CHAPTER 12

# Specialized Optimization Methods

## CONTENTS



OPTIMIZATION algorithms like Newton's method are completely generic approaches to minimizing a function $f(\vec{x})$, with or without constraints on $\vec{x}$. These algorithms make few assumptions about the form of $f$ or the constraints. Contrastingly, by designing the conjugate gradient algorithm specifically for minimizing the objective $f(\vec{x}) \equiv \frac{1}{2}\vec{x}^\top A\vec{x} - \vec{b}^\top \vec{x} + c$, we were able to guarantee more reliable and efficient behavior than general algorithms.

In this chapter, we continue to exploit special structure to solve optimization problems, this time for more complex nonlinear objectives. Replacing monolithic generic algorithms with ones tailored to a given problem can make optimization faster and easier to troubleshoot, although doing so requires more implementation effort than calling a pre-packaged solver.

## 12.1 NONLINEAR LEAST-SQUARES

Recall the nonlinear regression problem posed in Example 9.1. If we wish to fit a function $y = ce^{ax}$ to a set of data points $(x_1, y_1), \ldots, (x_k, y_k)$, an optimization mimicking linear least-squares is to minimize the function

$$E(a, c) \equiv \sum_i (y_i - ce^{ax_i})^2.$$

This energy reflects the fact that we wish $y_i - ce^{ax_i} \approx 0$ for all $i$.

More generally, suppose we are given a set of functions $f_1(\vec{x}), \ldots, f_k(\vec{x})$ for $\vec{x} \in \mathbb{R}^n$. If we want $f_i(\vec{x}) \approx 0$ for all $i$, then a reasonable objective trading off between these terms is

$$E_{\text{NLS}}(\vec{x}) \equiv \frac{1}{2}\sum_i [f_i(\vec{x})]^2.$$

Objective functions of this form are known as *nonlinear least-squares* problems. For the exponential regression problem above, we would take $f_i(a, c) \equiv y_i - ce^{ax_i}$.

### 12.1.1 Gauss-Newton

When we run Newton's method to minimize a function $f(\vec{x})$, we must know the gradient *and* Hessian of $f$. Knowing only the gradient of $f$ is not enough, since approximating functions with planes provides no information about their extrema. The BFGS algorithm carries out optimization without Hessians, but its approximate Hessians depend on the sequence of iterations and hence are not local to the current iterate.

Contrastingly, the Gauss-Newton algorithm for nonlinear least-squares makes the observation that approximating each $f_i$ with a linear function yields a nontrivial curved approximation of $E_{\text{NLS}}$ since each term in the sum is squared. The main feature of this approach is that it requires only *first-order* approximation of the $f_i$'s rather than Hessians.

Suppose we write

$$f_i(\vec{x}) \approx f_i(\vec{x}_0) + [\nabla f_i(\vec{x}_0)] \cdot (\vec{x} - \vec{x}_0).$$

Then, we can approximate $E_{\text{NLS}}$ with $E^0_{\text{NLS}}$ given by

$$E^0_{\text{NLS}}(\vec{x}) = \frac{1}{2}\sum_i \left(f_i(\vec{x}_0) + [\nabla f_i(\vec{x}_0)] \cdot (\vec{x} - \vec{x}_0)\right)^2.$$

Define $F(\vec{x}) \equiv (f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x}))$ by stacking the $f_i$'s into a column vector. Then,

$$E^0_{\text{NLS}}(\vec{x}) = \frac{1}{2}\|F(\vec{x}_0) + DF(\vec{x}_0)(\vec{x} - \vec{x}_0)\|_2^2,$$

where $DF$ is the Jacobian of $F$. Minimizing $E^0_{\text{NLS}}(\vec{x})$ is a *linear* least-squares problem $-F(\vec{x}_0) \approx DF(\vec{x}_0)(\vec{x} - \vec{x}_0)$ that can be solved via the normal equations:

$$\vec{x} = \vec{x}_0 - (DF(\vec{x}_0)^\top DF(\vec{x}_0))^{-1} DF(\vec{x}_0)^\top F(\vec{x}_0).$$

More practically, as we have discussed, the system can be solved using the QR factorization of $DF(\vec{x}_0)$ or—in higher dimensions—using conjugate gradients and related methods.

We can view $\vec{x}$ from minimizing $E^0_{\text{NLS}}(\vec{x})$ as an improved approximation of the minimum of $E_{\text{NLS}}(\vec{x})$ starting from $\vec{x}_0$. The Gauss-Newton algorithm iterates this formula to solve nonlinear least-squares:

$$\vec{x}_{k+1} = \vec{x}_k - (DF(\vec{x}_k)^\top DF(\vec{x}_k))^{-1} DF(\vec{x}_k)^\top F(\vec{x}_k).$$

This iteration is not guaranteed to converge in all situations. Given an initial guess sufficiently close to the minimum of the nonlinear least-squares problem, however, the approximation above behaves similarly to Newton's method and even can have quadratic convergence. Given the nature of the Gauss-Newton approximation, the algorithm works best when the optimal objective value $E_{\text{NLS}}(\vec{x}^*)$ is small; convergence can suffer when the optimal value is relatively large.

### 12.1.2 Levenberg-Marquardt

The Gauss-Newton algorithm uses an approximation $E^0_{\text{NLS}}(\vec{x})$ of the nonlinear least-squares energy as a proxy for $E_{\text{NLS}}(\vec{x})$ that is easier to minimize. In practice, this approximation is likely to fail as $\vec{x}$ moves farther from $\vec{x}_0$, so we might modify the Gauss-Newton step to include a step size limitation:

$$\begin{array}{rl} \min_{\vec{x}} & E^0_{\text{NLS}}(\vec{x}) \\ \text{subject to} & \|\vec{x}-\vec{x}_0\|_2^2 \leq \Delta. \end{array}$$

That is, we now restrict our change in $\vec{x}$ to have norm less than some user-provided value $\Delta$; the $\Delta$ neighborhood about $\vec{x}_0$ is called a *trust region*. Denote $H \equiv DF(\vec{x}_0)^\top DF(\vec{x}_0)$ and $\delta\vec{x} \equiv \vec{x}-\vec{x}_0$. Then, we can solve:

$$\begin{array}{rl} \min_{\delta\vec{x}} & \frac{1}{2}\delta\vec{x}^\top H\delta\vec{x} + F(\vec{x}_0)^\top DF(\vec{x}_0)\delta\vec{x} \\ \text{subject to} & \|\delta\vec{x}\|_2^2 \leq \Delta. \end{array}$$

That is, we displace $\vec{x}$ by minimizing the Gauss-Newton approximation after imposing the step size restriction. This problem has the following KKT conditions (see §10.2.2):

$$\begin{aligned} \text{Stationarity: } \vec{0} &= H\delta\vec{x} + DF(\vec{x}_0)^\top F(\vec{x}_0) + 2\mu\delta\vec{x} \\ \text{Primal feasibility: } \|\delta\vec{x}\|_2^2 &\leq \Delta \\ \text{Complementary slackness: } \mu(\Delta - \|\delta\vec{x}\|_2^2) &= 0 \\ \text{Dual feasibility: } \mu &\geq 0. \end{aligned}$$

Define $\lambda \equiv 2\mu$. Then, the stationarity condition can be written as follows:

$$(H + \lambda I_{n\times n})\delta\vec{x} = -DF(\vec{x}_0)^\top F(\vec{x}_0).$$

Assume the constraint $\|\delta\vec{x}\|_2^2 \leq \Delta$ is active, that is, $\|\delta\vec{x}\|_2^2 = \Delta$. Then, except in degenerate cases $\lambda > 0$; combining this inequality with the fact that $H$ is positive semidefinite, $H + \lambda I_{n\times n}$ must be positive definite.

The Levenberg-Marquardt algorithm *starts* from this stationarity formula, taking the following step derived from a user-supplied parameter $\lambda > 0$ [82, 85]:

$$\vec{x} = \vec{x}_0 - (DF(\vec{x}_0)^\top DF(\vec{x}_0) + \lambda I_{n\times n})^{-1} DF(\vec{x}_0)^\top F(\vec{x}_0).$$

This linear system also can be derived by applying Tikhonov regularization to the Gauss-Newton linear system. When $\lambda$ is small, it behaves similarly to the Gauss-Newton algorithm, while large $\lambda$ results in a gradient descent step for $E_{\text{NLS}}$.

Rather than specifying $\Delta$ as introduced above, Levenberg-Marquardt steps fix $\lambda > 0$ directly. By the KKT conditions, *a posteriori* we know this choice corresponds to having taken $\Delta = \|\vec{x}-\vec{x}_0\|_2^2$. As $\lambda \to \infty$, the step from Levenberg-Marquardt satisfies $\|\vec{x}-\vec{x}_0\|_2 \to 0$; so, we can regard $\Delta$ and $\lambda$ as approximately inversely proportional.

Typical approaches adaptively adjust the *damping parameter* $\lambda$ during each iteration:

$$\vec{x}_{k+1} = \vec{x}_k - (DF(\vec{x}_k)^\top DF(\vec{x}_k) + \lambda_k I_{n\times n})^{-1} DF(\vec{x}_0)^\top F(\vec{x}_k).$$

For instance, we can scale up $\lambda_k$ when the step in $E_{\text{NLS}}(\vec{x})$ agrees well with the approximate value predicted by $E^0_{\text{NLS}}(\vec{x})$, since this corresponds to increasing the size of the neighborhood in which the Gauss-Newton approximation is effective.

## 12.2 ITERATIVELY REWEIGHTED LEAST-SQUARES

Continuing in our consideration of least-squares problems, suppose we wish to minimize a function of the form:

$$E_{\text{IRLS}}(\vec{x}) \equiv \sum_i f_i(\vec{x})[g_i(\vec{x})]^2.$$

We can think of $f_i(\vec{x})$ as a *weight* on the least-squares term $g_i(\vec{x})$.

**Example 12.1** ($L^p$ optimization)**.** Similar to the compressed sensing problems in §10.4.1, given $A \in \mathbb{R}^{m\times n}$ and $\vec{b} \in \mathbb{R}^m$ we can generalize least-squares by minimizing

$$E_p(\vec{x}) \equiv \|A\vec{x} - \vec{b}\|_p^p.$$

Choosing $p = 1$ can promote sparsity in the residual $\vec{b} - A\vec{x}$. We can write this function in an alternative form:

$$E_p(\vec{x}) = \sum_i (\vec{a}_i \cdot \vec{x} - b_i)^{p-2}(\vec{a}_i \cdot \vec{x} - b_i)^2.$$

Here, we denote the rows of $A$ as $\vec{a}_i^\top$. Then, $E_p = E_{\text{IRLS}}$ after defining:

$$f_i(\vec{x}) = (\vec{a}_i \cdot \vec{x} - b_i)^{p-2}$$
$$g_i(\vec{x}) = \vec{a}_i \cdot \vec{x} - b_i.$$

The *iteratively reweighted least-squares* (IRLS) algorithm makes use of the following fixed point iteration:

$$\vec{x}_{k+1} = \min_{\vec{x}_{k+1}} \sum_i f_i(\vec{x}_k)[g_i(\vec{x}_{k+1})]^2.$$

In the minimization, $\vec{x}_k$ is fixed, so the optimization is a least-squares problem over the $g_i$'s. When $g_i$ is linear, the minimization can be carried out via linear least-squares; otherwise we can use the nonlinear least-squares techniques in §12.1.

**Example 12.2** ($L^1$ optimization)**.** Continuing Example 12.1, suppose we take $p = 1$. Then,

$$E_1(\vec{x}) = \sum_i |\vec{a}_i \cdot \vec{x} - b_i| = \sum_i \frac{1}{|\vec{a}_i \cdot \vec{x} - b_i|}(\vec{a}_i \cdot \vec{x} - b_i)^2.$$

This functional leads to the following IRLS iteration, after adjustment for numerical issues:

| | |
|---|---|
| $w_i \leftarrow [\max(\lvert\vec{a}_i \cdot \vec{x} - b_i\rvert, \delta)]^{-1}$ | ▷ Recompute weights |
| $\vec{x} \leftarrow \min_{\vec{x}} \sum_i w_i(\vec{a}_i \cdot \vec{x} - b_i)^2$ | ▷ Linear least-squares |

The parameter $\delta > 0$ avoids division by zero; large values of $\delta$ make better-conditioned linear systems but worse approximations of the original $\|\cdot\|_1$ problem.

**Example 12.3** (Weiszfeld algorithm)**.** Recall the geometric median problem from Example 9.3. In this problem, given $\vec{x}_1, \ldots, \vec{x}_k \in \mathbb{R}^n$, we wish to minimize

$$E(\vec{x}) \equiv \sum_i \|\vec{x} - \vec{x}_i\|_2.$$

Similar to the $L^1$ problem in Example 12.2, we can write this function like a weighted least-squares problem:

$$E(\vec{x}) \equiv \sum_i \frac{1}{\|\vec{x} - \vec{x}_i\|_2} \|\vec{x} - \vec{x}_i\|_2^2.$$

Then, IRLS provides the *Weiszfeld algorithm* for geometric median problems:

$$\begin{array}{ll} w_i \leftarrow [\max(\|\vec{x} - \vec{x}_i\|_2, \delta)]^{-1} & \triangleright \text{ Recompute weights} \\ \vec{x} \leftarrow \min_{\vec{x}} \sum_i w_i (\vec{x} - \vec{x}_i)^2 & \triangleright \text{ Linear least-squares} \end{array}$$

We can solve for the second step of the Weiszfeld algorithm in closed form. Differentiating the objective with respect to $\vec{x}$ shows

$$\vec{0} = \sum_i 2w_i(\vec{x} - \vec{x}_i) \implies \vec{x} = \frac{\sum_i w_i \vec{x}_i}{\sum_i w_i}.$$

Thus, the two alternating steps of Weiszfeld's algorithm can be carried out efficiently as:

$$\begin{array}{ll} w_i \leftarrow [\max(\|\vec{x} - \vec{x}_i\|_2, \delta)]^{-1} & \triangleright \text{ Recompute weights} \\ \vec{x} \leftarrow \frac{\sum_i w_i \vec{x}_i}{\sum_i w_i} & \triangleright \text{ Weighted centroid} \end{array}$$

IRLS algorithms are straightforward to formulate, so they are worth trying if an optimization can be written in the form of $E_{\text{IRLS}}$. When $g_i$ is linear for all $i$ as in Example 12.2, each iteration of IRLS can be carried out quickly using Cholesky factorization, QR, conjugate gradients, and so on, avoiding line search and other more generic strategies.

It is difficult to formulate general conditions under which IRLS will reach the minimum of $E_{\text{IRLS}}$. Often, iterates must be approximated somewhat as in the introduction of $\delta$ to Example 12.2 to avoid division by zero and other degeneracies. In the case of $L^1$ optimization, however, IRLS can be shown with small modification to converge to the optimal point [31].

## 12.3 COORDINATE DESCENT AND ALTERNATION

Suppose we wish to minimize a function $f : \mathbb{R}^{n+m} \to \mathbb{R}$. Rather than viewing the input as a single variable $\vec{x} \in \mathbb{R}^{n+m}$, we might write $f$ in an alternative form as $f(\vec{x}, \vec{y})$, for $\vec{x} \in \mathbb{R}^n$ and $\vec{y} \in \mathbb{R}^m$. One strategy for optimization is to fix $\vec{y}$ and minimize $f$ with respect to $\vec{x}$, fix $\vec{x}$ and minimize $f$ with respect to $\vec{y}$, and repeat:

$$\begin{array}{ll} \textbf{for } i \leftarrow 1, 2, \ldots & \\ \quad \vec{x}_{i+1} \leftarrow \min_{\vec{x}} f(\vec{x}, \vec{y}_i) & \triangleright \text{ Optimize } \vec{x} \text{ with } \vec{y} \text{ fixed} \\ \quad \vec{y}_{i+1} \leftarrow \min_{\vec{y}} f(\vec{x}_{i+1}, \vec{y}) & \triangleright \text{ Optimize } \vec{y} \text{ with } \vec{x} \text{ fixed} \end{array}$$

In this *alternating* approach, the value of $f(\vec{x}_i, \vec{y}_i)$ decreases monotonically as $i$ increases since a minimization is carried out at each step. We cannot prove that alternation always reaches a global or even local minimum, but in many cases it can be an efficient option for otherwise challenging problems.

### 12.3.1 Identifying Candidates for Alternation

There are a few reasons why we might wish to perform alternating optimization:

- The individual problems over $\vec{x}$ and $\vec{y}$ are optimizations in a lower dimension and may converge more quickly.
- We may be able to split the variables in such a way that the individual $\vec{x}$ and $\vec{y}$ steps are far more efficient than optimizing both variables jointly.

Below we provide a few examples of alternating optimization in practice.

**Example 12.4** (Generalized PCA)**.** In the PCA problem from §7.2.5, we are given a data matrix $X \in \mathbb{R}^{n\times k}$ whose columns are $k$ data points in $\mathbb{R}^n$. We seek a basis in $\mathbb{R}^n$ of size $d$ such that the projection of the data points onto the basis introduces minimal approximation error; we will store this basis in the columns of $C \in \mathbb{R}^{n\times d}$. Classical PCA minimizes $\|X - CY\|_{\text{Fro}}^2$ over both $C$ and $Y$, where the columns of $Y \in \mathbb{R}^{d\times k}$ are the coefficients of the data points in the $C$ basis. If $C$ is constrained to be orthogonal, then $Y = C^\top X$, recovering the formula in our previous discussion.

The Frobenius norm in PCA is somewhat arbitrary: The relevant relationship is $X - CY \approx 0$. Alternative PCA models minimize $\mu(X - CY)$ over $C$ and $Y$, for some other energy function $\mu : \mathbb{R}^{n\times k} \to \mathbb{R}$ favoring matrices with entries near zero; $\mu$ can provide enhanced robustness to noise or encode application-specific assumptions. Taking $\mu(M) \equiv \|M\|_{\text{Fro}}^2$ recovers classical PCA; another popular choice is *robust PCA* [71], which takes $\mu(M) \equiv \sum_{ij} |M_{ij}|$.

The product $CY$ in $\mu(X - CY)$ makes the energy nonlinear and nonconvex. A typical minimization routine for this problem uses alternation: First optimize $C$ with $Y$ fixed, then optimize $Y$ with $C$ fixed, and repeat. Whereas optimizing the energy with respect to $C$ and $Y$ jointly might require a generic large-scale method, the individual alternating $C$ and $Y$ steps can be easier:

- When $\mu(M) = \|M\|_{\text{Fro}}^2$, both the $Y$ and $C$ alternations are least-squares problems, leading to the *alternating least-squares* (ALS) algorithm for classical PCA.
- When $\mu(M) \equiv \sum_{ij} |M_{ij}|$, the $Y$ and $C$ alternations are linear programs, which can be optimized using the techniques mentioned in §10.4.1.

**Example 12.5** (ARAP)**.** Recall the planar "as-rigid-as-possible" (ARAP) problem introduced in Example 10.5:

$$\begin{aligned} \text{minimize}_{R_v,\vec{y}_v} & \sum_{v\in V} \sum_{(v,w)\in E} \|R_v(\vec{x}_v - \vec{x}_w) - (\vec{y}_v - \vec{y}_w)\|_2^2 \\ \text{subject to } & R_v^\top R_v = I_{2\times 2} \;\forall v \in V \\ & \vec{y}_v \text{ fixed } \forall v \in V_0. \end{aligned}$$

Solving for the matrices $R_v \in \mathbb{R}^{2\times 2}$ and vertex positions $\vec{y}_v \in \mathbb{R}^2$ simultaneously is a highly nonlinear and nonconvex task, especially given the orthogonality constraint $R_v^\top R_v = I_{2\times 2}$. There is one $\vec{y}_v$ and one $R_v$ for each vertex $v$ of a triangle mesh with potentially thousands or even millions of vertices, so such a direct optimization using quasi-Newton methods requires a large-scale linear solve per iteration and still is prone to finding local minima.

Instead, [116] suggests alternating between the following two steps:

1. Fixing the $R_v$ matrices and optimizing only for the positions $\vec{y}_v$:

$$\begin{aligned} \text{minimize}_{\vec{y}_v} & \sum_{v\in V} \sum_{(v,w)\in E} \|R_v(\vec{x}_v - \vec{x}_w) - (\vec{y}_v - \vec{y}_w)\|_2^2 \\ \text{subject to } & \vec{y}_v \text{ fixed } \forall v \in V_0. \end{aligned}$$

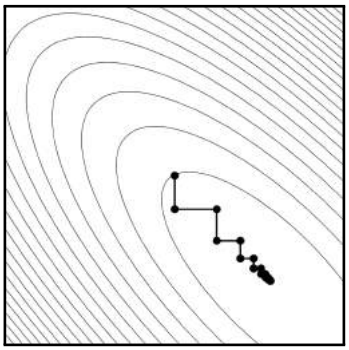

Figure 12.1 Coordinate descent in two dimensions alternates between minimizing in the horizontal and vertical axis directions.

This least-squares problem can be solved using a sparse, positive-definite linear system of equations.

2. Fixing the $\vec{y}_v$'s and optimizing for the $R_v$'s. No energy terms or constraints couple any pair $R_v, R_w$ for $v, w \in V$, so we can solve for each matrix $R_v$ independently. That is, rather than solving for $4|V|$ unknowns simultaneously, we loop over $v \in V$, solving the following optimization for each $R_v \in \mathbb{R}^{2\times 2}$:

$$\begin{aligned} &\text{minimize}_{R_v} \sum_{(v,w)\in E} \|R_v(\vec{x}_v - \vec{x}_w) - (\vec{y}_v - \vec{y}_w)\|_2^2 \\ &\text{subject to } R_v^\top R_v = I_{2\times 2}. \end{aligned}$$

This optimization problem is an instance of the Procrustes problem from §7.2.4 and can be solved in closed-form using a $2 \times 2$ SVD. We have replaced a large-scale minimization with the application of a formula that can be evaluated in parallel for each vertex, a massive computational savings.

Alternating between optimizing for the $\vec{y}_v$'s with the $R_v$'s fixed and vice versa decreases the energy using two efficient pieces of machinery, sparse linear solvers and $2 \times 2$ SVD factorization. This can be far more efficient than considering the $\vec{y}_v$'s and $R_v$'s simultaneously, and in practice a few iterations can be sufficient to generate elastic deformations like the one shown in Figure 10.3. Extensions of ARAP even run in real time, optimizing fast enough to provide interactive feedback to artists editing two- and three-dimensional shapes.

**Example 12.6** (Coordinate descent)**.** Taking the philosophy of alternating optimization to an extreme, rather than splitting the inputs of $f : \mathbb{R}^n \to \mathbb{R}$ into two variables, we could view $f$ as a function of several variables $f(x_1, x_2, \ldots, x_n)$. Then, we could cycle through each input $x_i$, performing a one-dimensional optimization in each step. This lightweight algorithm, illustrated in Figure 12.1, is known as *coordinate descent*.

For instance, suppose we wish to solve the least-squares problem $A\vec{x} \approx \vec{b}$ by minimizing $\|A\vec{x} - \vec{b}\|_2^2$. As in Chapter 11, line search over any single $x_i$ can be solved in closed form. If the columns of $A$ are vectors $\vec{a}_1, \ldots, \vec{a}_n$, then as shown in §1.3.1 we can write $A\vec{x} - \vec{b} =$

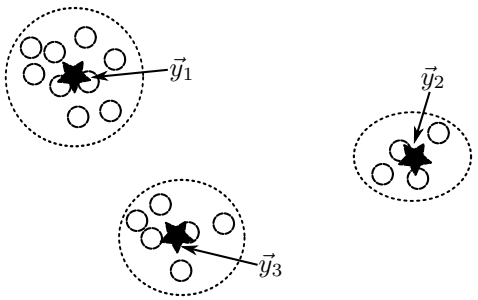


Figure 12.2 The $k$-means algorithm seeks cluster centers $\vec{y}_i$ that partition a set of data points $\vec{x}_1, \ldots, \vec{x}_m$ based on their closest center.

$x_1\vec{a}_1 + \cdots + x_n\vec{a}_n - \vec{b}$. By this expansion,

$$0 = \frac{\partial}{\partial x_i}\|x_1\vec{a}_1 + \cdots + x_n\vec{a}_n - \vec{b}\|_2^2 = 2(A\vec{x} - \vec{b}) \cdot \vec{a}_i = \sum_j \left[ \left( \sum_k a_{ji}a_{jk}x_k \right) - a_{ji}b_j \right].$$

Solving this equation for $x_i$ yields the following coordinate descent update for $x_i$:

$$x_i \leftarrow \frac{\vec{a}_i \cdot \vec{b} - \sum_{k \neq i} x_k(\vec{a}_i \cdot \vec{a}_k)}{\|\vec{a}_i\|_2^2}.$$

Coordinate descent for least-squares iterates this formula over $i = 1, 2, \ldots, n$ repeatedly until convergence. This approach has efficient localized updates and appears in machine learning methods where $A$ has many more rows than columns, sampled from a data distribution. We have traded a global method for one that locally updates the solution $\vec{x}$ by solving extremely simple subproblems.

**Example 12.7** ($k$-means clustering)**.** Suppose we are given a set of data points $\vec{x}_1, \ldots, \vec{x}_m \in \mathbb{R}^n$ and wish to group these points into $k$ clusters based on distance, as in Figure 12.2. Take $\vec{y}_1, \ldots, \vec{y}_k \in \mathbb{R}^n$ to be the centers of clusters $1, \ldots, k$, respectively. To cluster the data by assigning each point $\vec{x}_i$ to a single cluster centered at $\vec{y}_c$, the $k$-means technique optimizes the following energy:

$$E(\vec{y}_1, \ldots, \vec{y}_k) \equiv \sum_{i=1}^{m} \min_{c \in \{1,\ldots,k\}} \|\vec{x}_i - \vec{y}_c\|_2^2.$$

In words, $E$ measures the total squared distance of the data points $\vec{x}_i$ to their closest cluster center $\vec{y}_c$.

Define $c_i \equiv \arg\min_{c \in \{1,\ldots,k\}} \|\vec{x}_i - \vec{y}_c\|_2^2$; that is, $c_i$ is the index of the cluster center $\vec{y}_{c_i}$ closest to $\vec{x}_i$. Using this substitution, we can write an expanded formulation of the $k$-means objective as follows:

$$E(\vec{y}_1, \ldots, \vec{y}_k; c_1, \ldots, c_m) \equiv \sum_{i=1}^{m} \|\vec{x}_i - \vec{y}_{c_i}\|_2^2.$$

The variables $c_i$ are integers, but we can optimize them jointly with the $\vec{y}$'s using alternation:

- When the $c_i$'s are fixed, the optimization for the $\vec{y}_j$'s is a least-squares problem whose solution can be written in closed form as

$$\vec{y}_j = \frac{\sum_{c_i=j} \vec{x}_i}{|\{c_i = j\}|}.$$

  That is, $\vec{y}_j$ is the average of the points $\vec{x}_i$ assigned to cluster $j$.

- The optimization for $c_i$ also can be carried out in closed form using the expression $c_i \equiv \arg\min_{c \in \{1,\ldots,k\}} \|\vec{x}_i - \vec{y}_c\|_2^2$ by iterating from 1 to $k$ for each $i$. This iteration just assigns each $\vec{x}_i$ to its closest cluster center.

This alternation is known as the $k$-means algorithm and is a popular method for clustering. One drawback of this method is that it is sensitive to the initial guesses of $\vec{y}_1, \ldots, \vec{y}_k$. In practice, $k$-means is often run several times with different initial guesses, and only the best output is preserved. Alternatively, methods like "$k$-means++" specifically design initial guesses of the $\vec{y}_i$'s to encourage convergence to a better local minimum [3].

### 12.3.2 Augmented Lagrangians and ADMM

Nonlinear constrained problems are often the most challenging optimization tasks. While the general algorithms in §10.3 are applicable, they can be sensitive to the initial guess of the minimizer, slow to iterate due to large linear solves, and slow to converge in the absence of more information about the problems at hand. Using these methods is easy from an engineering perspective since they require providing only a function and its derivatives, but with some additional work on paper, certain objective functions can be tackled using faster techniques, many of which can be parallelized on multiprocessor machines. It is worth checking if a problem can be solved via one of these strategies, especially when the dimensionality is high or the objective has a number of similar or repeated terms.

In this section, we consider an alternating approach to equality-constrained optimization that has gained considerable attention in recent literature. While it can be used out-of-the-box as yet another generic optimization algorithm, its primary value appears to be in the decomposition of complex minimization problems into simpler steps that can be iterated, often in parallel. In large part we will follow the development of [14], which contains many examples of applications of this class of techniques.

As considered in Chapter 10, the equality-constrained optimization problem can be stated as follows:

$$\begin{aligned} &\text{minimize } f(\vec{x}) \\ &\text{subject to } g(\vec{x}) = \vec{0}. \end{aligned}$$

One incarnation of the barrier method suggested in §10.3.2 optimizes an unconstrained objective with a quadratic penalty:

$$f_\rho(\vec{x}) = f(\vec{x}) + \frac{1}{2}\rho\|g(\vec{x})\|_2^2.$$

As $\rho \to \infty$, critical points of $f_\rho$ satisfy the $g(\vec{x}) = \vec{0}$ constraint more and more closely. The trade-off for this method, however, is that the optimization becomes poorly conditioned as

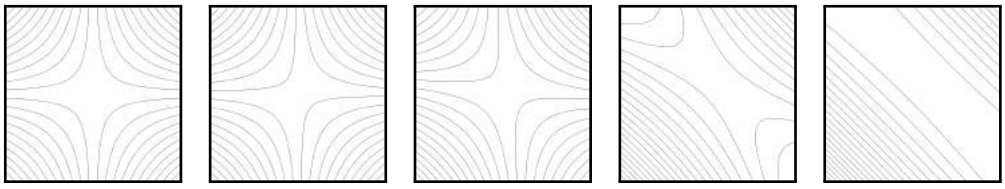


Figure 12.3 We can optimize $f(x, y) \equiv xy$ subject to $x + y = 1$ approximately by minimizing the penalized version $f_\rho(x, y) = xy + \rho(x + y - 1)^2$. As $\rho$ increases, however, level sets of $xy$ get obscured in favor of enforcing the constraint.

$\rho$ becomes large. This effect is illustrated in Figure 12.3; when $\rho$ is large, the level sets of $f_\rho$ mostly are dedicated to enforcing the constraint rather than minimizing the objective $f(\vec{x})$, making it difficult to distinguish between $\vec{x}$'s that all satisfy the constraint.

Alternatively, by the method of Lagrange multipliers (Theorem 1.1), we can seek first-order optima of this problem as the critical points of $\Lambda(\vec{x}, \vec{\lambda})$ given by

$$\Lambda(\vec{x}, \vec{\lambda}) \equiv f(\vec{x}) - \vec{\lambda}^\top g(\vec{x}).$$

This Lagrangian does not suffer from conditioning issues that affect the quadratic penalty method. On the other hand, it replaces a minimization problem—which can be solved by moving "downhill"—with a more challenging saddle point problem in which critical points should be minima of $\Lambda$ with respect to $\vec{x}$ and maxima of $\Lambda$ with respect to $\vec{\lambda}$. Optimizing by alternatively minimizing with respect to $\vec{x}$ and maximizing with respect to $\vec{\lambda}$ can be unstable; intuitively this makes some sense since it is unclear whether $\Lambda$ should be small or large.

The *augmented Lagrangian* method for equality-constrained optimization combines the quadratic penalty and Lagrangian strategies, using the penalty to "soften" individual iterations of the alternation for optimizing $\Lambda$ described above. It replaces the original equality-constrained optimization problem with the following equivalent augmented problem:

$$\begin{aligned} &\text{minimize } f(\vec{x}) + \frac{1}{2}\rho\|g(\vec{x})\|_2^2 \\ &\text{subject to } g(\vec{x}) = \vec{0}. \end{aligned}$$

Any $\vec{x}$ satisfying the $g(\vec{x}) = \vec{0}$ constraint makes the second objective term vanish. But, when the constraint is not exactly satisfied, the second energy term biases the objective toward points $\vec{x}$ that approximately satisfy the equality constraint. In other words, during iterations of augmented Lagrangian optimization, the $\rho\|g(\vec{x})\|_2^2$ acts like a rubber band pulling $\vec{x}$ closer to the constraint set even during the minimization step.

This modified problem has a new Lagrangian given by

$$\Lambda_\rho(\vec{x}, \vec{\lambda}) \equiv f(\vec{x}) + \frac{1}{2}\rho\|g(\vec{x})\|_2^2 - \vec{\lambda}^\top g(\vec{x}).$$

Hence, the augmented Lagrangian method optimizes this objective by alternating as follows:

**for** $i \leftarrow 1, 2, \ldots$
  $\vec{\lambda}_{i+1} \leftarrow \vec{\lambda}_i - \rho g(\vec{x}_i)$ ▷ Dual update
  $\vec{x}_i \leftarrow \min_{\vec{x}} \Lambda_\rho(\vec{x}, \vec{\lambda}_{i+1})$ ▷ Primal update

The dual update step can be thought of as a gradient ascent step for $\vec{\lambda}$. The parameter $\rho$ here no longer has to approach infinity for exact constraint satisfaction, since the Lagrange multiplier enforces the constraint regardless. Instead, the quadratic penalty serves to make sure the output of the $\vec{x}$ iteration does not violate the constraints too strongly.

Augmented Lagrangian optimization has the advantage that it alternates between applying a formula to update $\vec{\lambda}$ and solving an *unconstrained* minimization problem for $\vec{x}$. For many optimization problems, however, the unconstrained objective still may be nondifferentiable or difficult to optimize. A few special cases, e.g., Uzawa iteration for dual decomposition [124], can be effective for optimization but in many circumstances quasi-Newton algorithms outperform this approach with respect to speed and convergence.

A small alteration to general augmented Lagrangian minimization, however, yields the *alternating direction method of multipliers* (ADMM) for optimizing slightly more specific objectives of the form

$$\begin{aligned} &\text{minimize } f(\vec{x}) + h(\vec{z}) \\ &\text{subject to } A\vec{x} + B\vec{z} = \vec{c}. \end{aligned}$$

Here, the optimization variables are both $\vec{x}$ and $\vec{z}$, where $f, h : \mathbb{R}^n \to \mathbb{R}$ are given functions and the equality constraint is linear. As we will show, this form encapsulates many important optimization problems. We will design an algorithm that carries out alternation between the two primal variables $\vec{x}$ and $\vec{z}$, as well as between primal and dual optimization.

The augmented Lagrangian in this case is

$$\Lambda_\rho(\vec{x}, \vec{z}, \vec{\lambda}) \equiv f(\vec{x}) + h(\vec{z}) + \frac{1}{2}\rho\|A\vec{x} + B\vec{z} - \vec{c}\|_2^2 + \vec{\lambda}^\top(A\vec{x} + B\vec{z} - \vec{c}).$$

Alternating in three steps between optimizing $\vec{x}$, $\vec{z}$, and $\vec{\lambda}$ suggests a modification of the augmented Lagrangian method:

**for** $i \leftarrow 1, 2, \ldots$
  $\vec{x}_{i+1} \leftarrow \arg\min_{\vec{x}} \Lambda_\rho(\vec{x}, \vec{z}_i, \vec{\lambda}_i)$ ▷ $\vec{x}$ update
  $\vec{z}_{i+1} \leftarrow \arg\min_{\vec{z}} \Lambda_\rho(\vec{x}_{i+1}, \vec{z}, \vec{\lambda}_i)$ ▷ $\vec{z}$ update
  $\vec{\lambda}_{i+1} \leftarrow \vec{\lambda}_i + \rho(A\vec{x}_{i+1} + B\vec{z}_{i+1} - \vec{c})$ ▷ Dual update

In this algorithm, $\vec{x}$ and $\vec{z}$ are optimized one at a time; the augmented Lagrangian method would optimize them jointly. Although this splitting can require more iterations for convergence, clever choices of $\vec{x}$ and $\vec{z}$ lead to powerful division-of-labor strategies for breaking down difficult problems. Each individual iteration will take *far* less time, even though more iterations may be needed for convergence. In a sense, ADMM is a "meta-algorithm" used to design optimization techniques. Rather than calling a generic package to minimize $\Lambda_\rho$ with respect to $\vec{x}$ and $\vec{z}$, we will find choices of $\vec{x}$ and $\vec{z}$ that make individual steps fast.

Before working out examples of ADMM in action, it is worth noting that it is guaranteed to converge to a critical point of the objective under fairly weak conditions. For instance, ADMM reaches a global minimum when $f$ and $h$ are convex and $\Lambda_\rho$ has a saddle point. ADMM has also been observed to converge even for nonconvex problems, although current theoretical understanding in this case is limited. In practice, ADMM tends to be quick to generate *approximate* minima of the objective but can require a long tail of iterations to squeeze out the last decimal points of accuracy; for this reason, some systems use ADMM to do initial large-scale steps and transition to other algorithms for localized optimization.

We dedicate the remainder of this section to working out examples of ADMM in practice. The general pattern is to split the optimization variables into $\vec{x}$ and $\vec{z}$ in such a way that

the two primal update steps each can be carried out efficiently, preferably in closed form or decoupling so that parallelized computations can be used to solve many subproblems at once. This makes individual iterations of ADMM inexpensive.

**Example 12.8** (Nonnegative least-squares)**.** Suppose we wish to minimize $\|A\vec{x}-\vec{b}\|_2^2$ with respect to $\vec{x}$ subject to the constraint $\vec{x} \geq \vec{0}$. The $\vec{x} \geq 0$ constraint rules out using Gaussian elimination, but ADMM provides one way to bypass this issue.

Consider solving the following equivalent problem:

$$\begin{aligned} &\text{minimize } \|A\vec{x} - \vec{b}\|_2^2 + h(\vec{z}) \\ &\text{subject to } \vec{x} = \vec{z}. \end{aligned}$$

Here, we define the new function $h(\vec{z})$ as follows:

$$h(\vec{z}) = \begin{cases} 0 & \vec{z} \geq \vec{0} \\ \infty & \text{otherwise.} \end{cases}$$

The function $h(\vec{z})$ is discontinuous, but it is convex. This equivalent form of nonnegative least-squares may be harder to read, but it provides an effective ADMM splitting.

For this optimization, the augmented Lagrangian is

$$\Lambda_\rho(\vec{x}, \vec{z}, \vec{\lambda}) = \|A\vec{x} - \vec{b}\|_2^2 + h(\vec{z}) + \frac{1}{2}\rho\|\vec{x} - \vec{z}\|_2^2 + \vec{\lambda}^\top(\vec{x} - \vec{z}).$$

For fixed $\vec{z}$ with $z_i \neq \infty$ for all $i$, then $\Lambda_\rho$ is differentiable with respect to $\vec{x}$. Hence, we can carry out the $\vec{x}$ step of ADMM by setting the gradient with respect to $\vec{x}$ equal to $\vec{0}$:

$$\begin{aligned} \vec{0} &= \nabla_{\vec{x}}\Lambda_\rho(\vec{x}, \vec{z}, \vec{\lambda}) \\ &= 2A^\top A\vec{x} - 2A^\top\vec{b} + \rho(\vec{x} - \vec{z}) + \vec{\lambda} \\ &= (2A^\top A + \rho I_{n\times n})\vec{x} + (\vec{\lambda} - 2A^\top\vec{b} - \rho\vec{z}) \\ \implies \vec{x} &= (2A^\top A + \rho I_{n\times n})^{-1}(2A^\top\vec{b} + \rho\vec{z} - \vec{\lambda}). \end{aligned}$$

This linear solve is a Tikhonov-regularized least-squares problem. For extra speed, the Cholesky factorization of $2A^\top A + \rho I_{n\times n}$ can be computed before commencing ADMM and used to find $\vec{x}$ in each iteration.

Minimizing $\Lambda_\rho$ with respect to $\vec{z}$ can be carried out in closed form. Any objective function involving $h$ effectively constrains each component of $\vec{z}$ to be nonnegative, so we can find $\vec{z}$ using the following optimization:

$$\begin{aligned} &\text{minimize}_{\vec{z}} \ \frac{1}{2}\rho\|\vec{x} - \vec{z}\|_2^2 + \vec{\lambda}^\top(\vec{x} - \vec{z}) \\ &\text{subject to } \vec{z} \geq \vec{0}. \end{aligned}$$

The $\|A\vec{x} - \vec{b}\|_2^2$ term in the full objective is removed because it has no $\vec{z}$ dependence. This problem *decouples* over the components of $\vec{z}$ since no energy terms involve more than one dimension of $\vec{z}$ at a time. So, we can solve many instances of the following one-dimensional problem:

$$\begin{aligned} &\text{minimize}_{z_i} \ \frac{1}{2}\rho(x_i - z_i)^2 + \lambda_i(x_i - z_i) \\ &\text{subject to } z_i \geq 0. \end{aligned}$$

In the absence of the $z_i \geq 0$ constraint, the objective is minimized when $0 = \rho(z_i - x_i) - \lambda_i \implies z_i = x_i + {}^{\lambda_i}/_{\rho}$; when this value is negative, we fix $z_i = 0$.

Hence, the ADMM algorithm for nonnegative least-squares is:

**for** $i \leftarrow 1, 2, \ldots$
  $\vec{x}_{i+1} \leftarrow (2A^\top A + \rho I_{n\times n})^{-1}(2A^\top \vec{b} + \rho \vec{z}_i - \vec{\lambda}_i)$ ▷ $\vec{x}$ update; least-squares
  $\vec{z}^0 \leftarrow \vec{\lambda}_i/\rho + \vec{x}_{i+1}$ ▷ Unconstrained $\vec{z}$ formula
  $\vec{z}_{i+1} \leftarrow \textsc{Elementwise-Max}(\vec{z}^0, \vec{0})$ ▷ Enforce $\vec{z} \geq \vec{0}$
  $\vec{\lambda}_{i+1} \leftarrow \vec{\lambda}_i + \rho(\vec{x}_{i+1} - \vec{z}_{i+1})$ ▷ Dual update

This algorithm for nonnegative least-squares took our original problem—a quadratic program that could require difficult constrained optimization techniques—and replaced it with an alternation between a linear solve for $\vec{x}$, a formula for $\vec{z}$, and a formula for $\vec{\lambda}$. These individual steps are straightforward to implement and efficient computationally.

**Example 12.9** (ADMM for geometric median)**.** Returning to Example 12.3, we can reconsider the energy $E(\vec{x})$ for the geometric median problem using the machinery of ADMM:

$$E(\vec{x}) \equiv \sum_{i=1}^{N} \|\vec{x} - \vec{x}_i\|_2.$$

This time, we will split the problem into two unknowns $\vec{z}_i, \vec{x}$:

$$\begin{aligned} &\text{minimize} \sum_i \|\vec{z}_i\|_2 \\ &\text{subject to } \vec{z}_i + \vec{x} = \vec{x}_i \ \forall i. \end{aligned}$$

The augmented Lagrangian for this problem is:

$$\Lambda_\rho = \sum_i \left[ \|\vec{z}_i\|_2 + \frac{1}{2}\rho\|\vec{z}_i + \vec{x} - \vec{x}_i\|_2^2 + \vec{\lambda}_i^\top(\vec{z}_i + \vec{x} - \vec{x}_i) \right].$$

As a function of $\vec{x}$, the augmented Lagrangian is differentiable and hence to find the $\vec{x}$ iteration we write:

$$\begin{aligned} \vec{0} = \nabla_{\vec{x}}\Lambda_\rho &= \sum_i \left[ \rho(\vec{x} - \vec{x}_i + \vec{z}_i) + \vec{\lambda}_i \right] \\ \implies \vec{x} &= \frac{1}{N}\sum_i \left[ \vec{x}_i - \vec{z}_i - \frac{1}{\rho}\vec{\lambda}_i \right]. \end{aligned}$$

The optimization for the $\vec{z}_i$'s decouples over $i$ when $\vec{x}$ is fixed, so after removing constant terms we minimize $\|\vec{z}_i\|_2 + \frac{1}{2}\rho\|\vec{z}_i + \vec{x} - \vec{x}_i\|_2^2 + \vec{\lambda}_i^\top \vec{z}_i$ for each $\vec{z}_i$ separately. We can combine the second and third terms by "completing the square" as follows:

$$\begin{aligned} \frac{1}{2}\rho\|\vec{z}_i + \vec{x} - \vec{x}_i\|_2^2 + \vec{\lambda}_i^\top \vec{z}_i &= \frac{1}{2}\rho\|\vec{z}_i\|_2^2 + \rho\vec{z}_i^\top\left(\frac{1}{\rho}\vec{\lambda}_i + \vec{x} - \vec{x}_i\right) + \text{const.} \\ &= \frac{1}{2}\rho\left\|\vec{z}_i + \frac{1}{\rho}\vec{\lambda}_i + \vec{x} - \vec{x}_i\right\|_2^2 + \text{const.} \end{aligned}$$

The constant terms can have $\vec{x}$ dependence since it is fixed in the $\vec{z}_i$ iteration. Defining $\vec{z}^0 \equiv -\frac{1}{\rho}\vec{\lambda}_i - \vec{x} + \vec{x}_i$, in the $\vec{z}_i$ iteration we have shown that we can solve:

$$\min_{\vec{z}_i}\left[ \|\vec{z}_i\|_2 + \frac{1}{2}\rho\|\vec{z}_i - \vec{z}^0\|_2^2 \right].$$

Written in this form, it is clear that the optimal $\vec{z}_i$ satisfies $\vec{z}_i = t\vec{z}^0$ for some $t \in [0, 1]$, since the two terms of the objective balance the distance of $\vec{z}_i$ to $\vec{0}$ and to $\vec{z}^0$. After dividing by $\|\vec{z}^0\|_2$, we can solve:

$$\min_{t \geq 0} \left[ t + \frac{1}{2}\rho\|\vec{z}^0\|_2(t-1)^2 \right].$$

Using elementary calculus techniques we find:

$$t = \begin{cases} 1 - 1/\rho\|\vec{z}^0\|_2 & \text{when } \rho\|\vec{z}^0\|_2 \geq 1 \\ 0 & \text{otherwise.} \end{cases}$$

Taking $\vec{z}_i = t\vec{z}_0$ finishes the $\vec{z}$ iteration of ADMM.

In summary, the ADMM algorithm for geometric medians is as follows:

**for** $i \leftarrow 1, 2, \ldots$
  $\vec{x} \leftarrow \frac{1}{N}\sum_i \left[\vec{x}_i - \vec{z}_i - \frac{1}{\rho}\vec{\lambda}_i\right]$ ▷ $\vec{x}$ update
  **for** $j \leftarrow 1, 2, \ldots, N$ ▷ Can parallelize
    $\vec{z}^0 \leftarrow -\frac{1}{\rho}\vec{\lambda}_i - \vec{x} + \vec{x}_i$
    $t \leftarrow \begin{cases} 1 - 1/\rho\|\vec{z}^0\|_2 & \text{when } \rho\|\vec{z}^0\|_2 \geq 1 \\ 0 & \text{otherwise} \end{cases}$
    $\vec{z}_j \leftarrow t\vec{z}^0$ ▷ $\vec{z}$ update
    $\vec{\lambda}_j \leftarrow \vec{\lambda}_j + \rho(\vec{z}_i + \vec{x} - \vec{x}_i)$ ▷ Dual update

The examples above show the typical ADMM strategy, in which a difficult nonlinear problem is split into two subproblems that can be carried out in closed form or via more efficient operations. The art of posing a problem in terms of $\vec{x}$ and $\vec{z}$ to get these savings requires practice and careful study of individual problems.

The parameter $\rho > 0$ often does not affect whether or not ADMM will eventually converge, but an intelligent choice of $\rho$ can help this technique reach the optimal point faster. Some experimentation can be required, or $\rho$ can be adjusted from iteration to iteration depending on whether the primal or dual variables are converging more quickly [127]. In some cases, ADMM provably converges faster when $\rho \to \infty$ as the iterations proceed [104].

## 12.4 GLOBAL OPTIMIZATION

Nonlinear least-squares, IRLS, and alternation are lightweight approaches for nonlinear objectives that can be optimized quickly after simplification. On the other side of the spectrum, some minimization problems not only do not readily admit fast specialized algorithms but also are failure modes for Newton's method and other generic solvers. Convergence guarantees for Newton's method and other algorithms based on the Taylor approximation assume that we have a strong initial guess of the minimum that we wish to refine. When we lack such an initial guess or a simplifying assumption like convexity, we must solve a *global optimization* problem searching over the entire space of feasible output.

As discussed briefly in §9.2, global optimization is a challenging, nearly ill-posed problem. For example, in the unconstrained case it is difficult to know whether $\vec{x}^*$ yields the minimum possible $f(\vec{x})$ *anywhere*, since this is a statement over an infinitude of points $\vec{x}$. Hence, global optimization methods use one or more strategies to improve the odds of finding a minimum:

- Initially approximate the objective $f(\vec{x})$ with an easier function to minimize to get a better starting point for the original problem.

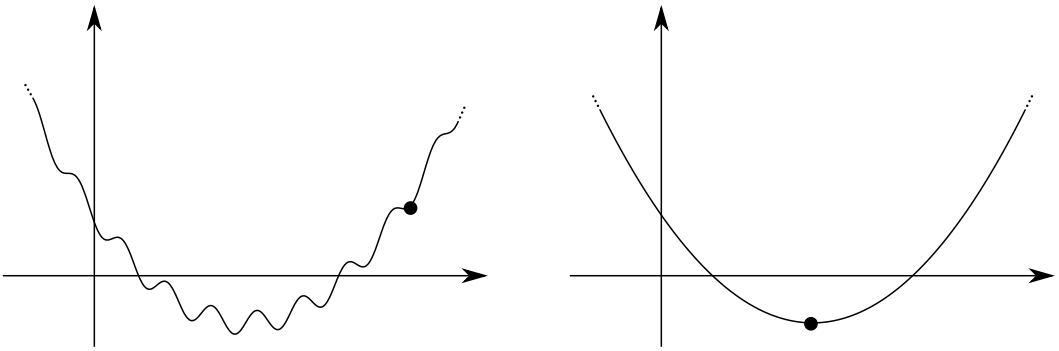

Figure 12.4 Newton's method can get caught in any number of local minima in the function on the left; smoothing this function, however, can generate a stronger initial guess of the global optimum.

- Sample the space of possible inputs $\vec{x}$ to get a better idea of the behavior of $f$ over a large domain.

These and other strategies are *heuristic*, meaning that they usually cannot be used to guarantee that the output of such a minimization is globally optimal. In this section, we mention a few common techniques for global optimization as pointers to more specialized literature.

### 12.4.1 Graduated Optimization

Consider the optimization objective illustrated in Figure 12.4. Locally, this objective wiggles up and down, but at a larger scale, a more global pattern emerges. Newton's method seeks *any* critical point of $f(x)$ and easily can get caught in one of its local minima. To avoid this suboptimal output, we might attempt to minimize a smoothed version of $f(x)$ to generate an initial guess for the minimum of the more involved optimization problem.

*Graduated optimization* techniques solve progressively harder optimization problems with the hope that the coarse initial iterations will generate better initial guesses for the more accurate but sensitive later steps. In particular, suppose we wish to minimize some function $f(\vec{x})$ over $\vec{x} \in \mathbb{R}^n$ with many local optima as in Figure 12.4. Graduated methods generate a sequence of functions $f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x})$ with $f_k(\vec{x}) = f(\vec{x})$, using critical points of $f_i$ as initial guesses for minima of $f_{i+1}$.

**Example 12.10** (Image alignment)**.** A common task making use of graduated optimization is photograph alignment as introduced in §4.1.4. Consider the images in Figure 12.5. Aligning the original two images can be challenging because they have lots of high-frequency detail; for instance, the stones on the wall all look similar and easily could be misidentified. By *blurring* the input images, a better initial guess of the alignment can be obtained, because high-frequency details are suppressed.

The art of graduated optimization lies in finding an appropriate sequence of $f_i$'s to help reach a global optimum. In signal and image processing, like in Example 12.10, a typical approach is to use the same optimization objective in each iteration but blur the underlying data to reveal larger-scale patterns. *Scale space* methods like [81] blur the objective itself,

Original Blurred

Figure 12.5 The photos on the left can be hard to align using automatic methods because they have lots of high-frequency detail that can obscure larger alignment patterns; by blurring the photos we can align larger features before refining the alignment using texture and other detail.

for instance by defining $f_i$ to be $f(\vec{x}) * g_{\sigma_i}(\vec{x})$, the result of blurring $f(\vec{x})$ using a Gaussian of width $\sigma_i$, with $\sigma_i \to 0$ as $i \to \infty$.

A related set of algorithms known as *homotopy continuation methods* continuously changes the optimization objective by leveraging intuition from topology. These algorithms make use of the following notion from classical mathematics:

**Definition 12.1** (Homotopic functions)**.** Two continuous functions $f(\vec{x})$ and $g(\vec{x})$ are *homotopic* if there exists continuous function $H(\vec{x}, s)$ with

$$H(\vec{x}, 0) = f(\vec{x}) \quad \text{and} \quad H(\vec{x}, 1) = g(\vec{x})$$

for all $\vec{x}$.

The idea of homotopy is illustrated in Figure 12.6.

Similar to graduated methods, homotopy optimizations minimize $f(\vec{x})$ by defining a new function $H(\vec{x}, s)$ where $H(\vec{x}, 0)$ is easy to optimize and $H(\vec{x}, 1) = f(\vec{x})$. Taking $\vec{x}_0^*$ to be the minimum of $H(\vec{x}, 0)$ with respect to $\vec{x}$, basic homotopy methods incrementally increase $s$, each time updating to a new $\vec{x}_s^*$. Assuming $H$ is continuous, we expect the minimum $\vec{x}_s^*$ to trace a continuous path in $\mathbb{R}^n$ as $s$ increases; hence, the solve for each $\vec{x}_s^*$ after increasing $s$ differentially has a strong initial guess from the previous iteration.

**Example 12.11** (Homotopy methods, [45])**.** Homotopy methods also apply to root-finding. As a small example, suppose we wish to find points $x$ satisfying $\arctan(x) = 0$. Applying the formula from §8.1.4, Newton's method for finding such a root iterates

$$x_{k+1} = x_k - (1 + x_k^2)\arctan(x).$$

If we provide an initial guess $x_0 = 4$, however, this iteration diverges. Instead, we can define a homotopy function as

$$H(x, s) \equiv \arctan(x) + (s - 1)\arctan(4).$$

We know $H(x, 0) = \arctan(x) - \arctan(4)$ has a root at the initial guess $x_0 = 4$. Stepping $s$ by increments of $1/10$ from 0 to 1, each time minimizing $H(x, s_i)$ with initial guess $x_{i-1}^*$ via Newton's method yields a sequence of convergent problems reaching $x^* = 0$.

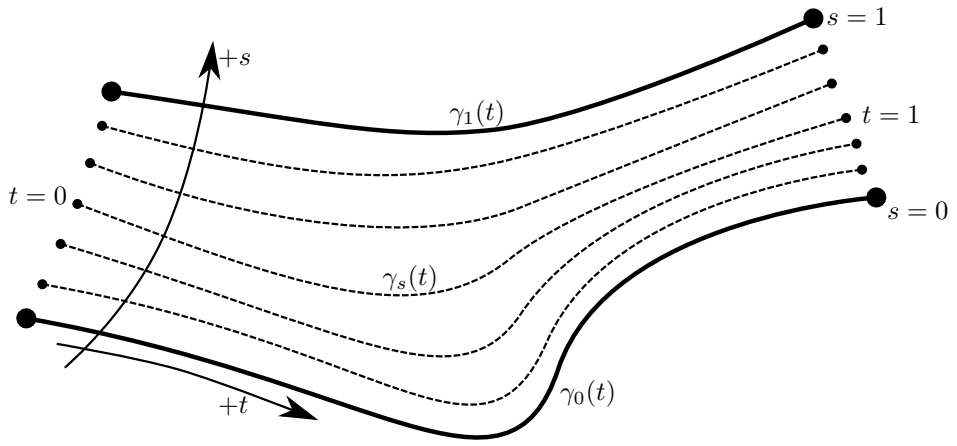


Figure 12.6 The curves $\gamma_0(t)$ and $\gamma_1(t)$ are homotopic because there exists a continuously varying set of curves $\gamma_s(t)$ for $s \in [0, 1]$ coinciding with $\gamma_0$ at $s = 0$ and $\gamma_1$ at $s = 1$.

More generally, we can think of a *solution path* as a curve of points $(\vec{x}(t), s(t))$ such that $s(0) = 0$, $s(1) = 1$, and at each time $t$, $\vec{x}(t)$ is a local minimizer of $H(\vec{x}, s(t))$ over $\vec{x}$. Our initial description of homotopy optimization would take $s(t) = t$, but now we can allow $s(t)$ to be non-monotonic as a function of $t$ as long as it eventually reaches $s = 1$. Advanced homotopy continuation methods view $(\vec{x}(t), s(t))$ as a curve satisfying certain ordinary differential equations, which you will derive in Exercise 12.6; these equations can be solved using the techniques we will define in Chapter 15.

### 12.4.2 Randomized Global Optimization

When smoothing the objective function is impractical or fails to remove local minima from $f(\vec{x})$, it makes sense to sample the space of possible inputs $\vec{x}$ to get some idea of the energy landscape. Newton's method, gradient descent, and others all have strong dependence on the initial guess of the location of the minimum, so trying more than one starting point increases the chances of success.

If the objective $f$ is sufficiently noisy, we may wish to remove dependence on differential estimates altogether. Without gradients, we do not know which directions locally point downhill, but via sampling we can find such patterns on a larger scale. Heuristics for global optimization at this scale commonly draw inspiration from the natural world and the idea of *swarm intelligence*, that complex natural processes can arise from individual actors following simple rules, often in the presence of stochasticity, or randomness. For instance, optimization routines have been designed to mimic ant colonies transporting food [26], thermodynamic energy in "annealing" processes [73], and evolution of DNA and genetic material [87]. These methods usually are considered heuristics without convergence guarantees but can help guide a large-scale search for optima.

As one example of a method well-tuned to continuous problems, we consider the *particle swarm* method introduced in [72] as an optimization technique inspired by social behavior in bird flocks and fish schools. Many variations of this technique have been proposed, but we explore one of the original versions introduced in [36].

Suppose we have a set of candidate minima $\vec{x}_1, \ldots, \vec{x}_k$. We will think of these points as particles moving around the possible space of $\vec{x}$ values, and hence they will also be assigned velocities $\vec{v}_1, \ldots, \vec{v}_k$. The particle swarm method maintains a few additional variables:

- $\vec{p}_1, \ldots, \vec{p}_k$, the position over all iterations so far of the lowest value $f(\vec{p}_i)$ observed by each particle $i$.
- The position $\vec{g} \in \{\vec{p}_1, \ldots, \vec{p}_k\}$ with the smallest objective value; this position is the *globally* best solution observed so far.

This notation is illustrated in Figure 12.7.

In each iteration of particle swarm optimization, the velocities of the particles are updated to guide them toward likely minima. Each particle is attracted to its own best observed minimum as well as to the global best position so far:

$$\vec{v}_i \leftarrow \vec{v}_i + \alpha(\vec{p}_i - \vec{x}_i) + \beta(\vec{g} - \vec{x}_i).$$

The parameters $\alpha, \beta \geq 0$ determine the amount of force felt from $\vec{x}_i$ to move toward these two positions; larger $\alpha, \beta$ values will push particles toward minima faster at the cost of more limited exploration of the space of possible minima. Once velocities have been updated, the particles move along their velocity vectors:

$$\vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i.$$

Then, the process repeats. This algorithm is not guaranteed to converge, but it can be terminated at any point, with $\vec{g}$ as the best observed minimum. The final method is documented in Figure 12.8.

## 12.5 ONLINE OPTIMIZATION

We briefly consider a class of optimization problems from machine learning, game theory, and related fields in which the objective itself is allowed to change from iteration to iteration. These problems, known as *online optimization* problems, reflect a world in which evolving input parameters, priorities, and desired outcomes can make the output of an optimization irrelevant soon after it is generated. Hence, techniques in this domain must adaptively react to the changing objective in the presence of noise. Our discussion will introduce a few basic ideas from [107]; we refer the reader to that survey article for a more detailed treatment.

**Example 12.12** (Stock market)**.** Suppose we run a financial institution and wish to maintain an optimal portfolio of investments. On the morning of day $t$, in a highly simplified model we might choose how much of each stock $1, \ldots, n$ to buy, represented by a vector $\vec{x}_t \in (\mathbb{R}^+)^n$. At the end of the day, based on fluctuations of the market, we will know a function $f_t$ so that $f_t(\vec{x})$ gives us our total profit or loss based on the decision $\vec{x}$ made in the morning. The function $f_t$ can be different every day, so we must attempt to design a policy that predicts the objective function and/or its optimal point every day.

Problems in this class often can be formalized as *online convex optimization* problems. In the unconstrained case, online convex optimization algorithms are designed for the following feedback loop:

**for** $t = 1, 2, \ldots$ ▷ At each time $t$
  ▷ Predict $\vec{x}_t \in U$
  ▷ Receive loss function $f_t : U \to \mathbb{R}$
  ▷ Suffer loss $f_t(\vec{x}_t)$

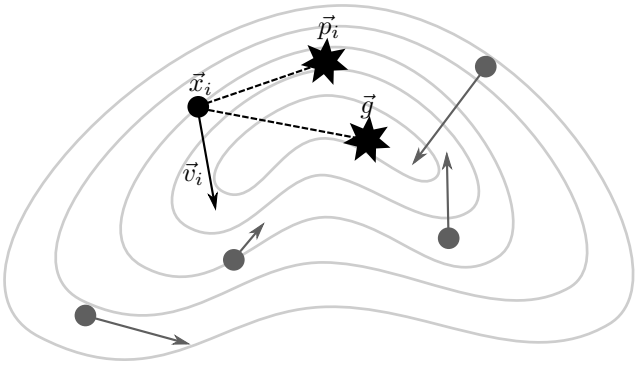


Figure 12.7 The particle swarm navigates the landscape of $f(\vec{x})$ by maintaining positions and velocities for a set of potential minima $\vec{x}_i$; each $\vec{x}_i$ is attracted to the position $\vec{p}_i$ at which it has observed the smallest value of $f(\vec{x}_i)$ as well as to the minimum $\vec{g}$ observed thus far by any particle.

```
function PARTICLE-SWARM(f(x⃗), k, α, β, x⃗_min, x⃗_max, v⃗_min, v⃗_max)
  f_min ← ∞
  for i ← 1, 2, ..., k
    x⃗_i ← RANDOM-POSITION(x⃗_min, x⃗_max)        ▷ Initialize positions randomly
    v⃗_i ← RANDOM-VELOCITY(v⃗_min, v⃗_max)        ▷ Initialize velocities randomly
    f_i ← f(x⃗_i)                                ▷ Evaluate f
    p⃗_i ← x⃗_i                                   ▷ Current particle optimum
    if f_i < f_min then                          ▷ Check if it is global optimum
      f_min ← f_i                                ▷ Update optimal value
      g⃗ ← x⃗_i                                   ▷ Set global optimum
  for j ← 1, 2, ...                              ▷ Stop when satisfied with g⃗
    for i ← 1, 2, ..., k
      v⃗_i ← v⃗_i + α(p⃗_i − x⃗_i) + β(g⃗ − x⃗_i)     ▷ Update velocity
      x⃗_i ← x⃗_i + v⃗_i                           ▷ Update position
    for i ← 1, 2, ..., k
      if f(x⃗_i) < f_i then                       ▷ Better minimum for particle i
        p⃗_i ← x⃗_i                               ▷ Update particle optimum
        f_i ← f(x⃗_i)                            ▷ Store objective value
        if f_i < f_min then                      ▷ Check if it is a global optimum
          f_min ← f_i                            ▷ Update optimal value
          g⃗ ← x⃗_i                               ▷ Global optimum
```

Figure 12.8 Particle swarm optimization attempts to minimize $f(\vec{x})$ by simulating a collection of particles $\vec{x}_1, \ldots, \vec{x}_k$ moving in the space of potential inputs $\vec{x}$.

We will assume the $f_t$'s are convex and that $U \subseteq \mathbb{R}^n$ is a convex set. There are a few features of this setup worth highlighting:

- To stay consistent with our discussion of optimization in previous chapters, we phrase the problem as *minimizing loss* rather than, e.g., maximizing profit.
- The optimization objective can change at each time $t$, and we do not get to know the objective $f_t$ before choosing $\vec{x}_t$. In the stock market example, this feature reflects the fact that we do not know the price of a stock on day $t$ until the day is over, and we must decide how much to buy before getting to that point.
- The online convex optimization algorithm can choose to store $f_1, \ldots, f_{t-1}$ to inform its choice of $\vec{x}_t$. For stock investment, we can use the stock prices on previous days to predict them for the future.

Since online convex optimization algorithms do not know $f_t$ before predicting $\vec{x}_t$, we cannot expect them to perform perfectly. An "adversarial" client might wait for $\vec{x}_t$ and purposefully choose a loss function $f_t$ to make $\vec{x}_t$ look bad! For this reason, metrics like *cumulative loss* $\sum_{t=1}^{T} f_t(\vec{x}_t)$ are unfair measures for the quality of an online optimization method at time $T$. In some sense, we must lower our standards for success.

One model for online convex optimization is minimization of *regret*, which compares performance to that of a fixed expert benefiting from hindsight:

**Definition 12.2** (Regret)**.** The *regret* of an online optimization algorithm at time $T$ over a set $U$ is given by

$$R_T \equiv \max_{\vec{u} \in U} \left[ \sum_{t=1}^{T} (f_t(\vec{x}_t) - f_t(\vec{u})) \right].$$

The regret $R_T$ measures the difference between how well our algorithm has performed over time—as measured by summing $f_t(\vec{x}_t)$ over $t$—and the performance of any constant point $\vec{u}$ that must remain the same over all $t$. For the stock example, regret compares the profits lost by using our algorithm and the loss of using any single stock portfolio over all time. Ideally, the ratio $R_T/T$ measuring average regret over time should decrease as $T \to \infty$.

The most obvious approach to online optimization is the "follow the leader" (FTL) strategy, which chooses $\vec{x}_t$ based on how it would have performed at times $1, \ldots, t-1$ :

$$\textbf{Follow the leader: } \vec{x}_t \equiv \arg\min_{\vec{x} \in U} \sum_{s=1}^{t-1} f_s(\vec{x}).$$

FTL is a reasonable heuristic if we assume past performance has some bearing on future results. After all, if we do not know $f_t$ we might as well hope that it is similar to the objectives $f_1, \ldots, f_{t-1}$ we have observed in the past.

For many classes of functions $f_t$, FTL is an effective approach that makes increasingly well-informed choices of $\vec{x}_t$ as $t$ progresses. It can experience some serious drawbacks, however, as illustrated in the following example:

**Example 12.13** (Failure of FTL, [107] §2.2)**.** Suppose $U = [0, 1]$ and we generate a sequence of functions as follows:

$$f_t(x) = \begin{cases} -x/2 & \text{if } t = 1 \\ x & \text{if } t \text{ is even} \\ -x & \text{otherwise.} \end{cases}$$

FTL minimizes the sum over all previous objective functions, giving the following series of outputs:

$$\begin{aligned}
\mathbf{t = 1}: &\quad x \text{ arbitrary } \in [0,1] \\
\mathbf{t = 2}: &\quad x_2 = \arg\min_{x\in[0,1]} -x/2 = 1 \\
\mathbf{t = 3}: &\quad x_3 = \arg\min_{x\in[0,1]} x/2 = 0 \\
\mathbf{t = 4}: &\quad x_4 = \arg\min_{x\in[0,1]} -x/2 = 1 \\
\mathbf{t = 5}: &\quad x_5 = \arg\min_{x\in[0,1]} x/2 = 0 \\
\vdots &\qquad\quad \vdots
\end{aligned}$$

From the above calculation, we find that in every iteration except $t = 1$, FTL incurs loss 1, while fixing $x = 0$ for all time would incur zero loss. For this example, FTL has regret growing proportionally to $t$.

This example illustrates the type of analysis and reasoning typically needed to design online learning methods. To bound regret, we must consider the *worst* possible adversary, who generates functions $f_t$ specifically designed to take advantage of the weaknesses of a given technique.

FTL failed because it was too strongly sensitive to the fluctuations of $f_t$ from iteration to iteration. To resolve this issue, we can take inspiration from Tikhonov regularization (§4.1.3), $L^1$ regularization (§10.4.1), and other methods that dampen the output of numerical methods by adding an energy term punishing irregular or large output vectors. To do so, we define the "follow the regularized leader" (FTRL) strategy:

$$\textbf{Follow the regularized leader: } \vec{x}_t \equiv \arg\min_{\vec{x}\in U} \left[ r(\vec{x}) + \sum_{s=1}^{t-1} f_s(\vec{x}) \right].$$

Here, $r(\vec{x})$ is a convex regularization function, such as $\|\vec{x}\|_2^2$ (Tikhonov regularization), $\|\vec{x}\|_1$ ($L^1$ regularization), or $\sum_i x_i \log x_i$ when $U$ includes only $\vec{x} \geq \vec{0}$ (entropic regularization).

Just as regularization improves the conditioning of a linear problem when it is close to singular, in this case the change from FTL to FTRL avoids fluctuation issues illustrated in Example 12.13. For instance, suppose $r(\vec{x})$ is *strongly convex* as defined below for differentiable $r$:

**Definition 12.3** (Strongly convex)**.** A differentiable regularizer $r(\vec{x})$ is *$\sigma$-strongly convex* with respect to a norm $\|\cdot\|$ if for any $\vec{x}, \vec{y}$ the following relationship holds:

$$(\nabla r(\vec{x}) - \nabla r(\vec{y})) \cdot (\vec{x} - \vec{y}) \geq \sigma \|\vec{x} - \vec{y}\|_2^2.$$

Intuitively, a strongly convex regularizer not only is bowl-shaped but has a lower bound for the curvature of that bowl. Then, we can prove the following statement:

**Proposition 12.1** ([107], Theorem 2.11)**.** Assume $r(\vec{x})$ is $\sigma$-strongly convex and that each $f_t$ is convex and $L$-Lipschitz (see §8.1.1). Then, the regret is bounded as follows:

$$R_T \leq \left[ \max_{\vec{u}\in U} r(\vec{u}) \right] - \left[ \min_{\vec{v}\in U} r(\vec{v}) \right] + \frac{TL^2}{\sigma}.$$

The proof of this proposition uses techniques well within the scope of this book but due to its length is omitted from our discussion.

Proposition 12.1 can be somewhat hard to interpret, but it is a strong result about the effectiveness of the FTRL technique given an appropriate choice of $r$. In particular, the max

and min terms as well as $\sigma$ are properties of $r(\vec{x})$ that should guide which regularizer to use for a particular problem. The value $\sigma$ contributes to both terms in competing ways:

- The difference between the maximum and minimum values of $r$ is its range of possible outputs. Increasing $\sigma$ has the potential to increase this difference, since it is bounded below by a "steeper" bowl. So, minimizing this term in our regret bound prefers *small* $\sigma$.
- Minimizing $TL^2/\sigma$ prefers *large* $\sigma$.

Practically speaking, we can decide what range of $T$ we care about and choose a regularizer accordingly:

**Example 12.14** (FTRL choice of regularizers)**.** Consider the regularizer $r_\sigma(\vec{x}) \equiv \frac{1}{2}\sigma\|\vec{x}\|_2^2$. It has gradient $\nabla r_\sigma(\vec{x}) = \sigma\vec{x}$, so by direct application of Definition 12.3, it is $\sigma$-strongly convex. Suppose $U = \{\vec{x} \in \mathbb{R}^n : \|\vec{x}\|_2 \leq 1\}$ and that we expect to run our optimization for $T$ time steps. If we take $\sigma = \sqrt{T}$, then the regret bound from Proposition 12.1 shows:

$$R_T \leq (1 + L^2)\sqrt{T}.$$

For large $T$, this value is small relative to $T$, compared to the linear growth for FTL in Example 12.13.

Online optimization is a rich area of research that continues to be explored. Beyond FTRL, we can define algorithms with better or more usable regret bounds, especially if we know more about the class of functions $f_t$ we expect to observe. FTRL also has the drawback that it has to solve a potentially complex optimization problem at each iteration, which may not be practical for systems that have to make decisions quickly. Surprisingly, even easy-to-solve linearizations can behave fairly well for convex objectives, as illustrated in Exercise 12.14. Popular online optimization techniques like [34] have been applied to a variety of learning problems in the presence of huge amounts of noisy data.

## 12.6 EXERCISES

12.1 An alternative derivation of the Gauss-Newton algorithm shows that it can be thought of as an approximation of Newton's method for unconstrained optimization.

(a) Write an expression for the Hessian of $E_{\text{NLS}}(\vec{x})$ (defined in §12.1) in terms of the derivatives of the $f_i$'s.

(b) Show that the Gauss-Newton algorithm on $E_{\text{NLS}}$ is equivalent to Newton's method (§9.4.2) after removing second derivative terms from the Hessian.

(c) When is such an approximation of the Hessian reasonable?

12.2 Motivate the Levenberg-Marquardt algorithm by applying Tikhonov regularization to the Gauss-Newton algorithm.

12.3 Derive steps of an alternating least-squares (ALS) iterative algorithm for minimizing $\|X - CY\|_{\text{Fro}}$ with respect to $C \in \mathbb{R}^{n\times d}$ and $Y \in \mathbb{R}^{d\times k}$, given a fixed matrix $X \in \mathbb{R}^{n\times k}$. Explain how the output of your algorithm depends on the initial guesses of $C$ and $Y$. Provide an extension of your algorithm that orthogonalizes the columns of $C$ in each iteration using its reduced QR factorization, and argue why the energy still decreases in each iteration.

12.4 Incorporate matrix factorization into the nonnegative least-squares algorithm in Example 12.8 to make the $\vec{x}$ step more efficient. When do you expect this modification to improve the speed of the algorithm?

12.5 For a fixed parameter $\delta > 0$, the *Huber loss function* $L_\delta(x)$ is defined as:

$$L_\delta(x) \equiv \begin{cases} {}^{x^2}/_2, & \text{when } |x| \leq \delta \\ \delta(|x| - {}^{\delta}/_2), & \text{otherwise.} \end{cases}$$

This function "softens" the non-differentiable singularity of $|x|$ at $x = 0$.

(a) Illustrate the effect of choosing different values of $\delta$ on the shape of $L_\delta(x)$.

(b) Recall that we can find an $\vec{x}$ nearly satisfying the overdetermined system $A\vec{x} \approx \vec{b}$ by minimizing $\|A\vec{x} - \vec{b}\|_2$ (least-squares) or $\|A\vec{x} - \vec{b}\|_1$ (compressive sensing). Propose a similar optimization compromising between these two methods using $L_\delta$.

(c) Propose an IRLS algorithm for optimizing your objective from Exercise 12.5b. You can assume $A^\top A$ is invertible.

(d) Propose an ADMM algorithm for optimizing your objective from Exercise 12.5b. Again, assume $A^\top A$ is invertible.
*Hint:* Introduce a variable $\vec{z} = A\vec{x} - \vec{b}$.

[DH] 12.6 (From notes by P. Blomgren) In §12.4.1, we introduced homotopy continuation methods for optimization. These methods begin by minimizing a simple objective $H(\vec{x}, 0) = f_0(\vec{x})$ and then smoothly modify the objective and minimizer simultaneously until a minimum of $H(\vec{x}, 1) = f(\vec{x})$—the original objective—is found.

Suppose that $s(t)$ is a function of $t \geq 0$ such that $s(0) = 0$; we will assume that $s(t) \geq 0$ for all $t \geq 0$ and that $s(t)$ eventually reaches $s(t) = 1$. Our goal is to produce a path $\vec{x}(t)$ such that each $\vec{x}(t)$ minimizes $H(\vec{x}, s(t))$ with respect to $\vec{x}$.

(a) To maintain optimality of $\vec{x}(t)$, what relationship does $\nabla_{\vec{x}} H(\vec{x}, s)$ satisfy for all $t \geq 0$ at points $(\vec{x}(t), s(t))$ on the solution path?

(b) Differentiate this equation with respect to $t$. Write one side as a matrix-vector product.

(c) Provide a geometric interpretation of the vector $\vec{g}(t) \equiv (\vec{x}'(t), s'(t))$ in terms of the solution path $(\vec{x}(t), s(t))$.

(d) We will impose the restriction that $||\vec{g}(t)||_2 = 1\ \forall t \geq 0$, i.e., that $\vec{g}(t)$ has unit length. In this case, what is the geometric interpretation of $t$ in terms of the solution path?

(e) Combine Exercises 12.6b and 12.6d to propose an ordinary differential equation (ODE) for computing $(\vec{x}'(t), s'(t))$ from $(\vec{x}(t), s(t))$, so that the resulting solution path maintains our design constraints.
*Note:* Using this formula, numerical ODE solvers like the ones we will propose in Chapter 15 can calculate a solution path for homotopy continuation optimization. This derivation provides a connection between topology, optimization, and differential equations.

12.7 ("Least absolute deviations") Instead of solving least-squares, to take advantage of methods from compressive sensing we might wish to minimize $\|A\vec{x} - \vec{b}\|_1$ with $\vec{x}$ unconstrained. Propose an ADMM-style splitting of this optimization and give the alternating steps of the optimization technique in this case.

[DH] 12.8 Suppose we have two convex sets $S, T \subseteq \mathbb{R}^n$. The *alternating projection* method discussed in [9] and elsewhere is used to find a point $\vec{x} \in S \cap T$. For any initial guess $\vec{x}_0$, alternating projection performs the iteration

$$\vec{x}_{k+1} = \mathcal{P}_S\left(\mathcal{P}_T\left(\vec{x}_k\right)\right),$$

where $\mathcal{P}_S$ and $\mathcal{P}_T$ are operators that project onto the nearest point in $S$ or $T$ with respect to $\|\cdot\|_2$, respectively. As long as $S \cap T \neq \emptyset$, this iterative procedure is guaranteed to converge to an $\vec{x} \in S \cap T$, though this convergence may be impractically slow [23]. Instead of this algorithm, we will consider finding a point in the intersection of convex sets using ADMM.

(a) Propose an unconstrained optimization problem whose solution is a point $\vec{x} \in S \cap T$, assuming $S \cap T \neq \emptyset$.
*Hint:* Use indicator functions.

(b) Write this problem in a form that is amenable to ADMM, using $\vec{x}$ and $\vec{z}$ as your variables.

(c) Explicitly write the ADMM iterations for updating $\vec{x}$, $\vec{z}$, and any dual variables. Your expressions can use $\mathcal{P}_S$ and $\mathcal{P}_T$.

[DH] 12.9 A popular technique for global optimization is *simulated annealing* [73], a method motivated by ideas from statistical physics. The term *annealing* refers to the process in metallurgy whereby a metal is heated and then cooled so its constituent particles arrange in a minimum energy state. In this thermodynamic process, atoms may move considerably at higher temperatures but become restricted in motion as the temperature decreases. Borrowing from this analogy in the context of global optimization, we could let a potential optimal point take large, random steps early on in a search to explore the space of outputs, eventually taking smaller steps as the number of iterations gets large, to obtain a more refined output. Pseudocode for the resulting simulated annealing algorithm is provided in the following box.

```
function SIMULATED-ANNEALING(f(x⃗), x⃗_0)
  T_0 ← High temperature
  T_i ← Cooling schedule, e.g., T_i = αT_{i−1} for some α < 1
  x⃗ ← x⃗_0                          ▷ Current model initialized to the input x⃗_0
  for i ← 1, 2, 3, ...
    y⃗ ← RANDOM-MODEL                          ▷ Random guess of output
    Δf ← f(y⃗) − f(x⃗)                          ▷ Compute change in objective
    if Δf < 0 then                             ▷ Objective improved at y⃗
      x⃗ ← y⃗
    else if UNIFORM(0,1)< e^{−Δf/T_i} then     ▷ True with probability e^{−Δf/T_i}
      x⃗ ← y⃗                                   ▷ Randomly keep suboptimal output
```

Simulated annealing randomly guesses a solution to the optimization problem in each iteration. If the new solution achieves a lower objective value than the current solution,

the algorithm keeps the new solution. If the new solution is less optimal, however, it is not necessarily rejected. Instead, the suboptimal point is accepted with exponentially small probability as temperature decreases. The hope of this construction is that local minima will be avoided early on in favor of global minima due to the significant amount of exploration during the first few iterations, while some form of convergence is still obtained as the iterates stabilize at lower temperatures.

Consider the Euclidean traveling salesman problem (TSP): Given a set of points $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^2$ representing the positions of cities on a map, we wish to visit each city exactly once while minimizing the total distance traveled. While Euclidean TSP is NP-hard, simulated annealing provides a practical way to approximate its solution.

(a) Phrase Euclidean TSP as a global optimization problem. It is acceptable to have variables that are discrete rather than continuous.

(b) Propose a method for generating random tours that reach each city exactly once. What $f$ should you use to evaluate the quality of a tour?

(c) Implement your simulated annealing solution to Euclidean TSP and explore the trade-off between solution quality and runtime when the initial temperature $T_0$ is changed. Also, experiment with different cooling schedules, either by varying $\alpha$ in the example $T_i$ or by proposing your own cooling schedule.

(d) Choose another global optimization algorithm and explain how to use it to solve Euclidean TSP. Analyze how its efficiency compares to that of simulated annealing.

(e) Rather than generating a completely new tour in each iteration of simulated annealing, propose a method that perturbs tours slightly to generate new ones. What would be the advantages and/or disadvantages of using this technique in place of totally random models?

SC 12.10 Recall the setup from Exercise 10.7 for designing a slow-dissolving medicinal capsule shaped as a cylinder with hemispherical ends.

(a) Suppose we are unhappy with the results of the optimization proposed in Exercise 10.7 and want to ensure that the volume of the *entire* capsule (including the ends) is at least $V$. Explain why the resulting problem cannot be solved using geometric programming methods.

(b) Propose an alternating optimization method for this problem. Is it necessary to solve a geometric program in either alternation?

12.11 The mean shift algorithm, originally proposed in [27], is an iterative clustering technique appearing in literature on nonparametric machine learning and image processing. Given $n$ data points $\vec{x}_i \in \mathbb{R}^d$, the algorithm groups points together based on their closest maxima in a smoothed density function approximating the distribution of data points.

(a) Take $k(x) : \mathbb{R} \to \mathbb{R}^+$ to be a nonnegative function. For a fixed bandwidth parameter $h > 0$, define the *kernel density estimator* $\hat{f}(\vec{x})$ to be

$$\hat{f}_k(\vec{x}) \equiv \frac{c_{k,d}}{nh^d} \sum_{i=1}^{n} k\left( \left\| \frac{\vec{x} - \vec{x}_i}{h} \right\|_2^2 \right).$$

If $k(x)$ is peaked at $x = 0$, explain how $\hat{f}_k(\vec{x})$ encodes the density of data points $\vec{x}_i$. What is the effect of increasing the parameter $h$?
*Note:* The constant $c_{k,d}$ is chosen so that $\int_{\mathbb{R}^d} \hat{f}(\vec{x})\, d\vec{x} = 1$. Choosing $k(x) \equiv e^{-x/2}$ makes $\hat{f}$ a sum of Gaussians.

(b) Define $g(x) \equiv -k'(x)$ and take $m(\vec{x})$ to be the *mean shift* vector given by

$$m(\vec{x}) \equiv \frac{\sum_i \vec{x}_i g\left(\left\|\frac{\vec{x}-\vec{x}_i}{h}\right\|_2^2\right)}{\sum_i g\left(\left\|\frac{\vec{x}-\vec{x}_i}{h}\right\|_2^2\right)} - \vec{x}.$$

Show that $\nabla \hat{f}_k(\vec{x})$ can be factored as follows:

$$\nabla \hat{f}_k(\vec{x}) = \frac{\alpha}{h^2} \cdot \hat{f}_g(\vec{x}) \cdot m(\vec{x}),$$

for some constant $\alpha$.

(c) Suppose $\vec{y}_0$ is a guess of the location of a peak of $\hat{f}_k$. Using your answer from Exercise 12.11b, motivate the *mean shift* algorithm for finding a peak of $\hat{f}_k(\vec{x})$, which iterates the formula

$$\vec{y}_{k+1} \equiv \frac{\sum_i \vec{x}_i g\left(\left\|\frac{\vec{y}_k-\vec{x}_i}{h}\right\|_2^2\right)}{\sum_i g\left(\left\|\frac{\vec{y}_k-\vec{x}_i}{h}\right\|_2^2\right)}.$$

*Note:* This algorithm is guaranteed to converge under mild conditions on $k$. Mean shift clustering runs this method to convergence starting from $\vec{y}_0 = \vec{x}_i$ for each $i$ in parallel; $\vec{x}_i$ and $\vec{x}_j$ are assigned to the same cluster if mean shift iteration yields the same output (within some tolerance) for starting points $\vec{y}_0 = \vec{x}_i$ and $\vec{y}_0 = \vec{x}_j$.

(d) Suppose we represent a grayscale image as a set of pairs $(\vec{p}_i, q_i)$, where $\vec{p}_i$ is the center of pixel $i$ (typically laid out on a grid), and $q_i \in [0, 1]$ is the intensity of pixel $i$. The *bilateral filter* [120] for blurring images while preserving their sharp edges is given by:

$$\hat{q}_i \equiv \frac{\sum_j q_j k_1(\|\vec{p}_j - \vec{p}_i\|_2) k_2(|q_j - q_i|)}{\sum_j k_1(\|\vec{p}_j - \vec{p}_i\|_2) k_2(|q_j - q_i|)},$$

where $k_1, k_2$ are Gaussian kernels given by $k_i(x) \equiv e^{-a_i x^2}$. Fast algorithms have been developed in the computer graphics community for evaluating the bilateral filter and its variants [97].
Propose an algorithm for clustering the pixels in an image using iterated calls to a modified version of the bilateral filter; the resulting method is called the "local mode filter" [125, 96].

12.12 The *iterative shrinkage-thresholding algorithm* (ISTA) is another technique relevant to large-scale optimization applicable to common objectives from machine learning. Extensions such as [11] have led to renewed interest in this technique. We follow the development of [20].

(a) Show that the iteration from gradient descent

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

can be rewritten in *proximal form* as

$$\vec{x}_{k+1} = \arg\min_{\vec{x}} \left[ f(\vec{x}_k) + \nabla f(\vec{x}_k)^\top (\vec{x} - \vec{x}_k) + \frac{1}{2\alpha} \|\vec{x} - \vec{x}_k\|_2^2 \right].$$

(b) Suppose we wish to minimize a sum $f(\vec{x}) + g(\vec{x})$. Based on the previous part, ISTA attempts to combine exact optimization for $g$ with gradient descent on $f$:

$$\vec{x}_{k+1} \equiv \arg\min_{\vec{x}} \left[ f(\vec{x}_k) + \nabla f(\vec{x}_k)^\top (\vec{x} - \vec{x}_k) + \frac{1}{2\alpha} \|\vec{x} - \vec{x}_k\|_2^2 + g(\vec{x}) \right].$$

Derive the alternative form

$$\vec{x}_{k+1} = \arg\min_{\vec{x}} \left[ g(\vec{x}) + \frac{1}{2\alpha} \|\vec{x} - (\vec{x}_k - \alpha \nabla f(\vec{x}_k))\|_2^2 \right].$$

(c) Derive a formula for ISTA iterations when $g(\vec{x}) = \lambda \|\vec{x}\|_1$, where $\lambda > 0$.
*Hint:* This case reduces to solving a set of single-variable problems.

12.13 Suppose $\mathcal{D}$ is a bounded, convex, and closed domain in $\mathbb{R}^n$ and $f(\vec{x})$ is a convex, differentiable objective function. The *Frank-Wolfe* algorithm for minimizing $f(\vec{x})$ subject to $\vec{x} \in \mathcal{D}$ is as follows [43]:

$$\begin{aligned} \vec{s}_k &\leftarrow \arg\min_{\vec{s} \in \mathcal{D}} [\vec{s} \cdot \nabla f(\vec{x}_{k-1})] \\ \gamma_k &\leftarrow \frac{2}{k+2} \\ \vec{x}_k &\leftarrow (1 - \gamma_k)\vec{x}_{k-1} + \gamma_k \vec{s}_k. \end{aligned}$$

A starting point $\vec{x}_0 \in \mathcal{D}$ must be provided. This algorithm has gained renewed attention for large-scale optimization in machine learning in the presence of sparsity and other specialized structure [66].

(a) Argue that $\vec{s}_k$ minimizes a linearized version of $f$ subject to the constraints. Also, show that if $\mathcal{D} = \{\vec{x} : A\vec{x} \leq \vec{b}\}$ for fixed $A \in \mathbb{R}^{m \times n}$ and $\vec{b} \in \mathbb{R}^m$, then each iteration of the Frank-Wolfe algorithm solves a linear program.

(b) Show that $\vec{x}_k \in \mathcal{D}$ for all $k > 0$.

(c) Assume $\nabla f(\vec{x})$ is $L$-Lipschitz on $\mathcal{D}$, meaning $\|\nabla f(\vec{x}) - \nabla f(\vec{y})\|_2 \leq L\|\vec{x} - \vec{y}\|_2$, for all $\vec{x}, \vec{y} \in \mathcal{D}$. Derive the bound (proposed in [88]):

$$|f(\vec{y}) - f(\vec{x}) - (\vec{y} - \vec{x}) \cdot \nabla f(\vec{x})| \leq \frac{L}{2} \|\vec{y} - \vec{x}\|_2^2.$$

*Hint:* By the Fundamental Theorem of Calculus, $f(\vec{y}) = f(\vec{x}) + \int_0^1 (\vec{y} - \vec{x}) \cdot \nabla f(\vec{x} + \tau(\vec{y} - \vec{x}))\, d\tau$.

(d) Define the *diameter* of $\mathcal{D}$ to be $d \equiv \max_{\vec{x},\vec{y}\in\mathcal{D}} \|\vec{x}-\vec{y}\|_2$. Furthermore, assume $\nabla f(\vec{x})$ is $L$-Lipschitz on $\mathcal{D}$. Show that

$$\frac{2}{\gamma^2}(f(\vec{y}) - f(\vec{x}) - (\vec{y}-\vec{x})\cdot\nabla f(\vec{x})) \leq d^2 L,$$

for all $\vec{x},\vec{y},\vec{s}\in\mathcal{D}$ with $\vec{y} = \vec{x} + \gamma(\vec{s}-\vec{x})$ and $\gamma\in[0,1]$. Conclude that

$$f(\vec{y}) \leq f(\vec{x}) + \gamma(\vec{s}-\vec{x})\cdot\nabla f(\vec{x}) + \frac{\gamma^2 d^2 L}{2}.$$

(e) Define the *duality gap* $g(\vec{x}) \equiv \max_{\vec{s}\in\mathcal{D}}(\vec{x}-\vec{s})\cdot\nabla f(\vec{x})$. For the Frank-Wolfe algorithm, show that

$$f(\vec{x}_k) \leq f(\vec{x}_{k-1}) - \gamma g(\vec{x}_{k-1}) + \frac{\gamma_k^2 d^2 L}{2}.$$

(f) Take $\vec{x}^*$ to be the location of the minimum for the optimization problem, and define $h(\vec{x}) \equiv f(\vec{x}) - f(\vec{x}^*)$. Show $g(\vec{x}) \geq h(\vec{x})$, and using the previous part conclude

$$h(\vec{x}_k) \leq (1-\gamma_k)h(\vec{x}_{k-1}) + \frac{\gamma_k^2 d^2 L}{2}.$$

(g) Conclude $h(\vec{x}_k)\to 0$ as $k\to\infty$. What does this imply about the Frank-Wolfe algorithm?

12.14 The FTRL algorithm from §12.5 can be expensive when the $f_t$'s are difficult to minimize. In this problem, we derive a linearized alternative with similar performance guarantees.

(a) Suppose we make the following assumptions about an instance of FTRL:
- $U = \{\vec{x}\in\mathbb{R}^n : \|\vec{x}\|_2 \leq 1\}$.
- All of the objectives $f_t$ provided to FTRL are of the form $f_t(\vec{x}) = \vec{z}_t\cdot\vec{x}$ for $\|\vec{z}_t\|_2 \leq 1$.
- $r(\vec{x}) \equiv \frac{1}{2}\sigma\|\vec{x}\|_2^2$.

Provide an explicit formula for the iterates $\vec{x}_t$ in this case, and specialize the bound from Proposition 12.1.

(b) We wish to apply the bound from 12.14a to more general $f_t$'s. To do so, suppose we replace FTRL with a linearized objective for $\vec{x}_t$:

$$\vec{x}_t \equiv \arg\min_{\vec{x}\in U}\left[r(\vec{x}) + \sum_{s=1}^{t-1}(f_s(\vec{x}_s) + \nabla f_s(\vec{x}_s)\cdot(\vec{x}-\vec{x}_s))\right].$$

Provide an explicit formula for $\vec{x}_t$ in this case, assuming the same choice of $U$ and $r$.

(c) Propose a regret bound for the linearized method in 12.14b.
*Hint:* Apply convexity of the $f_t$'s and the result of 12.14a.