratic programs) when it can be solved analytically.

Fixed: We can also just use a constant step size $\alpha_k = \alpha$. This will work if the step size is small enough, but usually this results in way too many iterations.

Backtracking: The problem with a fixed step size is that we cannot guarantee convergence of α is too large, but when α is too small we may not make much progress on each iteration. A popular strategy is to do some kind of rudimentary search for a step size α that gives us sufficient progress as measured by the inequality

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k) \geq c\alpha \langle \boldsymbol{d}_k, \nabla f(\boldsymbol{x}_k) \rangle,$$

where $c \in (0,1)$. This is known as the **Armijo condition**. For α satisfying the inequality we have that the reduction in f is proportional to both the step length α and the directional derivative in the direction $\boldsymbol{d}_k$.

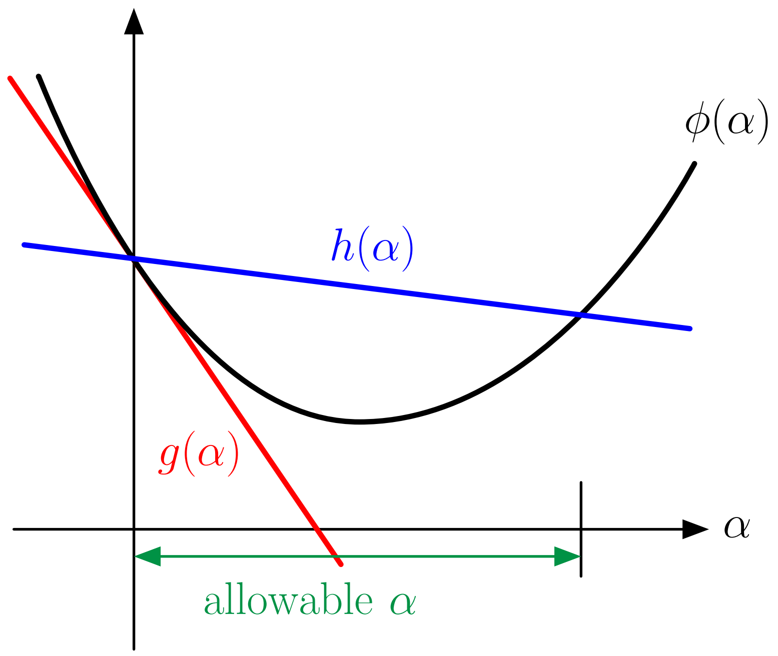
Note that we can equivalently write this condition as

$$\phi(\alpha) \leq h(\alpha) := f(\boldsymbol{x}_k) + c\alpha \langle \boldsymbol{d}_k, \nabla f(\boldsymbol{x}_k) \rangle.$$

Recall that from convexity, we also have that

$$\phi(\alpha) \geq g(\alpha) := f(\boldsymbol{x}_k) + \alpha \langle \boldsymbol{d}_k, \nabla f(\boldsymbol{x}_k) \rangle.$$

Since $c < 1$, we always have $\phi(\alpha) \leq h(\alpha)$ for sufficiently small α. An example is illustrated below:



We still haven't said anything about how to actually use the Armijo condition to pick α. Within the set of allowable α satisfying the condition, the (guaranteed) reduction in f is proportional to α, so we would generally like to select α to be large.

This inspires the following very simple **backtracking** algorithm: start with a step size of $\alpha = \bar{\alpha}$, and then decrease by a factor of ρ until the Armijo condition is satisfied.

Backtracking line search

```
Input: $\boldsymbol{x}_k$, $\boldsymbol{d}_k$, $\bar{\alpha} > 0$, $c \in (0,1)$, and $\rho \in (0,1)$.
Initialize: $\alpha = \bar{\alpha}$
while $\phi(\alpha) > h(\alpha)$ do
    $\alpha = \rho\alpha$
end while
```

The backtracking line search tends to be cheap, and works very well in practice. A common choice for $\bar{\alpha}$ is $\bar{\alpha} = \frac{1}{2}$, but this can vary somewhat depending on the algorithm. The choice of c can range from extremely small (10^{-4}, encouraging larger steps) to relatively large (0.3, encouraging smaller steps), and typical values of ρ range from 0.1, (corresponding to a relatively coarse search) to 0.8 (corresponding to a finer search).