Chapter 10

Advanced Descent Methods

Relevant sections of the textbooks:

- [1] Chapter 11.
- [2] Section 12.5.

10.1 Coordinate Descent

Recall from Chapter 6 the general idea of descent-based methods for solving optimization problems. These methods are based on the process of starting with some initial guess $\vec{x}^{(0)} \in \mathbb{R}^n$, then generating a sequence of refined guesses $\vec{x}^{(1)}, \vec{x}^{(2)}, \vec{x}^{(3)}, \ldots$ using the general update rule

$$\vec{x}^{(t+1)} = \vec{x}^{(t)} + \eta \vec{v}^{(t)} \tag{10.1}$$

for some search direction $\vec{v}^{(t)}$ and step size η. In Chapter 6 we covered the gradient descent method, which uses the gradient of the function as the search direction. In this chapter we will revisit descent-based optimization methods and introduce alternative update rules.

In this section we will introduce *coordinate descent*, a class of descent-based algorithms that finds a minimizer of multivariate functions by iteratively minimizing it along one direction at a time. Consider the unconstrained convex optimization problem

$$p^\star = \min_{\vec{x} \in \mathbb{R}^n} f(\vec{x}), \tag{10.2}$$

with the optimization variable

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}. \tag{10.3}$$

Before we introduce the algorithm, we introduce some notation. For indices i and j, we introduce the notation $\vec{x}_{i:j} = (x_i, x_{i+1}, \ldots, x_{j-1}, x_j) \in \mathbb{R}^{j-i+1}$ to be the entries of $\vec{x}$ between indices i and j (inclusive on both ends).

Given an initial guess $\vec{x}^{(0)}$, for $t \geq 0$ the coordinate descent algorithm updates the iterate $\vec{x}^{(t)}$ by *sequentially* minimizing the function $f(\vec{x})$ with respect to each coordinate, namely using the update rule

$$x_i^{(t+1)} \in \operatorname*{argmin}_{x_i \in \mathbb{R}} f(\vec{x}_{1:i-1}^{(t+1)}, x_i, \vec{x}_{i+1:n}^{(t)}). \tag{10.4}$$

Namely, we perform the update

$$x_1^{(t+1)} \in \underset{x_1 \in \mathbb{R}}{\operatorname{argmin}} f(x_1, \vec{x}_{2:n}^{(t)}) \tag{10.5}$$

$$x_2^{(t+1)} \in \underset{x_2 \in \mathbb{R}}{\operatorname{argmin}} f(x_1^{(t+1)}, x_2, \vec{x}_{3:n}^{(t)}) \tag{10.6}$$

$$\vdots \qquad \vdots \tag{10.7}$$

$$x_n^{(t+1)} \in \underset{x_n \in \mathbb{R}}{\operatorname{argmin}} f(\vec{x}_{1:n-1}^{(t+1)}, x_n). \tag{10.8}$$

This is a sequential process since after finding the minimizer along the i^{th} coordinate (i.e. $x_i^{(t+1)}$) we use its values for minimizing subsequent coordinates. Also we note that the order of the coordinates is arbitrary. We formalize this update in the following algorithm.

Algorithm 6 CoordinateDescent

function CoordinateDescent($f, \vec{x}^{(0)}, T$)
  for $t = 0, 1, \ldots, T_1$ **do**
    for $i = 1, \ldots, N$ **do**
      $x_i^{(t+1)} \leftarrow \operatorname{argmin}_{x_i \in \mathbb{R}} f(x_{1:i-1}^{(t+1)}, x_i, x_{i+1:n}^{(t)})$.
    end for
  end for
  return $\vec{x}^T$
end function

The algorithm breaks down the difficult multivariate optimization problem into a sequence of simpler univariate optimization problems.

We first want to discuss the issue of well-posedness of the algorithm. We know that any of the argmins used may not exist, in which case the algorithm is not well-defined, and so we cannot even think about its behavior or convergence. Nevertheless, in a large class of problems which have many different characterizations, the argmins are well-defined. We say in this case that the coordinate descent algorithm is well-posed.

We now want to address the question of convergence. It is not obvious that minimizing the function $f(\vec{x})$ can be achieved by minimizing along each direction separately. In fact, the algorithm is not guaranteed to converge to an optimal solution for general convex functions. However, under some additional assumptions on the function, we can guarantee convergence. To build an intuition for what additional assumptions are needed we consider the following question. Let $f(\vec{x})$ be a convex differentiable function. Suppose that $x_i^\star \in \operatorname{argmin}_{x_i \in \mathbb{R}} f(x_{1:i-1}^\star, x_i, x_{i+1:n}^\star)$ for all i. Can we conclude that $\vec{x}^\star$ is a global minimizer of $f(\vec{x})$? The answer to this question is yes. We can prove this by recalling the first order optimality condition for unconstrained convex functions and the definition of partial derivatives. If $\vec{x}^\star$ is a minimizer of $f(\vec{x})$ along the direction $\vec{e}_i$ then we have

$$\frac{\partial f}{\partial x_i}(\vec{x}^\star) = 0. \tag{10.9}$$

If this is true for all i then $\nabla f(\vec{x}^\star) = \vec{0}$, implying that $\vec{x}^\star$ is a global minimizer for f. This discussion forms a proof of the following theorem, which is Theorem 12.4 in [2].

Theorem 212 (Convergence of Coordinate Descent for Differentiable Convex Functions)
Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a differentiable convex function which is *separately strictly convex* in each argument. That is, suppose that for each i, and each fixed $\vec{x}_{1:i-1}$ and $\vec{x}_{i+1:n}$, the function $x_i \mapsto f(\vec{x}_{1:i-1}, x_i, \vec{x}_{i+1:n})$ is strictly convex. If the coordinate descent algorithm is well-posed, and the unconstrained minimization problem

$$\min_{\vec{x}\in\mathbb{R}^n} f(\vec{x}) \tag{10.10}$$

has a solution, then the sequence of iterates $\vec{x}^{(0)}, \vec{x}^{(1)}, \ldots$ generated by the coordinate descent algorithm converges to an optimal solution to (10.10).

The coordinate descent algorithm may not converge to an optimal solution for general non-differentiable functions, even if they are convex. However, we can still prove that coordinate descent converges for a special class of functions of the form

$$f(\vec{x}) = g(\vec{x}) + \sum_{i=1}^{n} h_i(x_i) \tag{10.11}$$

where $g\colon \mathbb{R}^n \to \mathbb{R}$ is convex and differentiable, and each $h_i\colon \mathbb{R} \to \mathbb{R}$ is convex (but not necessarily differentiable). This form includes various ℓ^1 regularization problems (such as LASSO regression) which have a separable non-differentiable component. The provable convergence of coordinate descent algorithm makes it an attractive choice for this class of problems.

Example 213. In this example we will consider the LASSO regression problem and examine how coordinate descent algorithm can be applied to solve it. Note that the LASSO objective follows the form described in (10.11). For $A \in \mathbb{R}^{m\times n}$ which has columns $\vec{a}_1, \ldots, \vec{a}_n \in \mathbb{R}^m$, and $\vec{y} \in \mathbb{R}^m$, we consider the LASSO objective

$$f(\vec{x}) = \frac{1}{2}\|A\vec{x} - \vec{y}\|_2^2 + \lambda \|x\|_1 . \tag{10.12}$$

We aim to use coordinate descent to minimize this function. Let $\vec{x}^{(0)}$ be the initial guess. Then we perform the coordinate descent update by solving the following optimization problem:

$$x_i^{(t+1)} = \operatorname*{argmin}_{x_i\in\mathbb{R}} f(\vec{x}_{1:i-1}^{(t+1)}, x_i, \vec{x}_{i+1:n}^{(t)}). \tag{10.13}$$

Each of these optimization problems will be solved similarly to what we did in Section 9.3. For notational clarity, let us instead solve the more generic problem

$$x_i^\star \dot{\in} \operatorname*{argmin}_{x_i\in\mathbb{R}} f(\vec{x}_{1:i-1}, x_i, \vec{x}_{i+1:n}). \tag{10.14}$$

Then we have that $\frac{\partial f}{\partial x_i}(\vec{x}_{1:i-1}, x_i^\star, \vec{x}_{i+1:n})$ is either 0 or undefined. Thus we compute each partial derivative of f. Indeed, we have

$$\frac{\partial f}{\partial x_i}(\vec{x}) = \frac{1}{2}\frac{\partial}{\partial x_i}\|A\vec{x} - \vec{y}\|_2^2 + \lambda \frac{\partial}{\partial x_i}\|\vec{x}\|_1 \tag{10.15}$$

$$= \vec{e}_i^\top \nabla \left(\frac{1}{2}\|A\vec{x} - \vec{y}\|_2^2\right) + \lambda \frac{\partial}{\partial x_i}\sum_{j=1}^{n}|x_j| \tag{10.16}$$

$$= \vec{e}_i^\top A^\top (A\vec{x} - \vec{y}) + \lambda \frac{\partial}{\partial x_i}|x_i| \tag{10.17}$$

$$= \vec{a}_i^\top (A\vec{x} - \vec{y}) + \lambda \begin{cases} \operatorname{sgn}(x_i), & \text{if } x_i \neq 0 \\ \text{undefined}, & \text{if } x_i = 0. \end{cases} \tag{10.18}$$

We now introduce the additional notation $A_{i:j} \in \mathbb{R}^{m\times(j-i+1)}$ as the sub-matrix of A whose columns are the i^{th} through j^{th} columns of A (inclusive). Using this notation, we can simplify the first term as

$$\vec{a}_i^\top (A\vec{x} - \vec{y}) = \vec{a}_i^\top A\vec{x} - \vec{a}_i^\top \vec{y} \tag{10.19}$$

$$= \vec{a}_i^\top \left(\sum_{j=1}^{n} \vec{a}_j x_j - \vec{y} \right) \tag{10.20}$$

$$= \|\vec{a}_i\|_2^2 \, x_i + \vec{a}_i^\top \left(\sum_{\substack{j=1 \\ j\neq i}}^{n} \vec{a}_j x_j - \vec{y} \right) \tag{10.21}$$

$$= \|\vec{a}_i\|_2^2 \, x_i + \vec{a}_i^\top (A_{1:i-1}\vec{x}_{1:i-1} + A_{i+1:n}\vec{x}_{i+1:n} - \vec{y}) \tag{10.22}$$

Now there are two cases depending on the value of $x_i^\star$.

Case 1. Suppose $x_i^\star \neq 0$. Then we have by the optimality condition that

$$0 = \frac{\partial f}{\partial x_i}(\vec{x}_{1:i-1}, x_i^\star, \vec{x}_{i+1:n}) \tag{10.23}$$

$$= \|\vec{a}_i\|_2^2 x_i^\star + \vec{a}_i^\top \left(A_{1:i-1}\vec{x}_{1:i-1} + A_{i+1:n}\vec{x}_{i+1:n} - \vec{y}\right) + \lambda \operatorname{sgn}(x_i^\star) \tag{10.24}$$

$$\implies x_i^\star = \frac{1}{\|\vec{a}_i\|_2^2} \left(\vec{a}_i^\top \left[\vec{y} - A_{1:i-1}\vec{x}_{1:i-1} - A_{i+1:n}\vec{x}_{i+1:n}\right] - \lambda \operatorname{sgn}(x_i^\star) \right). \tag{10.25}$$

Thus in particular we have

$$x_i^\star > 0 \iff \vec{a}_i^\top \left(\vec{y} - A_{1:i-1}\vec{x}_{1:i-1} - A_{i+1:n}\vec{x}_{i+1:n}\right) > \lambda, \tag{10.26}$$

and the corresponding relation

$$x_i^\star < 0 \iff \vec{a}_i^\top \left(\vec{y} - A_{1:i-1}\vec{x}_{1:i-1} - A_{i+1:n}\vec{x}_{i+1:n}\right) < -\lambda. \tag{10.27}$$

Case 2. By contrapositive, we have

$$-\lambda \leq \vec{a}_i^\top \left(\vec{y} - A_{1:i-1}\vec{x}_{1:i-1} - A_{i+1:n}\vec{x}_{i+1:n}\right) \leq \lambda \iff x_i^\star = 0. \tag{10.28}$$

Therefore we have derived a closed-form coordinate descent update:

$$\vec{a}_i^\top \left(\vec{y} - A_{1:i-1}\vec{x}_{1:i-1}^{(t+1)} - A_{i+1:n}\vec{x}_{i+1:n}^{(t)}\right) > \lambda \implies x_i^{(t+1)} = \frac{1}{\|\vec{a}_i\|_2^2} \left(\vec{a}_i^\top \left[\vec{y} - A_{1:i-1}\vec{x}_{1:i-1}^{(t+1)} - A_{i+1:n}\vec{x}_{i+1:n}^{(t)}\right] - \lambda \right) \tag{10.29}$$

$$\vec{a}_i^\top \left(\vec{y} - A_{1:i-1}\vec{x}_{1:i-1}^{(t+1)} - A_{i+1:n}\vec{x}_{i+1:n}^{(t)}\right) < -\lambda \implies x_i^{(t+1)} = \frac{1}{\|\vec{a}_i\|_2^2} \left(\vec{a}_i^\top \left[\vec{y} - A_{1:i-1}\vec{x}_{1:i-1}^{(t+1)} - A_{i+1:n}\vec{x}_{i+1:n}^{(t)}\right] + \lambda \right) \tag{10.30}$$

$$\left| \vec{a}_i^\top \left(\vec{y} - A_{1:i-1}\vec{x}_{1:i-1}^{(t+1)} - A_{i+1:n}\vec{x}_{i+1:n}^{(t)}\right) \right| \leq \lambda \implies x_i^{(t+1)} = 0. \tag{10.31}$$

10.2 Newton's Method

Let us consider the unconstrained optimization problem

$$p^\star = \min_{\vec{x}\in\mathbb{R}^n} f(\vec{x}). \tag{10.32}$$

Recall that in the gradient descent algorithm, we assumed that the objective function $f(\vec{x})$ is differentiable. Furthermore, we assumed that at every point $\vec{x} \in \mathbb{R}^n$ we can compute $f(\vec{x})$ as well as $\nabla f(\vec{x})$. Here, we make the additional assumption that $f(\vec{x})$ is twice differentiable and that we can compute the Hessian $\nabla^2 f(\vec{x})$. We wish to use the Hessian to choose a good search direction and accelerate convergence. Optimization algorithms that utilize second derivatives (e.g. the Hessian) are called *second-order methods*.

One of the most famous second-order methods is *Newton's method*. Newton's method is based on the following idea for minimizing strictly-convex functions with positive definite Hessians: first, start with an initial guess $\vec{x}^{(0)}$. Then in each iteration $t = 1, 2, 3, \ldots$, approximate the objective function with its second-order Taylor approximation around the point $\vec{x}^{(t)}$. The minimizer of this quadratic approximation is then chosen as the next iterate $\vec{x}^{(t+1)}$.

More formally, let us assume that f is strictly convex and twice-differentiable with positive definite Hessian at $\vec{x}^{(t)}$, and let us write the second-order Taylor approximation of the function $f(\vec{x})$ around the point $\vec{x}^{(t)}$. We obtain

$$\widehat{f}_2(\vec{x}; \vec{x}^{(t)}) = f(\vec{x}^{(t)}) + [\nabla f(\vec{x}^{(t)})]^\top (\vec{x} - \vec{x}^{(t)}) + \frac{1}{2}(\vec{x} - \vec{x}^{(t)})^\top [\nabla^2 f(\vec{x}^{(t)})](\vec{x} - \vec{x}^{(t)}). \tag{10.33}$$

Since the Hessian $\nabla^2 f(\vec{x}^{(t)}) \succ 0$, we can solve the problem

$$\min_{\vec{x} \in \mathbb{R}^n} \widehat{f}_2(\vec{x}; \vec{x}^{(t)}), \tag{10.34}$$

which is a convex quadratic program, using our (by now) standard techniques. Setting the gradient (in $\vec{x}$) to $\vec{0}$, we obtain

$$\vec{0} = \nabla \widehat{f}_2(\vec{x}^\star; \vec{x}^{(t)}) \tag{10.35}$$

$$= \nabla f(\vec{x}^{(t)}) + [\nabla^2 f(\vec{x}^{(t)})](\vec{x}^\star - \vec{x}^{(t)}) \tag{10.36}$$

$$\implies \vec{x}^\star = \vec{x}^{(t)} - [\nabla^2 f(\vec{x}^{(t)})]^{-1} [\nabla f(\vec{x}^{(t)})]. \tag{10.37}$$

This gives the Newton's method update rule:

$$\vec{x}^{(t+1)} = \vec{x}^{(t)} - [\nabla^2 f(\vec{x}^{(t)})]^{-1} [\nabla f(\vec{x}^{(t)})]. \tag{10.38}$$

The formal algorithm, detailed in Algorithm 7, just repeats this iteration.

Algorithm 7 Newton's Method

function NEWTONMETHOD($f, \vec{x}^{(0)}, T$)
  for $t = 0, 1, \ldots, T-1$ **do**
    $\vec{x}^{(t+1)} \leftarrow \vec{x}^{(t)} - \left[\nabla^2 f(\vec{x}^{(t)})\right]^{-1} [\nabla f(\vec{x}^{(t)})]$
  end for
  return $\vec{x}^{(T)}$
end function

We call the vector $\left[\nabla^2 f(\vec{x}^{(t)})\right]^{-1} [\nabla f(\vec{x}^{(t)})]$ the Newton direction. Here, we do not choose a step-size η. Instead, we take a *full* step in the Newton direction towards the minimizer of the quadratic approximation of the objective function. This is the basic version of Newton's method; it is *not* guaranteed to converge in general. To achieve convergence, we can introduce a step-size $\eta > 0$ to the Newton update, obtaining the so-called *damped* Newton's method, which has the iteration

$$\vec{x}^{(t+1)} = \vec{x}^{(t)} - \eta \left[\nabla^2 f(\vec{x}^{(t)})\right]^{-1} [\nabla f(\vec{x}^{(t)})]. \tag{10.39}$$

A full algorithm is very similar to Algorithm 7 and is provided in Algorithm 8.

Algorithm 8 Damped Newton's Method

function DAMPEDNEWTONMETHOD($f, \vec{x}^{(0)}, \eta, T$)
  for $t = 0, 1, \ldots, T-1$ **do**
    $\vec{x}^{(t+1)} \leftarrow \vec{x}^{(t)} - \eta \left[\nabla^2 f(\vec{x}^{(t)})\right]^{-1} \left[\nabla f(\vec{x}^{(t)})\right]$
  end for
  return $\vec{x}^{(T)}$
end function

We will not discuss convergence proofs of Newton's method in this course; you may use [7] for further reading.

All of our discussion thus far has been under the assumption $\nabla^2 f(\vec{x}^{(t)}) \succ 0$. If the Hessian is not positive definite, one may adapt the algorithm accordingly, forming a new class of methods called *quasi-Newton's methods*. Discussion of quasi-Newton's methods are out of scope of the course.

Finally, we discuss the algorithmic complexity of Newton's method. In every iteration, we need to compute and invert the Hessian $\nabla^2 f(\vec{x}^{(t)})$ to obtain the search direction. This is much more expensive than computing the gradient $\nabla f(\vec{x}^{(t)})$, which is used both in the gradient descent method and in Newton's method. However, this expensive step is not without justification; in many convex optimization problems, Newton's method can be shown to converge to the optimal solution much faster (i.e., in fewer iterations) than gradient descent.

10.3 Newton's Method with Linear Equality Constraints

Our derivation of Newton's method can be used to handle equality-constrained optimization problems. Let $A \in \mathbb{R}^{m \times n}$ and $\vec{y} \in \mathbb{R}^m$, and let $f \colon \mathbb{R}^n \to \mathbb{R}$ be a twice-differentiable strictly convex function with positive definite Hessian. Consider the following convex optimization problem:

$$p^\star = \quad \min_{\vec{x} \in \mathbb{R}^n} f(\vec{x}). \tag{10.40}$$

$$\text{s.t.} \quad A\vec{x} = \vec{y}. \tag{10.41}$$

We will use the same approach as with the unconstrained Newton's method, that is, we will take the second-order Taylor approximation around $\vec{x}^{(t)}$ and minimize it over the constraint set $\Omega \doteq \{\vec{x} \in \mathbb{R}^n \colon A\vec{x} = \vec{y}\}$. This method gives the following constrained convex quadratic program:

$$\min_{\vec{x} \in \mathbb{R}^n} \quad f(\vec{x}^{(t)}) + [\nabla f(\vec{x}^{(t)})]^\top (\vec{x} - \vec{x}^{(t)}) + \frac{1}{2}(\vec{x} - \vec{x}^{(t)})^\top [\nabla^2 f(\vec{x}^{(t)})](\vec{x} - \vec{x}^{(t)}) \tag{10.42}$$

$$\text{s.t.} \quad A\vec{x} = \vec{b}. \tag{10.43}$$

Note that the quadratic program is convex and, if the original problem is feasible (i.e., Ω is nonempty) that strong duality holds by Slater's condition. Thus, we can solve this QP by solving the KKT conditions, as they are necessary and sufficient for global optimality. We begin by writing the Lagrangian $L \colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ associated with this quadratic program, which is defined as

$$L(\vec{x}, \vec{\nu}) = f(\vec{x}^{(t)}) + [\nabla f(\vec{x}^{(t)})]^\top (\vec{x} - \vec{x}^{(t)}) + \frac{1}{2}(\vec{x} - \vec{x}^{(t)})^\top [\nabla^2 f(\vec{x}^{(t)})](\vec{x} - \vec{x}^{(t)}) + \vec{\nu}^\top (A\vec{x} - \vec{b}). \tag{10.44}$$

Suppose that $(\vec{x}^\star, \vec{\nu}^\star)$ are globally optimal for the constrained quadratic program. Then they must satisfy the KKT conditions, which are:

- Primal feasibility:

$$A\vec{x}^\star = \vec{y}. \tag{10.45}$$

- Stationarity/first-order condition:

$$\vec{0} = \nabla_{\vec{x}} L(\vec{x}^\star, \vec{\nu}^\star) \tag{10.46}$$
$$= \nabla f(\vec{x}^{(t)}) + [\nabla^2 f(\vec{x}^{(t)})](\vec{x}^\star - \vec{x}^{(t)}) + A^\top \vec{\nu}. \tag{10.47}$$

Let us define a vector $\vec{v}^{(t)} = \vec{x}^\star - \vec{x}^{(t)}$. Since $\vec{x}^{(t)}$ is feasible, we have $A\vec{x}^{(t)} = \vec{y}$. Thus we have

$$A\vec{v}^{(t)} = A(\vec{x}^\star - \vec{x}^{(t)}) \tag{10.48}$$
$$= A\vec{x}^\star - A\vec{x}^{(t)} \tag{10.49}$$
$$= \vec{y} - \vec{y} \tag{10.50}$$
$$= \vec{0}. \tag{10.51}$$

Thus, if we write the system in terms of $\vec{v}^{(t)}$ instead of $\vec{x}^\star$, we have the system of equations

$$\vec{0} = \nabla f(\vec{x}^{(t)}) + [\nabla^2 f(\vec{x}^{(t)})]\vec{v}^{(t)} + A^\top \vec{\nu}^\star \tag{10.52}$$
$$\vec{0} = A\vec{v}^{(t)} \tag{10.53}$$

which can be expressed in matrix form as

$$\begin{bmatrix} \nabla^2 f(\vec{x}^{(t)}) & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} \vec{v}^{(t)} \\ \vec{\nu} \end{bmatrix} = \begin{bmatrix} -\nabla f(\vec{x}^{(t)}) \\ 0 \end{bmatrix}. \tag{10.54}$$

After solving this system of equations for $\vec{v}^{(t)}$, our update rule becomes

$$\vec{x}^{(t+1)} = \vec{x}^\star = \vec{x}^{(t)} + \vec{v}^{(t)}, \tag{10.55}$$

which is equivalent to setting the new iterate as the minimizer of the constrained QP. The formal iteration is given in Algorithm 9.

Algorithm 9 Newton's Method with Linear Equality Constraints

function EQUALITYCONSTRAINEDNEWTONMETHOD($f, \vec{x}^{(0)}, T, A, \vec{y}$)
  for $t = 0, 1, \ldots, T-1$ **do**
    Solve the system $\begin{bmatrix} \nabla^2 f(\vec{x}^{(t)}) & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} \vec{v}^{(t)} \\ \vec{\nu} \end{bmatrix} = \begin{bmatrix} -\nabla f(\vec{x}^{(t)}) \\ 0 \end{bmatrix}$ for $\vec{v}^{(t)}$
    $\vec{x}^{(t+1)} \leftarrow \vec{x}^{(t)} + \vec{v}^{(t)}$
  end for
  return $\vec{x}^{(T)}$
end function

There also exist damped versions of this algorithm, but their analysis is out of scope of the course.

10.4 (OPTIONAL) Interior Point Method

In the previous section we introduced Newton's method, which allows us to solve convex optimization problems with simple linear equality constraints. In this section, we will build on top of Newton's method, introducing a new class

of algorithms to handle convex optimization problems with inequality constraints. Precisely, we will introduce the *interior point method*, which allows us to solve convex optimization problems of the following form

$$p^\star = \min_{\vec{x}\in\mathbb{R}^n} \quad f_0(\vec{x}) \tag{10.56}$$

$$\text{s.t.} \quad f_i(\vec{x}) \leq 0, \qquad \forall i = 1, 2, \ldots, m \tag{10.57}$$

$$A\vec{x} = \vec{y}, \tag{10.58}$$

where f_0, f_1, …, f_m are all convex twice-differentiable functions. Interior point methods (IPM) are a class of algorithms which solves the problem (10.56) by solving a sequence of convex optimization problems with linear constraints using Newton's method. The key idea used in IPM is the *barrier function*, which we introduce next.

10.4.1 Barrier Functions

Consider the problem (10.56). Our goal is to eliminate the inequality constraints and convert the problem to an equality constrained problem to which Newton's method can be applied. One way to do so is to augment the inequality constraints to the objective using indicator functions, as we did when developing our theory of duality. More precisely, consider the following unconstrained problem, which we denote by $\mathcal{P}$:

$$\text{problem } \mathcal{P}: \qquad p^\star = \min_{\vec{x}\in\mathbb{R}^n} \quad f_0(\vec{x}) + \sum_{i=1}^m I(f_i(\vec{x})) \tag{10.59}$$

$$A\vec{x} = \vec{b}. \tag{10.60}$$

where $I\colon \mathbb{R} \to \mathbb{R}\cup\{+\infty\}$ is given by

$$I(z) = \mathbb{1}[z \leq 0] = \begin{cases} 0, & \text{if } z \leq 0 \\ +\infty, & \text{if } z > 0 \end{cases} \tag{10.61}$$

This gives us an optimization problem with only linear equality constraints that is *equivalent* to the original optimization problem (10.56) (i.e., they have the same solution). However, introducing the indicator function now makes the objective function non-differentiable so we can no longer apply Newton's method to solve this problem. To overcome this problem, we will instead approximate the indicator function with a differentiable function ϕ, which we call a *barrier function*.

There are several choices for ϕ, i.e., good approximations for the indicator function, but they must all have something in common. Namely, ϕ should be a convex increasing function on $\mathbb{R}_{--}$, such that $\lim_{z\nearrow 0}\phi(z) = +\infty$, just like the indicator function I. There are many candidate functions that satisfy these criteria. One of the most used barrier functions that we will introduce here is the logarithmic barrier function, which, for some $\alpha > 0$, takes the form

$$\phi_\alpha(z) = -\frac{1}{\alpha}\log(-z), \tag{10.62}$$

The parameter α controls the accuracy of the approximation — as α grows larger, the logarithmic barrier function becomes a better and better approximation to the indicator function.

Using this logarithmic barrier, we can define an approximate optimization problem $\widehat{\mathcal{P}}(\alpha)$ to $\mathcal{P}$ by the following:

$$\text{problem } \widehat{\mathcal{P}}(\alpha): \qquad \widehat{p}^\star_\alpha = \min_{\vec{x}\in\mathbb{R}^n} \quad f_0(\vec{x}) + \sum_{i=1}^m \phi_\alpha(f_i(\vec{x})) \tag{10.63}$$

$$A\vec{x} = \vec{y}. \tag{10.64}$$

This optimization problem has a convex twice-differentiable objective function and linear equality constraints, so Newton's method can be applied.

10.4.2 Barrier Method

The problem $\widehat{\mathcal{P}}(\alpha)$ is just an approximation of the original problem $\mathcal{P}$. In particular, they do not necessarily have the same solution. Thus, the natural question to ask is how close the solutions of $\mathcal{P}$ and $\widehat{\mathcal{P}}(\alpha)$ are. The choice of the parameter α will be crucial, since small α mean that ϕ_α does not behave like an indicator function and so $\widehat{\mathcal{P}}(\alpha)$ and $\mathcal{P}$ are totally different in general, whereas large α offer much better approximations and thus allows us to expect much better solutions. Indeed, it is possible to show, with some additional assumptions [7], some more complicated bounds relating the distance of the solutions to $\mathcal{P}$ to the solutions to $\widehat{\mathcal{P}}(\alpha)$, as a function of α. We do not go into this analysis here, but its main takeaway is that large values of α give us good approximate solutions.

One natural question is: why don't we just take the largest possible α and get a near-perfect approximation to the indicator functions, and thus the original program $\mathcal{P}$? This actually may not be optimal from an algorithmic standpoint, as as solving problem $\widehat{\mathcal{P}}(\alpha)$ using Newton's method becomes difficult for large values of α. This is because Newton's method relies on computing and inverting the Hessian of the objective, and with large values of α the Hessian changes rapidly for points near the boundary of the feasible set for the original problem, even points which may be the solution.

The barrier method overcomes this problem by solving the approximate problem for a relatively small value of α and obtain the approximate solution $\vec{x}^\star(\alpha)$. The algorithm then refines this approximate solution by using it as an initial guess to solve the approximate problem with a larger value of α. This way, when the algorithm attempts to solve the difficult approximate problems $\widehat{\mathcal{P}}(\alpha)$ (as the value of α increases), it does so with a good initial guess. Newton's method converges extremely fast near the optimal solution, so this procedure greatly improves the convergence of Newton's method even when α is very large.[1]

When solving constrained optimization problems, it is necessary for most algorithms to start with a feasible initialization $\vec{x}^{(0)}$. This is particularly important for the barrier method; in fact, it is necessary to start at a strictly feasible initial guess, so that the value of the approximate problem is finite and the derivative of the objective function is defined. In Algorithm 10 we present one instance of the barrier method in which the value of α is updated by scaling it with some constant $\mu > 1$.

Algorithm 10 Barrier Method.

function BARRIERMETHOD($f_0, f_1, \ldots, f_m, \vec{x}^{(0)}, \alpha^{(0)}, A, \vec{y}, \mu, T$)
  for $t = 1, \ldots, T$ **do**
    $\vec{x}^{(t)} \leftarrow$ Solution of $\widehat{\mathcal{P}}(\alpha^{(t-1)})$ using Newton's method (Algorithm 9) starting at $\vec{x}^{(t-1)}$
    $\alpha^{(t)} \leftarrow \mu\alpha^{(t-1)}$
  end for
  return $\vec{x}^{(T)}$
end function

[1] This easy-to-hard solution process is one instance of a more general algorithmic paradigm called *homotopy continuation* or *homotopy analysis*, which is used to precisely simulate very unstable dynamical systems.

Chapter 11

Applications

In this chapter, we will discuss some applications of the theory we have developed so far in this class. Our exploration will include deterministic control and the linear-quadratic regulator, stochastic control and the policy gradient algorithm, and support vector machines.

11.1 (OPTIONAL) Deterministic Control and Linear-Quadratic Regulator

The first application we'll discuss is in the area of deterministic control.

Although control is usually thought of as related to motion, it can apply to any dynamical system where we can understand the state and its dependence on time based on some function such as $\vec{x}_{t+1} = \vec{f}(\vec{x}_t, \vec{u}_t)$ which takes in a state $\vec{x}_t$ and a control input $\vec{u}_t$ and outputs the next state $\vec{x}_{t+1}$. The goal of control is to choose the control inputs $\vec{u}_t$ to achieve some desired behavior of the system.

Example 214 (Vertical Rocket System)**.** For example, we can consider a vertical rocket. Our goal is to maximize its height by time T. Let $x_{1,t}$ denote its height, $x_{2,t}$ denote its vertical velocity, and $x_{3,t}$ denote the weight of the rocket (which we will approximate as the weight of the fuel), all at time t. The weight of the fuel will go down over time, and that can affect the rocket's velocity. The forces pushing the rocket down are drag and gravity, and the upward force comes from the rocket's thrust.

The forces at time t have the following expressions. (Here $\dot{x}$ is the time-derivative of x.)

- Inertial force: $x_{3,t} \cdot \ddot{x}_{1,t} = x_{3,t} \cdot \dot{x}_{2,t}$.
- Drag: $c_D \rho(x_{1,t}) x_{2,t}^2$ where c_D is a numerical constant and $\rho(x_1)$ is the density of the air at height x_1.
- Gravity: $g x_{3,t}$ where g is a numerical constant.
- Thrust: $c_H \dot{x}_{3,t}$ where c_H is a numerical constant.

Given the input $u_t = -\dot{x}_{3,t}$, we want to write our dynamical system in the standard form for continuous dynamics, i.e., $\dot{\vec{x}}_t = \vec{f}(\vec{x}_t, u_t)$. From the force expressions, we have

$$\dot{x}_{1,t} = x_{2,t} \tag{11.1}$$

$$\dot{x}_{2,t} = \frac{1}{x_{3,t}}[-c_D \rho(x_{1,t}) x_{2,t}^2 + c_H \dot{x}_{3,t} - g x_{3,t}] \tag{11.2}$$

$$\dot{x}_{3,t} = -u_t \tag{11.3}$$

Now we have a dynamical system of the form $\dot{\vec{x}}_t = \vec{f}(\vec{x}_t, u_t)$. Recall that we want to maximize the height of the rocket, $x_{1,T}$, at some terminal time $T > 0$, given an initial condition $\vec{x}_0 = \vec{\xi}$ for some $\vec{\xi} \in \mathbb{R}^3$. We can set up an optimization problem to determine the $(u_t)_{t\in[0,T]}$ which accomplishes this:

$$\begin{aligned} \max_{(u_t)_{t\in[0,T]}} \quad & x_{1,T} \\ \text{s.t.} \quad & \dot{\vec{x}}_t = \vec{f}(\vec{x}_t, u_t), \qquad \forall t \in [0,T] \\ & \vec{x}_0 = \vec{\xi}. \end{aligned}$$

In practice, we can use a numerical solver to solve this problem; conceptually one can solve many such systems by hand using the so-called *calculus of variations*, which is a sort of infinite-dimensional optimization paradigm which we do not explore more in this course.

Even though our dynamics are continuous-time and complex, we can discretize and locally linearize our system, obtaining an approximate system which is *discrete linear time-invariant*, i.e., of the form

$$\vec{x}_{k+1} = A\vec{x}_k + B\vec{u}_k, \qquad \forall k \in \{0, 1, \ldots, K-1\}. \tag{11.4}$$

where A and B are matrices of the appropriate sizes. This is a linear system because it is linear in $(\vec{x}_k)_{k=0}^K$ and $(\vec{u}_k)_{k=0}^{K-1}$, which we conceptually think of as very long (but finite-length) vectors. It is time-invariant because the matrices A and B do not depend on the discrete-time index k.

This particular type of control problem is ubiquitous within science and engineering, and thus deserves a special name — it is called the *linear quadratic regulator* problem.

Let's formally define our linear quadratic regulator system.

Definition 215 (Linear Quadratic Regulator (LQR))

Let $K \geq 0$ be a positive integer. Let $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $Q, Q_f \in \mathbb{S}_+^n$, and $R \in \mathbb{S}_{++}^m$. Let $\vec{\xi} \in \mathbb{R}^n$. A *linear quadratic regulator* is an optimization problem of the form:

$$\begin{aligned} \min_{\substack{(\vec{x}_k)_{k=0}^K \in \mathbb{R}^{(K+1)n} \\ (\vec{u}_k)_{k=0}^{K-1} \in \mathbb{R}^{Km}}} \quad & \frac{1}{2}\sum_{k=0}^{K-1} (\vec{x}_k^\top Q \vec{x}_k + \vec{u}_k^\top R \vec{u}_k) + \frac{1}{2}\vec{x}_K^\top Q_f \vec{x}_K \\ \text{s.t.} \quad & \vec{x}_{k+1} = A\vec{x}_k + B\vec{u}_k, \qquad \forall k \in \{0, \ldots, K-1\} \\ & \vec{x}_0 = \vec{\xi}. \end{aligned} \tag{11.5}$$

Equation (11.5) is actually a quadratic program, since the objective is quadratic in the variables $(\vec{x}_k)_{k=0}^K$ and $(\vec{u}_k)_{k=0}^{K-1}$, and the constraints are linear. We are able to solve this with the methods we already know. However, this problem is very large, having $(K+1)n + Km$ variables (as well as $K+1$) constraints, and for n, m, K large this quickly becomes intractable. Our saving grace is that this problem has significant *additional structure*. The traditional way to solve it is using the dynamic programming approach and Bellman's equation. However, in this section, we will solve it using the KKT conditions and the Riccati equation.

Theorem 216 (Optimal Control in LQR is Linear)

An optimal control for the LQR problem (11.5) is linear in the state, i.e.,

$$\vec{u}_k^\star = -R^{-1}B^\top(I + P_{k+1}BR^{-1}B^\top)^{-1}P_{k+1}A\vec{x}_k^\star\vec{x}_k^\star, \qquad \forall k \in \{0, \ldots, K-1\} \tag{11.6}$$

where P_k are given by the recurrence relation

$$P_K = Q_f \tag{11.7}$$
$$P_k = A^\top (I + P_{k+1} B R^{-1} B^\top)^{-1} P_{k+1} A + Q, \qquad \forall k \in \{0, \dots, K-1\}. \tag{11.8}$$

Proof. The Lagrangian of Equation (11.5) is

$$L((\vec{x}_k)_{k=0}^K, (\vec{u}_k)_{k=0}^{K-1}, (\vec{\lambda})_{k=1}^K, \vec{\nu}) = \frac{1}{2} \sum_{k=0}^{K-1} (\vec{x}_k^\top Q \vec{x}_k + \vec{u}_k^\top R \vec{u}_k) + \frac{1}{2} \vec{x}_K^\top Q_f \vec{x}_K \tag{11.9}$$
$$+ \sum_{k=0}^{K-1} \vec{\lambda}_{k+1}^\top (A\vec{x}_k + B\vec{u}_k - \vec{x}_{k+1}) + \vec{\nu}^\top (\vec{x}_0 - \vec{\xi}). \tag{11.10}$$

Since the objective function of Equation (11.5) is convex and the constraints are affine, Slater's condition holds automatically. Thus strong duality holds. Since Equation (11.5) is a convex problem and strong duality holds, the KKT conditions are necessary and sufficient for global optimality.

Let $((\vec{x}_k^\star)_{k=0}^K, (\vec{u}_k^\star)_{k=0}^{K-1}, (\vec{\lambda}_k^\star)_{k=1}^K, \vec{\nu}^\star)$ be globally optimal primal and dual variables for Equation (11.5), hence satisfying the KKT conditions. Then we have

1. Primal feasibility:
   (a) $\vec{x}_{k+1}^\star = A\vec{x}_k^\star + B\vec{u}_k^\star$ for all $k \in \{0, \dots, K-1\}$.
   (b) $\vec{x}_0^\star = \vec{\xi}$.
2. Dual feasibility: N/A since all constraints are equality constraints.
3. Complementary slackness: N/A since all constraints are equality constraints.
4. Lagrangian stationarity:
   (a) $\vec{0} = \nabla_{\vec{x}_0} L((\vec{x}_k^\star)_{k=0}^K, (\vec{u}_k^\star)_{k=0}^{K-1}, (\vec{\lambda}_k^\star)_{k=1}^K, \vec{\nu}^\star) = Q\vec{x}_0^\star + A^\top \vec{\lambda}_1^\star + \vec{\nu}^\star$.
   (b) $\vec{0} = \nabla_{\vec{x}_k} L((\vec{x}_k^\star)_{k=0}^K, (\vec{u}_k^\star)_{k=0}^{K-1}, (\vec{\lambda}_k^\star)_{k=1}^K, \vec{\nu}^\star) = Q\vec{x}_k^\star + A^\top \vec{\lambda}_{k+1}^\star - \vec{\lambda}_k^\star$ for all $k \in \{1, \dots, K-1\}$.
   (c) $\vec{0} = \nabla_{\vec{x}_K} L((\vec{x}_k^\star)_{k=0}^K, (\vec{u}_k^\star)_{k=0}^{K-1}, (\vec{\lambda}_k^\star)_{k=1}^K, \vec{\nu}^\star) = Q_f \vec{x}_K^\star - \vec{\lambda}_K^\star$.
   (d) $\vec{0} = \nabla_{\vec{x}_K} L((\vec{x}_k^\star)_{k=0}^K, (\vec{u}_k^\star)_{k=0}^{K-1}, (\vec{\lambda}_k^\star)_{k=1}^K, \vec{\nu}^\star) = R\vec{u}_k^\star + B^\top \vec{\lambda}_{k+1}^\star$ for all $k \in \{0, \dots, K-1\}$.

The stationarity condition gives an update dynamics for $\vec{\lambda}_k^\star$, i.e.,

$$\vec{\lambda}_k^\star = A^\top \vec{\lambda}_{k+1}^\star + Q\vec{x}_k^\star, \qquad \forall k \in \{1, \dots, K-1\}, \tag{11.11}$$
$$\vec{\lambda}_K^\star = Q_f \vec{x}_K^\star. \tag{11.12}$$

However, this update dynamics goes *backwards* in time from $k = K$ to $k = 1$. This is in contrast to the update dynamics for $\vec{x}_k$ which goes forwards in time.

When we find the $\vec{\lambda}_k^\star$, we are able to find the $\vec{u}_k^\star$, since a stationarity equation gives

$$\vec{u}_k^\star = -R^{-1} B^\top \vec{\lambda}_{k+1}^\star, \qquad \forall k \in \{0, \dots, K-1\}. \tag{11.13}$$

This motivates solving for $\vec{\lambda}_k^\star$, which we do via (backwards) induction. Our induction hypothesis is of the form

$$\vec{\lambda}_k^\star = P_k \vec{x}_k^\star, \tag{11.14}$$

which we aim to show for $k \in \{1, \dots, K\}$. The base case is $k = K$, whence we have $\vec{\lambda}_K^\star = Q_f \vec{x}_K^\star$, so that $P_K = Q_f$. For the inductive step, for $k \in \{1, \dots, K-1\}$ we have

$$\begin{aligned}
\vec{\lambda}_{k+1}^\star &= P_{k+1}\vec{x}_{k+1}^\star && (11.15)\\
&= P_{k+1}(A\vec{x}_k^\star + B\vec{u}_k^\star) && (11.16)\\
&= P_{k+1}(A\vec{x}_k^\star - BR^{-1}B^\top \vec{\lambda}_{k+1}^\star) && (11.17)\\
&= P_{k+1}A\vec{x}_k^\star - P_{k+1}BR^{-1}B^\top \vec{\lambda}_{k+1}^\star && (11.18)\\
(I + P_{k+1}BR^{-1}B^\top)\vec{\lambda}_{k+1}^\star &= P_{k+1}A\vec{x}_k^\star && (11.19)\\
\vec{\lambda}_{k+1}^\star &= (I + P_{k+1}BR^{-1}B^\top)^{-1}P_{k+1}A\vec{x}_k^\star. && (11.20)\\
\vec{\lambda}_k^\star &= A^\top \vec{\lambda}_{k+1}^\star + Q\vec{x}_k^\star && (11.21)\\
&= A^\top((I + P_{k+1}BR^{-1}B^\top)^{-1}P_{k+1}A\vec{x}_k^\star) + Q\vec{x}_k^\star && (11.22)\\
&= \underbrace{(A^\top(I + P_{k+1}BR^{-1}B^\top)^{-1}P_{k+1}A + Q)}_{\doteq P_k}\vec{x}_k^\star && (11.23)\\
&= P_k\vec{x}_k^\star. && (11.24)
\end{aligned}$$

Above, to show that (11.20) follows from (11.19), we need to confirm that $I + P_{k+1}BR^{-1}B^\top$ is invertible. To show this, we will explicitly construct its inverse in terms of the inverse of other matrices which we know are invertible. First, observe that $R + B^\top P_{k+1}B$ is invertible, since it is symmetric positive definite: R is symmetric positive definite and $B^\top P_{k+1}B$ is symmetric positive semidefinite, so their sum is symmetric positive definite. Next, we claim that $(I + P_{k+1}BR^{-1}B^\top)^{-1} = I - P_{k+1}B(R + B^\top P_{k+1}B)^{-1}B^\top$. This follows from the Sherman-Morrison-Woodbury identity, but for the sake of completeness we prove it here. Indeed,

$$\begin{aligned}
&(I + P_{k+1}BR^{-1}B^\top)(I - P_{k+1}B(R + B^\top P_{k+1}B)^{-1}B^\top) && (11.25)\\
&= I + P_{k+1}BR^{-1}B^\top - (I + P_{k+1}BR^{-1}B^\top)P_{k+1}B(R + B^\top P_{k+1}B)^{-1}B^\top && (11.26)\\
&= I + P_{k+1}BR^{-1}B^\top - P_{k+1}B(R + B^\top P_{k+1}B)^{-1}B^\top - P_{k+1}BR^{-1}B^\top P_{k+1}B(R + B^\top P_{k+1}B)^{-1}B^\top && (11.27)\\
&= I + P_{k+1}BR^{-1}B^\top - P_{k+1}BR^{-1}R(R + B^\top P_{k+1}B)^{-1}B^\top - P_{k+1}BR^{-1}B^\top P_{k+1}B(R + B^\top P_{k+1}B)^{-1}B^\top && (11.28)\\
&= I + P_{k+1}BR^{-1}B^\top - P_{k+1}BR^{-1}(R + B^\top P_{k+1}B)(R + B^\top P_{k+1}B)^{-1}B^\top && (11.29)\\
&= I + P_{k+1}BR^{-1}B^\top - P_{k+1}BR^{-1}B^\top && (11.30)\\
&= I. && (11.31)
\end{aligned}$$

This confirms that $I + P_{k+1}BR^{-1}B^\top$ is invertible. Thus we have for $k \in \{0, \dots, K-1\}$ that

$$\begin{aligned}
\vec{u}_k^\star &= -R^{-1}B^\top \vec{\lambda}_{k+1}^\star && (11.32)\\
&= -R^{-1}B^\top(I + P_{k+1}BR^{-1}B^\top)^{-1}P_{k+1}A\vec{x}_k^\star && (11.33)
\end{aligned}$$

as desired. □

Now, note that we have written our recurrence for P_k in terms of P_{k+1}. Starting from P_K, we can compute each P_k in a backwards order, completely offline (i.e., without processing any iterations of the forward system). Once we have the P_k, we can then compute each $\vec{x}_k$, $\vec{\lambda}_k$, and $\vec{u}_k$ directly. Therefore, we have a way to solve the LQR problem just by using matrix multiplication.

11.2 Support Vector Machines

Support vector machines are an application of optimization to the classical machine learning problem of binary classification. We will see that using optimization and a specific perspective induced by the KKT conditions will allow us to elegantly solve this classification problem.

Let's review the basics of classification. Suppose that we have some image, and we want to figure out what kind of animal it is. For example, we could be classifying it as a cat or a dog. Or, it could be X-ray images, and we want to figure out if there is a fracture present or not. Nowadays, we use a wide range of techniques such as deep neural networks to solve these tasks. These more advanced techniques build up from much more fundamental ideas such as support vector machines.

For the purpose of this class, we will focus on binary classification. In binary classification, we are given some data $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d$ as well as their corresponding labels $y_1, \ldots, y_n \in \{-1, +1\}$. We want to find a function $f \colon \mathbb{R}^d \to \{-1, 1\}$ such that $f(\vec{x}_i) = y_i$ for all i. We could try solving this problem using least squares, but in many cases the least squares solution is suboptimal at classification tasks, being too susceptible to outliers. Here, we will explore a different technique, called *support vector machines* (SVMs).

The basic premise of support-vector machines is that we want to find an *affine function* $g_{\vec{w},b} \colon \vec{x} \mapsto \vec{w}^\top \vec{x} - b$ which (approximately) separates the data into their respective classes, i.e., $y_i = \operatorname{sgn}(g_{\vec{w},b}(\vec{x}_i))$ for all i, so $f = \operatorname{sgn} \circ g_{\vec{w},b}$. Geometrically, $g_{\vec{w},b}$ can be thought of as a *separating hyperplane* for the two classes, in fact corresponding to the hyperplane $\mathcal{H}_{\vec{w},b} \doteq \{\vec{x} \in \mathbb{R}^d \mid \vec{w}^\top \vec{x} = b\}$.

If $y_i = 1$ then we would like $g_{\vec{w},b}(\vec{x}_i) > 0$. If $y_i = -1$ then we would like $g_{\vec{w},b}(\vec{x}_i) < 0$. We can express these two desires as always wanting $y_i g_{\vec{w},b}(\vec{x}_i) > 0$, combining these two cases into one inequality.

Hard-Margin SVM

First, let us work through the case that the data are *strictly linearly separable*; that is, there exists some $\vec{w}, b$ such that $y_i g_{\vec{w},b}(\vec{x}_i) > 0$ for all i. This hypothesis is unreasonable in many cases, but it allows us to build intuition for the problem. We will later remove this hypothesis, but many tools remain the same.

In this case, we would like to find one $(\vec{w}, b)$ pair for which $y_i g_{\vec{w},b}(\vec{x}_i) > 0$ for all i. However, there can be many such pairs. Thus, we have to determine which one we would like to pick.

One possible heuristic[1] is to pick a pair $(\vec{w}, b)$ with the largest *margin* — that is, the distance between the hyperplane $\mathcal{H}_{\vec{w},b}$ and the closest point towards it. Thus we want to solve the problem:

$$\begin{aligned} \max_{\substack{\vec{w} \in \mathbb{R}^d \\ b \in \mathbb{R}}} \quad & \min_{i \in \{1,\ldots,n\}} \operatorname{dist}(\mathcal{H}_{\vec{w},b}, \vec{x}_i) \\ \text{s.t.} \quad & y_i g_{\vec{w},b}(\vec{x}_i) > 0, \qquad \forall i \in \{1, \ldots, n\}. \end{aligned} \tag{11.34}$$

This problem is called the *hard-margin SVM*, so named because we do not allow any misclassification of the training points — no training data can "cross the margin."

Unfortunately, the problem in Equation (11.34) seems intractable. Let us go about simplifying it. First, the distance between point $\vec{x}$ and $\mathcal{H}_{\vec{w},b}$ is defined as

$$\operatorname{dist}(\mathcal{H}_{\vec{w},b}, \vec{x}) \doteq \frac{\left|\vec{w}^\top \vec{x} - b\right|}{\|\vec{w}\|_2} = \frac{|g_{\vec{w},b}(\vec{x})|}{\|\vec{w}\|_2}. \tag{11.35}$$

[1] This heuristic makes sense from the perspective of robustness and generalization. We haven't sampled all the data that exists, but we hypothesize that the data we haven't sampled is geometrically close to the data we have sampled. We want to make sure that the largest fraction possible of all data (sampled and unsampled) is correctly classified by the $\vec{w}$ and b we learn. Thus, to capture the most unsampled data possible, we require the classification to be as robust to these geometric deviations as possible.

Thus[2] we obtain the slightly simplified, yet equivalent, problem

$$\begin{aligned} \max_{\substack{\vec{w}\in\mathbb{R}^d \\ b\in\mathbb{R}}} \quad & \frac{1}{\|\vec{w}\|_2} \min_{i\in\{1,\dots,n\}} |g_{\vec{w},b}(\vec{x}_i)| \\ \text{s.t.} \quad & y_i g_{\vec{w},b}(\vec{x}_i) > 0, \qquad \forall i \in \{1,\dots,n\}. \end{aligned} \tag{11.36}$$

Adding the real-valued slack variable s to denote the minimizing component, we have

$$\begin{aligned} \max_{\substack{\vec{w}\in\mathbb{R}^d \\ b\in\mathbb{R} \\ s\in\mathbb{R}_{++}}} \quad & \frac{s}{\|\vec{w}\|_2} \\ \text{s.t.} \quad & y_i g_{\vec{w},b}(\vec{x}_i) > 0, \qquad \forall i \in \{1,\dots,n\}, \\ & |g_{\vec{w},b}(\vec{x}_i)| \geq s, \qquad \forall i \in \{1,\dots,n\}. \end{aligned} \tag{11.37}$$

Since $y_i \in \{-1,+1\}$ and $s > 0$, we have that for *any* scalar u_i that $|u_i| \geq s$ and $y_i u_i > 0$ if and only if $y_i u_i \geq s$. This relation certainly holds for $u_i = g_{\vec{w},b}(\vec{x}_i)$, and so the above problem simplifies to the following:

$$\begin{aligned} \max_{\substack{\vec{w}\in\mathbb{R}^d \\ b\in\mathbb{R} \\ s\in\mathbb{R}_{++}}} \quad & \frac{s}{\|\vec{w}\|_2} \\ \text{s.t.} \quad & y_i g_{\vec{w},b}(\vec{x}_i) \geq s, \qquad \forall i \in \{1,\dots,n\}. \end{aligned} \tag{11.38}$$

By the form of $g_{\vec{w},b}(\vec{x}) = \vec{w}^\top \vec{x} - b$, the above problem is equivalent to the following problem:

$$\begin{aligned} \max_{\substack{\vec{w}\in\mathbb{R}^d \\ b\in\mathbb{R} \\ s\in\mathbb{R}_{++}}} \quad & \frac{1}{\|\vec{w}/s\|_2} \\ \text{s.t.} \quad & y_i g_{\vec{w}/s,b/s}(\vec{x}_i) \geq 1, \qquad \forall i \in \{1,\dots,n\}. \end{aligned} \tag{11.39}$$

Since the above problem only depends on the values of $\vec{w}/s$ and b/s, where $\vec{w}/s$ is still an optimization variable allowed to take arbitrary values in $\mathbb{R}^d$ and b/s is still an optimization variable allowed to take arbitrary values in $\mathbb{R}$, we may as well remove s by setting it to 1, obtaining

$$\begin{aligned} \max_{\substack{\vec{w}\in\mathbb{R}^d \\ b\in\mathbb{R}}} \quad & \frac{1}{\|\vec{w}\|_2} \\ \text{s.t.} \quad & y_i g_{\vec{w},b}(\vec{x}_i) \geq 1, \qquad \forall i \in \{1,\dots,n\}. \end{aligned} \tag{11.40}$$

Finally, we can obtain an equivalent problem which is a convex minimization problem by using the transformation $x \mapsto \frac{1}{2x^2}$, which is monotonically decreasing on $\mathbb{R}_{++}$, on the objective function, whence (after expanding the form of $g_{\vec{w},b}$), we obtain the problem

$$\begin{aligned} \min_{\substack{\vec{w}\in\mathbb{R}^d \\ b\in\mathbb{R}}} \quad & \frac{1}{2}\|\vec{w}\|_2^2 \\ \text{s.t.} \quad & y_i(\vec{w}^\top \vec{x}_i - b) \geq 1, \qquad \forall i \in \{1,\dots,n\}. \end{aligned} \tag{11.41}$$

The problem (11.41) is by far the most common and simplified form of the hard-margin SVM problem. Moreover, this is a *quadratic program* in $(\vec{w}, b)$, which we know how to solve.

[2] It may seem natural to make $g_{\vec{w},b}$ more complex and thus obtain a recipe for learning more complicated types of classifiers, but note that the above simplification no longer holds if we do, so the final problem may be much more complex and not efficiently solvable.

Soft Margin SVM

Most of our real world data isn't actually linearly separable. In that case, our hard margin SVM problem would just be infeasible. But maybe we still want to try to separate the points, even if our classifier is not perfect on the training data.

To develop this relaxed SVM problem, let us consider the hard-margin SVM:

$$\begin{aligned} \min_{\substack{\vec{w}\in\mathbb{R}^d \\ b\in\mathbb{R}}} \quad & \frac{1}{2}\|\vec{w}\|_2^2 \\ \text{s.t.} \quad & y_i g_{\vec{w},b}(\vec{x}_i) \geq 1, \qquad \forall i \in \{1,\dots,n\}. \end{aligned} \tag{11.42}$$

It is a constrained optimization problem, so we can reformulate it into an unconstrained problem using indicator variables:

$$\min_{\substack{\vec{w}\in\mathbb{R}^d \\ b\in\mathbb{R}}} \left(\frac{1}{2}\|\vec{w}\|_2^2 + \sum_{i=1}^n \ell_{0-\infty}(1 - y_i g_{\vec{w},b}(\vec{x}_i)) \right) \tag{11.43}$$

where the $\ell_{0-\infty}$ loss is defined by

$$\ell_{0-\infty}(z) = \begin{cases} 0, & z \leq 0 \\ \infty, & z > 0. \end{cases} \tag{11.44}$$

This corresponds to infinitely penalizing any violation of the margin. In order to relax the penalties for margin violation, we can instead consider finite penalties which increase as the degree of violation of the margin increase. Namely, we can consider constant multiples of the so-called *hinge* loss

$$\ell_{\text{hinge}}(z) = \begin{cases} 0, & z \leq 0 \\ z, & z > 0 \end{cases} = \max\{z, 0\}. \tag{11.45}$$

This relaxation yields the following program:

$$\min_{\substack{\vec{w}\in\mathbb{R}^d \\ b\in\mathbb{R}}} \left(\frac{1}{2}\|\vec{w}\|_2^2 + C\sum_{i=1}^n \ell_{\text{hinge}}(1 - y_i g_{\vec{w},b}(\vec{x}_i)) \right), \tag{11.46}$$

or using the definition of the hinge loss, the equivalent program

$$\min_{\substack{\vec{w}\in\mathbb{R}^d \\ b\in\mathbb{R}}} \left(\frac{1}{2}\|\vec{w}\|_2^2 + C\sum_{i=1}^n \max\{1 - y_i g_{\vec{w},b}(\vec{x}_i), 0\} \right). \tag{11.47}$$

We can introduce some slack variables $\vec{\xi}$ to model this maximization term and make the problem differentiable. After expanding the form of $g_{\vec{w},b}$, we have the program

$$\min_{\substack{\vec{w}\in\mathbb{R}^d \\ b\in\mathbb{R} \\ \vec{\xi}\in\mathbb{R}^n}} \quad \frac{1}{2}\|\vec{w}\|_2^2 + C\sum_{i=1}^n \xi_i \tag{11.48}$$

$$\text{s.t.} \quad \xi_i \geq 0, \qquad \forall i \in \{1,\dots,n\} \tag{11.49}$$

$$\xi_i \geq 1 - y_i(\vec{w}^\top \vec{x}_i - b), \qquad \forall i \in \{1,\dots,n\}. \tag{11.50}$$

This is the usual form of the *soft margin SVM* problem, and the solutions are parameterized by C.

In accordance with our derivation, if C is large then we allow only small violations to the margin, because the second term becomes a better approximation of the sum of indicators in Equation (11.43). If C is small, then we allow larger violations to the margin.

Depending on the perspective, either the first term or the second term of the loss can be viewed as the regularizer. The first term works to maximize the margin, while the second term works to penalize the margin violations. Both work together to form an approximate maximum-margin classifier.

KKT Conditions

It turns out that we get significant insight into the solutions to the hard-margin and soft-margin SVM using the KKT conditions. First, let us consider the hard-margin SVM:

$$\begin{aligned} \min_{\substack{\vec{w}\in\mathbb{R}^d \\ b\in\mathbb{R}}} \quad & \frac{1}{2}\|\vec{w}\|_2^2 \\ \text{s.t.} \quad & y_i(\vec{w}^\top \vec{x}_i - b) \geq 1, \qquad \forall i \in \{1,\dots,n\}. \end{aligned} \tag{11.51}$$

The Lagrangian is

$$L(\vec{w}, b, \vec{\lambda}) = \frac{1}{2}\|\vec{w}\|_2^2 + \sum_{i=1}^n \lambda_i (1 - y_i(\vec{w}^\top \vec{x}_i - b)). \tag{11.52}$$

This problem is convex, and the constraints are affine, so if the problem is feasible (i.e., the data are strictly linearly separable), then Slater's condition holds so strong duality holds. Since the problem is convex and strong duality holds, the KKT conditions are necessary and sufficient for optimality.

Suppose that $(\vec{w}^\star, b^\star, \vec{\lambda}^\star)$ satisfy the KKT conditions. Then:

1. Primal feasibility: $y_i((\vec{w}^\star)^\top \vec{x}_i - b^\star) \geq 1$ for all $i \in \{1,\dots,n\}$.
2. Dual feasibility: $\lambda_i^\star \geq 0$ for all $i \in \{1,\dots,n\}$.
3. Stationarity:
   (a) $\vec{0} = \nabla_{\vec{w}} L(\vec{w}^\star, b^\star, \vec{\lambda}^\star) = \vec{w}^\star - \sum_{i=1}^n \lambda_i^\star y_i \vec{x}_i$.
   (b) $0 = \nabla_b L(\vec{w}^\star, b^\star, \vec{\lambda}^\star) = \sum_{i=1}^n \lambda_i^\star y_i$.
4. Complementary slackness: $\lambda_i^\star(1 - y_i((\vec{w}^\star)^\top \vec{x}_i - b)) = 0$ for all $i \in \{1,\dots,n\}$.

We say that $(\vec{x}_i, y_i)$ is a *support vector* if $\lambda_i^\star > 0$. To see why, we consider the following cases:

1. If $\lambda_i^\star = 0$ then, since $\vec{w}^\star = \sum_{i=1}^n \lambda_i^\star y_i \vec{x}_i$, we see that $(\vec{x}_i, y_i)$ does not contribute to the optimal solution.
2. If $\lambda_i^\star > 0$ then by complementary slackness we have $y_i((\vec{w}^\star)^\top \vec{x}_i - b) = 1$. Thus $\vec{x}_i$ is on the margin of the SVM. Furthermore, since $\vec{w}^\star = \sum_{i=1}^n \lambda_i^\star y_i \vec{x}_i$, we see that $(\vec{x}_i, y_i)$ does contribute to the optimal solution.

Now let us consider the analogous notion for soft-margin SVMs. Consider the soft-margin SVM problem:

$$\min_{\substack{\vec{w}\in\mathbb{R}^d \\ b\in\mathbb{R} \\ \vec{\xi}\in\mathbb{R}^n}} \quad \frac{1}{2}\|\vec{w}\|_2^2 + C\sum_{i=1}^n \xi_i \tag{11.53}$$

$$\text{s.t.} \quad \xi_i \geq 0, \qquad \forall i \in \{1,\dots,n\} \tag{11.54}$$

$$\xi_i \geq 1 - y_i(\vec{w}^\top \vec{x}_i - b), \qquad \forall i \in \{1,\dots,n\}. \tag{11.55}$$

It has Lagrangian

$$L(\vec{w}, b, \vec{\xi}, \vec{\lambda}, \vec{\mu}) = \frac{1}{2}\|\vec{w}\|_2^2 + C\sum_{i=1}^n \xi_i + \sum_{i=1}^n \mu_i(-\xi_i) + \sum_{i=1}^n \lambda_i(1 - y_i(\vec{w}^\top \vec{x}_i - b) - \xi_i). \tag{11.56}$$

This problem is convex, and the constraints are affine; one can show that it is always feasible, so Slater's condition holds and strong duality holds. Since the problem is convex with strong duality, the KKT conditions are both necessary and sufficient for optimality.

Suppose that $(\vec{w}^\star, b^\star, \vec{\xi}^\star, \vec{\lambda}^\star, \vec{\mu}^\star)$ satisfy the KKT conditions. Then:

1. Primal feasibility: $\xi_i^\star \geq 0$ and $\xi_i^\star \geq 1 - y_i((\vec{w}^\star)^\top \vec{x}_i - b^\star)$ for all $i \in \{1, \ldots, n\}$.
2. Dual feasibility: $\lambda_i^\star \geq 0$ and $\mu_i^\star \geq 0$ for all $i \in \{1, \ldots, n\}$.
3. Stationarity:
   (a) $\vec{0} = \nabla_{\vec{w}} L(\vec{w}^\star, b^\star, \vec{\xi}^\star, \vec{\lambda}^\star, \vec{\mu}^\star) = \vec{w}^\star - \sum_{i=1}^n \lambda_i^\star y_i \vec{x}_i$.
   (b) $0 = \nabla_b L(\vec{w}^\star, b^\star, \vec{\xi}^\star, \vec{\lambda}^\star, \vec{\mu}^\star) = \sum_{i=1}^n \lambda_i^\star y_i$.
   (c) $\vec{0} = \nabla_{\vec{\xi}} L(\vec{w}^\star, b^\star, \vec{\xi}^\star, \vec{\lambda}^\star, \vec{\mu}^\star) = C\vec{1} - \vec{\mu}^\star - \vec{\lambda}^\star$.
4. Complementary slackness: $\lambda_i^\star(1 - y_i((\vec{w}^\star)^\top \vec{x}_i - b) - \xi_i^\star) = 0$ and $\mu_i^\star \xi_i^\star = 0$ for all $i \in \{1, \ldots, n\}$.

Similarly to the case of the hard-margin SVM, we say that $(\vec{x}_i, y_i)$ is a *support vector* if $\lambda_i > 0$. To see why, we consider the following cases:

1. If $\lambda_i^\star = 0$ then $\mu_i^\star = C$. Thus $\mu_i^\star > 0$, so $\xi_i^\star = 0$. Thus the point $\vec{x}_i$ does not violate the margin. Also since $\vec{w}^\star = \sum_{i=1}^n \lambda_i^\star y_i \vec{x}_i^\star$, we see that $(\vec{x}_i, y_i)$ does not contribute to the optimal solution.
2. If $\lambda_i^\star = C$ then $\mu_i^\star = 0$. Thus we cannot say anything about $\xi_i^\star$. But since $\lambda_i^\star > 0$, the other complementary slackness condition says that $y_i((\vec{w}^\star)^\top \vec{x}_i - b) = 1 - \xi_i^\star$. Thus $(\vec{x}_i, y_i)$ is either on the margin or violates the margin. Also since $\vec{w}^\star = \sum_{i=1}^n \lambda_i^\star y_i \vec{x}_i^\star$, we see that $(\vec{x}_i, y_i)$ does contribute to the optimal solution.
3. If $\lambda_i^\star \in (0, C)$, then $\mu_i^\star \in (0, C)$ as well. Thus by complementary slackness, we have $\xi_i^\star = 0$. Applying this to the other complementary slackness condition, we have $y_i((\vec{w}^\star)^\top \vec{x}_i - b) = 1$. Thus $(\vec{x}_i, y_i)$ is exactly on the margin. Also since $\vec{w}^\star = \sum_{i=1}^n \lambda_i^\star y_i \vec{x}_i^\star$, we see that $(\vec{x}_i, y_i)$ does contribute to the optimal solution.

In general, the support vectors contribute to the optimal solution, and they are on/violate the margin.

Bibliography

[1] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[2] G. Calafiore and L. El Ghaoui, *Optimization Models*. Cambridge University Press, 2014.

[3] C. C. Pugh, *Real Mathematical Analysis*. Springer, 2002, vol. 2011.

[4] P. Varaiya *et al.*, *Lecture notes on optimization*. Unpublished manuscript, University of California, Department of Electrical Engineering and Computer Science, 1998.

[5] D. P. Bertsekas, "Nonlinear programming," *Journal of the Operational Research Society*, vol. 48, no. 3, pp. 334–334, 1997.

[6] G. Garrigos and R. M. Gower, "Handbook of convergence theorems for (stochastic) gradient methods," *arXiv preprint arXiv:2301.11235*, 2023.

[7] Y. Nesterov *et al.*, *Lectures on convex optimization*. Springer, 2018, vol. 137.