Optimization I: Theory and Analytical Methods

Nathan Ratliff

Dec 5, 2014

Abstract

Optimization is one of the most critical components of intelligent systems. Google uses optimization to learn search rankings and registering street view representations; financial analysts and hedge funds optimize strategies to make lots and lots of money; the world's most sophisticated robots, especially big bulky humanoids, are built around optimizers for control and motion generation, and even physical optimality principles that define their underlying dynamic models. This document presents first- and second-order criteria for understanding what's required of an optimal solution, and discusses their application to analytically solving various forms of optimization problems that arise frequently in practice. The material here in such a short document necessarily can only skim the surface of the available and widely applicable literature on optimization. One garners a deep understanding only through multiple classes, many books, and years of practical experience. But the analytical tools presented here are the building blocks for much of the theory, and they're a good introduction to this broad and fascinating area of study.

1 Optimization in Intelligent Systems

Optimization is one of the most widely applicable topics you can find, just in general; it's used throughout both academia and industry, from material optimization and manufacturing, to economics and product distribution, and of course it's used heavily in all sorts of intelligent systems. Increasingly, behavior generation, probabilistic inference, games, and most of machine learning, are built atop optimization algorithms. It's a safe bet that an investment in learning some of the basic principles of optimization will be useful almost anywhere you go.

Discussing in depth even just the theoretical and algorithmic foundations of the material we'll be covering here could fill entire books, so this presentation will necessarily focus on only the most basic of techniques and practical application. But we focus on the most important components of the theory and build our ideas to emphasize intuition using the geometric analysis tools we've been developing in Ratliff (2014c,d). We present enough to be practically useful

for many analytical optimization applications, and it should be a good starting point for further study in this area.

1.1 Optimization as a building block for intelligence

Optimization is a critical component to the design of most intelligent system. Whether we want to find a hypothesis that best fits a set of data in a setting we typically characterize as machine learning, or we want to choose a good sequence of actions for a robot that must react to a complicated, uncertain, or even adversarial environment, intelligent systems tend to be best understood as optimization processes that leverage the structure of problems to find the best of *something*. What that something is, and what the properties of that *best* something are once we find it, is the stuff of robotics, machine learning, statistics, decision theory, operations research, and the likes. But all of these fields, by their very nature, rely on the theory of optimization and optimization algorithms.

1.2 Optimality principles in physics

We usually think of optimization as a process: we have an algorithm that searches around the domain, iteratively refining its hypothesis, until it's converged on some point that seems to be minimizing the objective at least within a region of the valid domain. But optimization is much more broad than that. It doesn't have to be just a tool for solving problems; it's often used as a principle in itself. In physics, for instance, physical laws are often represented as **optimality principles**. We'll see one instance of this in Section 4.3, but in general such an optimality principle is a statement that defines the true behavior of a system as an optimal (or critical) point of some objective.

An example is Lagrangian Mechanics. In Lagrangian Mechanics, the Principle of Stationary Action states that the real physical behavior of a system is a critical point of the system's "action" in the sense that it satisfies the first-order optimality conditions (defined below) of this action. The action is a function defined on the space of differentiable trajectories $\boldsymbol{q} : \mathbb{R} \to \mathbb{R}^d$ of the form

$$\mathcal{A}[\boldsymbol{q}] = \int_0^T \Big(\mathcal{T}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \mathcal{V}(\boldsymbol{q}) \Big) \, dt \tag{1}$$

where $\mathcal{T}$ and $\mathcal{V}$ are the kinetic and potential energies of the system at a given moment in time. This objective is actually a *functional* since it's a function of a trajectory, but the basic principles for analyzing the first-order optimality conditions of functionals are directly analogous to finite-dimensional the principles we discuss below.

This principle allows us to write down in a single expression everything we need to know to derive the equations of motion of the dynamical system. The equations, themselves, which are typically a much more verbose (although computationally practical) representation, are derived by explicitly calculating these

first-order optimality conditions. Importantly, this principle, since it's built on optimality, is agnostic to the particular choice of curvilinear coordinate system. That means we can easily compute the equations in arbitrary coordinate systems, whichever is most convenient, without much added pain. Direct calculation, on the other hand, using, or instance, Newton's laws of motion, is much more difficult in practice, especially when we start changing the coordinate system and introducing fictitious forces that arise solely from the coordinate (e.g. centrifugal or Coriolis forces).

As an example, if we model a robot as a set of rigid bodies connected together by a collection of joints, it's easy to write down the what resulting kinetic and potential energies of the system are. Then using the above principle and analysis techniques analogous to those we present below, we can derived that the behavior of the (unconstrained) system always takes the form

$$\boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{h} = \boldsymbol{\tau}, \tag{2}$$

where $\boldsymbol{M}$ is a positive definite matrix describing how the system's *inertia* manifests in coordinates of the joints, $\boldsymbol{h}$ is a vector typically depending on $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ that represents the fictitious forces induced by the coordinate system, and $\boldsymbol{\tau}$ is a vector of forces applied to the joints. These equations are the basic building blocks of modern control methods used on many of the world's most sophisticated robots; they're made possible by the types of analytical optimization tools presented below.

We won't discuss this principle any further, but it's an example of where analytical optimization offers theoretical insight into a complicated problem and ultimately significantly simplifies what would otherwise be a very tedious calculation. The lectures of Susskind (2011) give an excellent introduction to this intriguing area of physics. See also Ratliff (2014a).

2 Differentiable optimization problems

There are many different types of optimization problems we might find in various applications, and Section 7 mentions some of the broad classes, but here we'll introduce a specific, very common class of problems, that have been thoroughly studied. These problems are build around assumptions of first- and second-order differentiability which, as we've seen, gives us a lot of information about the analytical structure of the problem.

In general, an **unconstrained objective** is simply a function of the form $f : \mathbb{R}^n \to \mathbb{R}$ that we want to somehow minimize or maximize. In this document, we'll focus primarily on minimization, but all of what we discuss can equally well be adapted to maximization.

Formally, a **local minimum** of a function f is a point $\boldsymbol{x}^*$ in the domain around which there is a neighborhood, i.e. a subset of the domain $\mathcal{N} \in \mathbb{R}^n$ containing the point in question $\boldsymbol{x}^* \in \mathcal{N}$, on which the function never drops

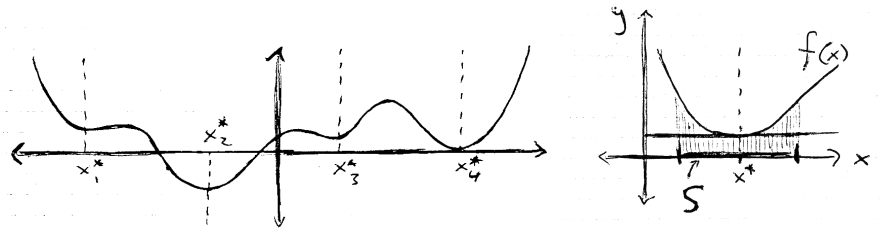

Figure 1: **Left:** The function may have multiple local minima throughout the domain. **Right:** A local minimum is a point around which there is a region $\mathcal{S}$ that never takes on cost smaller than the value at the local minimum.

below the value $f(\boldsymbol{x}^*)$. Specifically,

$$f(\boldsymbol{x}) \geq f(\boldsymbol{x}^*) \quad \text{for all} \quad \boldsymbol{x} \in \mathcal{N}.$$

This equation is just a mathematical description of a fairly intuitive idea: a local minimum is just a point that's at the bottom of a bowl-like region of the function (see Figure 1).

Generally, there may be many local optima throughout the function (see Figure 1), and the problem of finding the **global optimum**, i.e. specifically the smallest of those local optima, can be very very hard. That problem is an active area of modern research, leveraging even decision theoretical techniques for iteratively choosing where to search next for a possibly better local optimum. This document focuses on only the *local* optimization problem, which is usually a building block for more extensive global optimization techniques.

So far, we've said nothing about differentiability of the function, or even continuity. We could define strangely degenerate functions that might be smooth and bounded below in a region $\mathcal{N} \in \mathbb{R}^n$ where $v_{\min} \leq f(\boldsymbol{x})$ for all points $\boldsymbol{x} \in \mathcal{N}$ *except* a single point $\boldsymbol{x}^*$. There it inexplicably jumps down to some tiny value $f(\boldsymbol{x}^*) \ll v_{\min}$. Clearly, this point $\boldsymbol{x}^*$ is a local minimum in the neighborhood $\mathcal{N}$, but there's no real hope of finding it since there's no signal or indication that it's there unless we serendipitously stumble across it (which actually has zero probability of happening if we randomly sample from any smooth distribution across the domain). We need some sort of structure to the problem in order to make any progress at all in solving it.

Differentiability provides a very powerful structural assumption. Specifically, it allows us to make second-order Taylor approximations of the objective around any point $\boldsymbol{x}'$. These approximations take the form

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}') + \nabla f(\boldsymbol{x}')^T(\boldsymbol{x} - \boldsymbol{x}') + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^T \nabla^2 f(\boldsymbol{x}')(\boldsymbol{x} - \boldsymbol{x}') \qquad (3)$$

and describe to the second order how the function looks around the point $\boldsymbol{x}'$. As we see in the next section, this expansion gives us all the information we need to fully characterize whether or not a point $\boldsymbol{x}'$ is locally optimal.

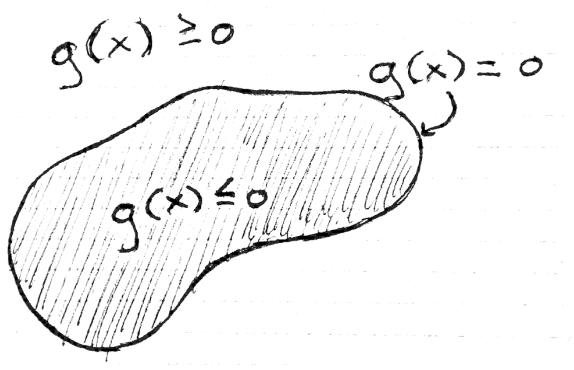

Figure 2: This diagram shows how an function $g(\boldsymbol{x})$ delineates three separate regions of the space. $g(\boldsymbol{x}) = 0$ is a hypersurface, $g(\boldsymbol{x}) \leq 0$ is the region enclosed by the hypersurface, and $\boldsymbol{g}(\boldsymbol{x}) > 0$ is the region outside the hypersurface.

Now, the most general problem we'll be discussing in this document is an optimization of an objective $f(\boldsymbol{x})$ over some *constrained* domain $\mathcal{S} \subset \mathbb{R}^n$. Again, if we know nothing about the structure of $\mathcal{S}$, then we can't really do anything but optimize f as though it were unconstrained and then worry really hard[1] if we ever find ourselves outside of $\mathcal{S}$. So we generally assume that our set has a nice representation as a differentiable surface in the space, or a region delineated by such a differentiable surface.

We saw in Ratliff (2014d) two ways to represent surfaces: implicitly and explicitly. Implicit surfaces, which are surfaces represented as the zero set $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}$ of some function $\boldsymbol{h} : \mathbb{R}^n \to \mathbb{R}^k$, are very nice representations for optimization problems because they naturally can be modified to additionally represent entire regions segmented from the rest of the space by turning the equality into an inequality (see Figure 2). The resulting mathematics is similar for both types of constraint regions as we'll see in detail in Section 4.

Using both types of constraints leads to a very general representation of a optimization problem

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0} \qquad \text{where} \quad \boldsymbol{h} : \mathbb{R}^n \to \mathbb{R}^k \\ & \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0} \qquad \text{where} \quad \boldsymbol{g} : \mathbb{R}^n \to \mathbb{R}^l, \end{aligned} \tag{4}$$

Problems of this sort are called *Nonlinear Programming Problems*. f is the objective, $\boldsymbol{h}$ compactly denotes k equality constraints as separate outputs of a vector valued function, and $\boldsymbol{g}$ compactly denotes l separate inequality constraints.

[1] There are formal algorithms for dealing with such constraints, but they're nowhere near as efficient as algorithms we can develop if we know something more about the structure of the set $\mathcal{S}$, so we focus here on what we can do when we do have a good handle on $\mathcal{S}$.

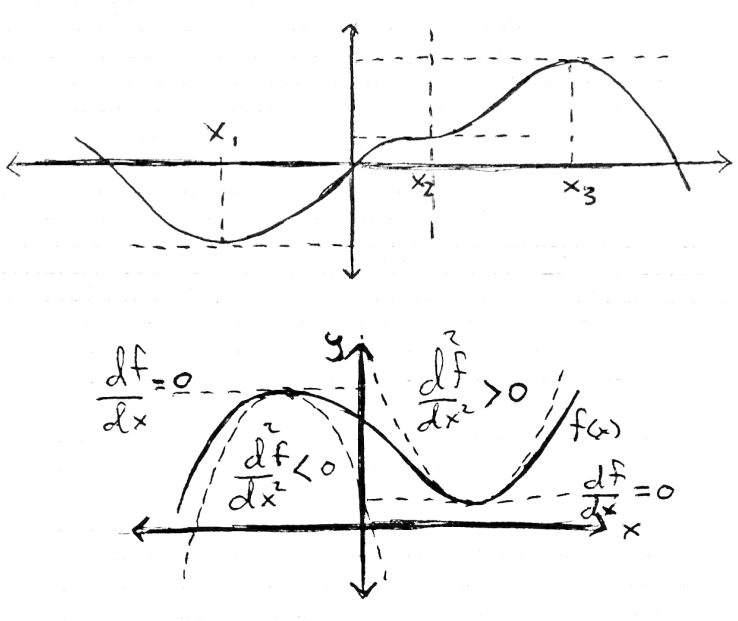

Figure 3: **Top:** There are three types of critical points in one-dimension, local minima ($\boldsymbol{x}_1$), local maxima ($\boldsymbol{x}_3$), and neither ($\boldsymbol{x}_2$). Each are distinguished by the value of the second derivative. **Bottom:** First and second-order information for local minima (right) and local maxima (left).

Just as we've found quadratic functions to admit straightforward analytical optimization, we'll find that a quadratic objective in conjunction with linear equality and inequality functions result in a particularly simple form of this problem called a **Quadratic Program**. Sections 4 and 5 show that we can often exploit the constrained optimality criteria that we develop below to solve many such problems analytically.

We'll also consider constraint surfaces with explicit parameterizations in Section 4 both because it's sometimes convenient to use such a parameterization and to better understand the geometry of constrained optimality conditions. From here on out, we always assume any objective function or constraint is second-order differentiable.

3 Unconstrained optimality criteria

We can gain a lot of insight into the optimality criteria for *unconstrained* optimization by considering the simple one-dimensional case. Univariate calculus studied critical points of a one-dimensional function, defined as the points where the first-derivative vanished. We easily saw that it wasn't enough to just con-

sider that condition alone when searching for a local minimum. There were three possible types of critical points, distinguishable only by the *second* derivative as shown in Figure 3. The function could actually be a local minimum, as indicated by a strictly positive second derivative. But it could also have a strictly *negative* second derivative and be curving down, which is exactly the opposite of what we're looking for. And, the final case, it could also be that the second derivative is exactly zero. In that case, we can only understand the broader implications of the behavior if we look at higher order derivatives. For instance, as shown in the figure, the point could be curing up in one direction, but curving down in the other.

All of these these conditions carry over to our multi-dimensional setting, with just one caveat. Now, since there are multiple dimensions, we can have different second-order behavior in different dimensions. We explore these in what follows.

Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is our objective. The second-order Taylor expansion says that as we move in a direction $\delta\boldsymbol{x}$ away from a point $\boldsymbol{x}$, the function behaves to the second order as

$$f(\boldsymbol{x} + \delta\boldsymbol{x}) \approx f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T \delta\boldsymbol{x} + \frac{1}{2}\delta\boldsymbol{x}^T \nabla^2 f(\boldsymbol{x}) \delta\boldsymbol{x}. \tag{5}$$

All of the other variations in the function affect the value only to the order $O(\|\delta\boldsymbol{x}\|^3)$, so for small $\delta\boldsymbol{x}$ they don't have any significant affect on the overall shape of the function. The linear term tells us that, similar to the one-dimensional case, the function is locally flat *only* when $\nabla f(\boldsymbol{x}) = \mathbf{0}$. That behavior dominates the second-order term; this condition is really just a restatement of the one-dimensional case, just applied to each dimension separately since the gradient is just the vector of partial derivatives.

Remember that the quadratic term can be described as a sum of one-dimensional quadratics along the orthogonal directions given by the Hessian's Eigenvectors. Specifically, the Hessian is symmetric, so it has a full set of real Eigenvalues $\{\lambda_i\}_{i=1}^n$ (some of which may be zero or negative), and corresponding full set of mutually orthogonal Eigenvectors $\{\boldsymbol{e}_i\}_{i=1}^n$. The full Eigenspectrum allows us to decompose the Hessian matrix as $\nabla^2 f = \sum_{i=1}^n \lambda_i \boldsymbol{e}_i \boldsymbol{e}_i^T$, which means the quadratic term decomposes as

$$\begin{aligned}
\frac{1}{2}\delta\boldsymbol{x}^T \nabla^2 f(\boldsymbol{x}) \delta\boldsymbol{x} &= \frac{1}{2}\delta\boldsymbol{x}^T \left(\sum_{i=1}^n \lambda_i \boldsymbol{e}_i \boldsymbol{e}_i^T \right) \delta\boldsymbol{x} \\
&= \sum_{i=1}^n \frac{1}{2} \left(\lambda_i (\boldsymbol{e}_i^T \delta\boldsymbol{x})^2 \right) \\
&= \sum_{i=1}^n \frac{1}{2} \lambda_i u_i^2,
\end{aligned}$$

where $u_i = \boldsymbol{e}_i^T \delta\boldsymbol{x}$ is the component of $\delta\boldsymbol{x}$ in the Eigen-direction $\boldsymbol{e}_i$. Importantly, the shape of each of these individual quadratics is determined by the Eigenvalues

λ_i. A positive Eigenvalue means the function is curving up in that direction which is what we want, but a negative Eigenvalue means the function is curving down in that direction, which is definitely not what we want. And again, a zero Eigenvalue could mean we're in a flat region, or it could mean the behavior is defined primarily by higher-order information (curving up in the positive direction and down in the negative direction as defined by the third derivative, or entirely up or entirely down depending on the sign of the forth derivative).

Notice that we can have mixed cases: some Eigenvalues could be strictly positive and others could be zero or strictly negative. A point where the gradient is zero, but the Hessian's Eigenvalues are of mixed sign is called a saddle point.[2]

So to definitively say a point where the gradient is zero is actually a local minimum of the objective, we need to verify that all of the Hessian's Eigenvalues are strictly positive. We have a name for symmetric matrices with a full spectrum of strictly positive Eigenvalues. We call them **positive definite** matrices. And with that, we can fully state first- and second-order optimality criteria for declaring a point $\boldsymbol{x}^*$ is a local optimum of an objective $f : \mathbb{R}^n \to \mathbb{R}$.

The point $\boldsymbol{x}^*$ is a local optimum of $f : \mathbb{R}^n \to \mathbb{R}$ if both of the following optimality criteria hold:

1. **First-order optimality criterion:** The objective's gradient is zero $\nabla f(\boldsymbol{x}) = \mathbf{0}$. This is a necessary condition: it's necessary that a point which is a local minimum satisfy this condition, but it's not sufficient to say that a point *only* satisfying this condition is a local minimum. For that, we need the second-order condition.

2. **Second-order optimality criterion:** The objective's Hessian is positive definite $\nabla^2 f(\boldsymbol{x}^*) = \sum_{i=1}^{n} \lambda_i \boldsymbol{e}_i \boldsymbol{e}_i$ with $\lambda_i > 0$ for all i. Positive definiteness is often denoted $\nabla^2 f(\boldsymbol{x}^*) \succeq 0$. The combination of the first- and second- order criteria are sufficient conditions. If both hold, then we *can* definitively say that the point is a local minimum.

For simple problems, such as unconstrained quadratic optimization, we can use these optimality criteria to directly solve for the minimizer. The gradient gives a linear equation that we can solve for the critical point, and the Hessian tells us what type of critical point we have. This document focuses primarily on what we can do with the analytical criteria, but more generally, they form the basis for any number of iterative algorithms that strive to converge toward a point that satisfies the conditions (Nocedal & Wright, 2006).

[2] It's a "saddle" point because the best real-world example mathematicians had when they started looking at these things was a horse's saddle—in one dimension it curves up and in another, orthogonal, direction, it curves down. Perhaps a better modern name for it might be a "Pringle" point 'cause we've all eaten those. Although, maybe that's too fleeting. Either way, it's a good real-world example of a saddle point, and probably more familiar today than grandpa's old dusty saddle decomposing in the attic. But alas the name's set for life and we'll be mathematically referring to saddles long after we've forgotten that everyone used to ride horses in the good old days of bandits and scarlet fever scares.

4 The geometry of differentiable constraints

This section examines the geometry of constraints and, in particular, what characterizes a locally optimal point when constraints are involved. We can gain insight into the geometry of this problem by considering first how the optimality criteria manifest under explicitly parameterized constraint surfaces. After establishing that intuition, we'll generalize our understanding to implicit surface constraints and inequality constraints.

Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is an objective function and $\phi : \mathbb{R}^d \to \mathbb{R}^n$, denoted $\phi(\boldsymbol{q}) = \boldsymbol{x} \in \mathbb{R}^n$, defines an explicitly parameterized surface embedded in $\mathbb{R}^n$. When the Jacobian of this map is invertible, this surface is of dimension d. Generally, as we saw in Ratliff (2014d), since for any trajectory $\boldsymbol{q} : \mathbb{R} \to \mathbb{R}^d$ through the parameter space we have $\dot{\boldsymbol{x}} = \boldsymbol{J}_\phi \dot{\boldsymbol{q}}$ (which says that any vector tangent to the surface can be constructed as a linear combination of the Jacobian's columns), the columns of $\boldsymbol{J}_\phi$ span the tangent space to the manifold at $\boldsymbol{x}$. Thus, if we compose the parameterized surface map with the objective function to form an equivalent unconstrained objective of the form $\widetilde{f}(\boldsymbol{q}) = f(\phi(\boldsymbol{q}))$ on the space of parameters, we can observe the affect of the constraint by simply analyzing the first-order optimality criteria of the unconstrained problem.

The first-order optimality criterion of this unconstrained variant of the problem is

$$\nabla \widetilde{f}(\boldsymbol{q}) = \boldsymbol{J}_\phi^T \nabla_{\boldsymbol{x}} f(\phi(\boldsymbol{q})) = \boldsymbol{0}. \tag{6}$$

Note that, notationally, $\nabla_{\boldsymbol{x}} f(\phi(\boldsymbol{q}))$ is the gradient of the ambient function f evaluated at $\boldsymbol{x} = \phi(\boldsymbol{q})$. There are two ways to satisfy this condition. The first is the traditional $\nabla_{\boldsymbol{x}} f(\phi(\boldsymbol{q})) = \boldsymbol{0}$. That means the constraint really does nothing—the critical point would have been there anyway. A more common case, though, is where $\nabla_{\boldsymbol{x}} f(\phi(\boldsymbol{q}))$ isn't $\boldsymbol{0}$, but instead is orthogonal to every column of $\boldsymbol{J}_\phi$. Denoting the jth column of $\boldsymbol{J}_\phi$ as $\boldsymbol{c}_j$ we have

$$\begin{bmatrix} \boldsymbol{c}_1^T \nabla_{\boldsymbol{x}} f \\ \boldsymbol{c}_2^T \nabla_{\boldsymbol{x}} f \\ \vdots \\ \boldsymbol{c}_d^T \nabla_{\boldsymbol{x}} f \end{bmatrix} = \boldsymbol{J}_\phi^T \nabla_{\boldsymbol{x}} f = \boldsymbol{0}.$$

Remembering that the columns of $\boldsymbol{J}_\phi$ span the tangent space, we can conclude that this condition suggests the gradient in the ambient space $\nabla_{\boldsymbol{x}} f$ needs to be entirely orthogonal to the surface's tangent space in order for it to be a local minimum.

Moreover, since this is the first-order optimality condition of the unconstrained problem in parameter space, we can conclude that if we find such a point, i.e. a point for which the ambient gradient is either zero or orthogonal to the tangent space, we're at a critical point and, specifically a local minimum if the corresponding second-order conditions are satisfied. That said, we postpone a discussion of constrained second-order optimality conditions to Section 6.

The specific way in which we parameterize the constraint surface is largely arbitrary, but as long as the shape of the resulting surface is the same, the geometry of the tangent space will always be the same and this first-order optimality condition will always trigger at the same point. The condition is a function of the tangent space and ambient gradient only, so it doesn't depend on the parameterization. Indeed, it doesn't even depend on the specific surface representation. As long as we have a differentiable surface embedded in the ambient space, we can always hypothesize such a parameterization and the condition is the same.

That suggests we can simply generalize the condition to be a property of the surface itself, rather than any particular parameterization: **first-order optimality manifests on a constraint surface as the condition that the ambient gradient is either zero, or orthogonal to the surface's tangent space.** Intuitively, we can see to the first order that if the ambient gradient weren't orthogonal to the surface's tangent space, there'd be some component parallel with it, and moving some tiny distance ϵ in that direction will change the value of the function on the surface.

Thus, we can now also apply the property to implicit surface representations. If the surface is represented as the zero set of a map $\boldsymbol{h} : \mathbb{R}^n \to \mathbb{R}^k$, we saw in Ratliff (2014d) that the tangent space is given by the Jacobian's null space $\text{Null}(\boldsymbol{J_h})$. So we can restate the condition that the gradient must be orthogonal to the tangent space by saying that it must be orthogonal to the Jacobian's null space, which in turn means that it must lie in the Jacobian's row space. Therefore, for implicitly defined differentiable surface constraints, we can characterize first-order optimality as $-\nabla f \in \textbf{span}(\boldsymbol{J_h})$. (We use the negative gradient since it's more natural when we're moving downhill.) Another way of writing that condition is

$$-\nabla f = \boldsymbol{J_h}^T \boldsymbol{\lambda} \quad \text{or} \quad \nabla f + \boldsymbol{J_h}^T \boldsymbol{\lambda} = \boldsymbol{0}, \tag{7}$$

for some vector of coefficients $\boldsymbol{\lambda} \in \mathbb{R}^k$. Note that this condition can be satisfied even if the actual equality constraint $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}$ isn't, so in practice we need to make sure both the constraint itself *and* Equation 7 are satisfied before we declare the first-order optimality conditions to hold.

If we think of $-\nabla f$ as a potential force induced by the objective at a given point pushing the point downhill, then since the rows of $\boldsymbol{J_h}$ consist of the gradients $\nabla h_i(\boldsymbol{x})^T$ of the individual output component functions, we can interpret this condition as

$$-\nabla f + \sum_{i=1}^{k} \lambda_i \Big(\ -\nabla h_i(\boldsymbol{x}) \ \Big) = \boldsymbol{0}. \tag{8}$$

In other words, the first-order optimality condition is a force-balance condition. When the constraint is satisfied and the force of the objective potential is fully balanced by corresponding forces of the constraint functions, then the constrained first-order optimality conditions are satisfied (for simple surface (equality) constraints).

We can summarize the resulting equality constrained first-order necessary conditions for optimality by adding the constraints to the objective introducing a new set of k variables $\boldsymbol{\lambda} \in \mathbb{R}^k$, one for each constraint, to form what we call a **Lagrangian**

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{h}(\boldsymbol{x}). \tag{9}$$

Then these two equations can be expressed

$$\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) + \boldsymbol{J}_{\boldsymbol{h}}^T \boldsymbol{\lambda} = \boldsymbol{0} \tag{10}$$

$$\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}. \tag{11}$$

An analysis of the Lagrangian (which we won't explore in depth here) shows that this critical point is actually a saddle point of the Lagrangian. This procedure of placing the equality constraints up into the objective and searching for a saddle point is called **The Method of Lagrange Multipliers**. The next few sections give some examples of its application.

4.1 Equality constrained quadratic programs

Now we know how to analytically solve equality constrained quadratic programs. Optimization problems of this type take the form

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \frac{1}{2} \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} - \boldsymbol{b}^T \boldsymbol{x} + c \\ \text{s.t.} \quad & \boldsymbol{C} \boldsymbol{x} = \boldsymbol{d}, \end{aligned} \tag{12}$$

where $\boldsymbol{A} \in \mathbb{R}^{n^2}$ is a positive definite matrix, $\boldsymbol{b} \in \mathbb{R}^n$ and $c \in \mathbb{R}$ are the linear and constant parameters of the quadratic, respectively, and $\boldsymbol{C} \in \mathbb{R}^{k \times n}$ and $\boldsymbol{d} \in \mathbb{R}^k$ define the k linear equality constraints.

The gradient of the objective $\nabla f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$ must be orthogonal to the tangent space of the linear constraint which is the (right) null space of the constraint function's Jacobian, and that means the ambient gradient needs to lie somewhere in span of the rows of $\frac{\partial}{\partial \boldsymbol{x}} (\boldsymbol{C}\boldsymbol{x} - \boldsymbol{d}) = \boldsymbol{C}$.

Thus, we have two equations

$$\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} + \boldsymbol{C}^T \boldsymbol{\lambda} = \boldsymbol{0} \tag{13}$$

$$\boldsymbol{C}\boldsymbol{x} = \boldsymbol{d} \tag{14}$$

that in combination describe the solution. We can solve the top equation for $\boldsymbol{x}$ as a function of $\boldsymbol{\lambda}$ and then plug that into the second to fully solve for $\boldsymbol{\lambda}$. Plugging the resulting expression for $\boldsymbol{\lambda}$ back into the the expression we got for $\boldsymbol{x}$ in terms of $\boldsymbol{\lambda}$ gives

$$\boldsymbol{x} = \boldsymbol{A}^{-1} \left(\boldsymbol{b} - \boldsymbol{C}^T \left(\boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{C}^T \right)^{-1} \left[\boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{b} - \boldsymbol{d} \right] \right).$$

This procedure provides an interesting analytical solution to the problem, but in practice one should be careful to consider the computational characteristics of such a solution when choosing whether to use it. If $\boldsymbol{A}$ is a diagonal weight matrix, we can invert it fast and this expression can be very efficient, especially when there are only a handful of constraints. For instance, if $k = 3$, the matrix that we would need to invert $\boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{C}^T$ is only 3×3, and again its inversion is efficient.

However, if $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is a more general positive definite matrix, and thereby computationally expensive to invert, we might want to find some other way of solving the system. Another approach to solving the equality constrained quadratic program in Equation 12 that may be more efficient in this case would be to directly compute a parameterization for the linear constraint surface and solve the problem with respect to that parameter. Note that the SVD of $\boldsymbol{C}$ is

$$\boldsymbol{C} = \boldsymbol{U} \begin{bmatrix} \boldsymbol{S} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_{/\!/}^T \\ \boldsymbol{V}_{\perp}^T \end{bmatrix}, \tag{15}$$

and we can explicitly write out the generic solution to the linear constraint system using what we know from linear algebra

$$\boldsymbol{x}(\boldsymbol{\beta}) = \underbrace{\boldsymbol{V}_{/\!/}\boldsymbol{S}^{-1}\boldsymbol{U}^T\boldsymbol{d}}_{\widetilde{\boldsymbol{x}}} + \boldsymbol{V}_{\perp}\boldsymbol{\beta}, \tag{16}$$

where, as indicated in the expression, it's convenient to denote the particular least-squares solution of the constraints (the solution of smallest norm) as $\widetilde{\boldsymbol{x}} = \boldsymbol{V}_{/\!/}\boldsymbol{S}^{-1}\boldsymbol{U}^T\boldsymbol{d}$.

This expression gives the desired parameterization in terms of the vector $\boldsymbol{\beta}$; plugging it back into the objective function gives a new unconstrained objective in terms of $\boldsymbol{\beta}$ which will solve the constrained problem for us:

$$\begin{aligned} \widetilde{f}(\boldsymbol{\beta}) &= f(\widetilde{\boldsymbol{x}} + \boldsymbol{V}_{\perp}\boldsymbol{\beta}) \\ &= \frac{1}{2}(\widetilde{\boldsymbol{x}} + \boldsymbol{V}_{\perp}\boldsymbol{\beta})^T\boldsymbol{A}(\widetilde{\boldsymbol{x}} + \boldsymbol{V}_{\perp}\boldsymbol{\beta}) - \boldsymbol{b}^T(\widetilde{\boldsymbol{x}} + \boldsymbol{V}_{\perp}\boldsymbol{\beta}) + c. \end{aligned}$$

The solution in terms of both $\boldsymbol{\beta}$ and the original variable $\boldsymbol{x} = \widetilde{\boldsymbol{x}} + \boldsymbol{V}_{\perp}\boldsymbol{\beta}$ is then

$$\boldsymbol{\beta}^* = \left(\boldsymbol{V}_{\perp}^T\boldsymbol{A}\boldsymbol{V}_{\perp}\right)^{-1}\boldsymbol{V}_{\perp}^T\left[\,\boldsymbol{b} - \boldsymbol{A}\widetilde{\boldsymbol{x}}\,\right] \tag{17}$$

$$\text{and} \quad \boldsymbol{x}^* = \widetilde{\boldsymbol{x}} + \boldsymbol{V}^T\left(\boldsymbol{V}_{\perp}^T\boldsymbol{A}\boldsymbol{V}_{\perp}\right)^{-1}\boldsymbol{V}_{\perp}^T\left[\,\boldsymbol{b} - \boldsymbol{A}\widetilde{\boldsymbol{x}}\,\right]. \tag{18}$$

We know from the above analysis of surface constraints that the ambient gradient should be orthogonal to the surface at the solution. So lets check ourselves. In this case, we want to verify that the resulting ambient gradient is orthogonal to the tangent space at the solution, which is given by the columns

of $\boldsymbol{V}_\perp$:

$$\begin{aligned}\boldsymbol{V}_\perp^T \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^*) &= \boldsymbol{V}_\perp^T \Big(\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b} \Big) \\ &= \boldsymbol{V}_\perp^T \boldsymbol{A} \Big(\widetilde{\boldsymbol{x}} + \boldsymbol{V}^T \left(\boldsymbol{V}_\perp^T \boldsymbol{A} \boldsymbol{V}_\perp\right)^{-1} \boldsymbol{V}_\perp^T \Big[\boldsymbol{b} - \boldsymbol{A}\widetilde{\boldsymbol{x}} \Big] \Big) - \boldsymbol{V}_\perp^T \boldsymbol{b} \\ &= \boldsymbol{V}_\perp^T \boldsymbol{A}\widetilde{\boldsymbol{x}} + \boldsymbol{V}_\perp^T \Big[\boldsymbol{b} - \boldsymbol{A}\widetilde{\boldsymbol{x}} \Big] - \boldsymbol{V}_\perp^T \boldsymbol{b} \\ &= \boldsymbol{0}.\end{aligned}$$

Good. We were right. In practice, validating your calculations this way using expected theoretical properties can be an extremely good way to strengthen your confidence that your solution is correct.

4.2 Manipulator inverse kinematics

These tools also enable us to derive more general motion algorithms for nonlinear robotic manipulators. Suppose our manipulator has d joints and a forward kinematics map of the form $\phi : \mathbb{R}^d \to \mathbb{R}^3$ mapping the joint angles to the 3D point location of the end-effector. Classically, inverse kinematics studies how to make that end-effector move in a particular desired direction $\dot{\boldsymbol{x}}_d$. Suppose we have enough joints such that we can solve the equation $\boldsymbol{J}_\phi \dot{\boldsymbol{q}} = \dot{\boldsymbol{x}}$ for at least one solution. We know $\dot{\boldsymbol{q}} = \boldsymbol{J}_\phi^\dagger \dot{\boldsymbol{x}}$ always gives a particular solution, but typically since $d > 3$, the Jacobian $\boldsymbol{J}_\phi$ has a significant null space that the least-squares solution resolves somewhat arbitrarily.

In practice, there's often some default configuration $\boldsymbol{q}_0$ that we'd like to move toward in the null space so that the pose of the robot over time remains somewhat natural. We can, therefore, say that we want to move in a default direction $\dot{\boldsymbol{q}}_d = \boldsymbol{q} - \boldsymbol{q}_0$ within the null space while ensuring that the end-effector always definitively moves in the direction $\dot{\boldsymbol{x}}$. These requirements suggest an equality constrained quadratic program of the form

$$\begin{aligned}\min_{\dot{\boldsymbol{q}}} \quad & \frac{1}{2}\|\dot{\boldsymbol{q}}_d - \dot{\boldsymbol{q}}\|^2 \\ \text{s.t.} \quad & \boldsymbol{J}_\phi \dot{\boldsymbol{q}} = \dot{\boldsymbol{x}}_d.\end{aligned}$$

We have a constraint that dictates the end-effector *must* move in the right direction, but given that that's doing the right thing, we do our best to move in the direction $\dot{\boldsymbol{q}}_d$ with the remaining degrees of freedom. Solving this problem using the above techniques (assuming $\boldsymbol{J}_\phi$ is full rank) gives a solution of the form

$$\dot{\boldsymbol{q}}^* = \boldsymbol{J}_\phi^\dagger \dot{\boldsymbol{x}}_d + \Big(\boldsymbol{I} - \boldsymbol{J}_\phi^\dagger \boldsymbol{J}_\phi \Big) \dot{\boldsymbol{q}}_d,$$

where $\boldsymbol{J}_\phi^\dagger = \boldsymbol{J}_\phi^T \left(\boldsymbol{J}_\phi \boldsymbol{J}_\phi^T \right)^{-1}$ is the pseudoinverse. As we saw in Ratliff (2014d), the first term is just the pseudoinverse solution, and the second term expresses a projection of $\dot{\boldsymbol{q}}_d$ onto the null space of $\boldsymbol{J}_\phi$.

4.3 Gauss's principle of least constraint

Another example of the application of the Method of Lagrange Multipliers, this time from classical mechanics, is Gauss's Principle of Least Constraint. This principle is another example of an *optimality principle* in physics as introduced in Section 1.2.

To start the discussion we just state that the *unconstrained* dynamics of a rigid body dynamical system, such as a robot, is

$$\boldsymbol{M}\ddot{\boldsymbol{q}}_u + \boldsymbol{h} = \boldsymbol{\tau}, \tag{19}$$

where the subscript u on $\ddot{\boldsymbol{q}}_u$ denotes that they're unconstrained accelerations. The matrix $\boldsymbol{M} \in \mathbb{R}^{d\times d}$ is a positive definite matrix describing how the inertias manifest in joint coordinates, and $\boldsymbol{h}$ is a vector describing how velocities create fictional forces such as centripetal and Coriolis forces that contribute to the system. Typically, $\boldsymbol{M}$ is a function of the configuration $\boldsymbol{q}$ and $\boldsymbol{h}$ is a function of both $\boldsymbol{q}$ and velocities $\dot{\boldsymbol{q}}$, but when analyzing only accelerations at a fixed moment of time, we can consider $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ to be constant, and we thereby get a linear relationship between the applied system forces and torques $\boldsymbol{\tau}$ and the resulting unconstrained accelerations of the system $\ddot{\boldsymbol{q}}_u$.

Given those unconstrained dynamics, Gauss's principle tells us that in the presence of constraints, when we measure the size of a difference between acceleration vectors using the inertia matrix $\boldsymbol{M}$ as a metric, the actual constrained accelerations realized by physics are those that are as close as possible to the corresponding unconstrained accelerations that also satisfy the constraints. Mathematically, denoting the constraints in terms of accelerations as $\boldsymbol{C}\ddot{\boldsymbol{q}} = \boldsymbol{d}$, we can write

$$\begin{aligned} \min_{\ddot{\boldsymbol{q}}} \ & \frac{1}{2}\|\ddot{\boldsymbol{q}} - \ddot{\boldsymbol{q}}_u\|_{\boldsymbol{M}}^2 \\ \text{s.t.} \quad & \boldsymbol{C}\ddot{\boldsymbol{q}} = \boldsymbol{d}, \end{aligned}$$

where $\ddot{\boldsymbol{q}}_u = \boldsymbol{M}^{-1}(\boldsymbol{\tau} - \boldsymbol{h})$.

For instance, if the equation $\boldsymbol{M}\ddot{\boldsymbol{q}}_u + \boldsymbol{h} = \boldsymbol{\tau}$ describes the unconstrained dynamics of a manipulator and now we introduce the constraint that the end-effector is touching a wall and (because of friction and the physicality of the barrier) can't move from that point, we get a non-linear constraint $\phi(\boldsymbol{q}) = \boldsymbol{c}$, where $\boldsymbol{c} \in \mathbb{R}^3$ is a constant. Since we only care about accelerations, we can differentiate that equation twice to get

$$\underbrace{\boldsymbol{J}_\phi}_{\boldsymbol{C}}\ddot{\boldsymbol{q}} = \underbrace{-\dot{\boldsymbol{J}}_\phi\dot{\boldsymbol{q}}}_{\boldsymbol{d}},$$

which places the constraint into the above form.

Using this particular version of the constraints, we can use the first-order optimality principle given by the Method of Lagrange Multipliers to get an expression for how the end-effector constraint affects the dynamics. What results

is a system of equations of the form

$$\begin{aligned} \boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{h} &= \boldsymbol{\tau} + \boldsymbol{J}_\phi^T \boldsymbol{\lambda} \\ \boldsymbol{J}_\phi \ddot{\boldsymbol{q}} &= -\dot{\boldsymbol{J}}_\phi \dot{\boldsymbol{q}}. \end{aligned} \tag{20}$$

The second equation is just the kinematic constraint on how the system can accelerate given that it's end-effector shouldn't move from the wall. What's interesting is the first equation. That equation is exactly the unconstrained dynamics, modified by an added term $\boldsymbol{J}_\phi^T \boldsymbol{\lambda}$ which expresses how the Lagrange multipliers $\boldsymbol{\lambda}$ play into the equation.

In Section 4, we discussed how the individual equality constraint functions could be viewed as potential functions whose negative gradients model "forces" opposing the corresponding "force" of the objective. In this case, we can remove the quotes since the negative gradients *are* actually in units of force, so we can explicitly interpret them as real physical forces. The constraints generate constraint forces $\boldsymbol{J}_\phi^T \boldsymbol{\lambda}$ that intermix with the combined unconstrained forces $\boldsymbol{\tau} - \boldsymbol{h}$ to create a net force whose action doesn't violate the constraints. Equations 20, in combination, then define how that net force translates into the resulting constrained accelerations. In particular, $\boldsymbol{J}_\phi^T$ may be interpreted as a transformation from workspace coordinates to joint space coordinates, which means we can interpret the Lagrange multipliers $\boldsymbol{\lambda}$ physically as forces applied to the system in the workspace. $\boldsymbol{\lambda}$ is the force exerted on the end-effector by the wall that prevents the end-effector from penetrating the wall or sliding along its surface. The Method of Lagrange Multipliers, in combination with this fundamental physical principle of Gauss, gives us an explicit set of equations that define how forces in the workspace interact with a dynamical system to define its net acceleration.

5 Karush-Kuhn-Tucker conditions and their application

So far, we've restricted our discussion to just equality constraints, but the most general form of the problem given in Equation 4 additionally includes inequality constraints.

Fortunately, dealing with those is relatively straightforward given the machinery we've established above. The trick is to realized that a given inequality constraint $g_j(\boldsymbol{x}) \leq 0$ either holds with equality, i.e. the local minimum is butting up against the constraint surface, or it can be ignored because the local minimum already naturally satisfies the constraint.

Consider the simple case of a quadratic potential with a linear inequality constraint of the form $\boldsymbol{v}^T \boldsymbol{x} \leq c$, where $\boldsymbol{v} \in \mathbb{R}^n$ is just an n-dimensional vector. The constraint forms an $(n-1)$-dimensional constraint surface described by the null space of the vector $\boldsymbol{v}$ (the space of all points orthogonal to that vector), offset from the origin by the appropriate amount $c\frac{\boldsymbol{v}}{\|\boldsymbol{v}\|^2}$. The vector $\boldsymbol{v}$ points in the direction of increasing value, so the set of all valid points are on the *opposite* side of the surface, which we'll call the negative side. If the quadratic objective

Karush-Kuhn-Tucker (KKT) conditions. For any problem of the form given by Equation 4, the following are necessary conditions for a constrained local minimum $\boldsymbol{x}^*$.

1. The **objective gradient is orthogonal to the implicit surface** defined by the collection of all equality and active inequality constraints:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}^*) + \boldsymbol{J}_{\boldsymbol{h}}^T \boldsymbol{\lambda} + \boldsymbol{J}_{\boldsymbol{g}}^T \boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{J}_{\boldsymbol{h}}^T & \boldsymbol{J}_{\boldsymbol{g}}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\mu} \end{bmatrix} = \mathbf{0}, \tag{21}$$

   where $\boldsymbol{\lambda} \in \mathbb{R}^k$ are Lagrange multipliers for the equality constraints and $\boldsymbol{\mu} \in \mathbb{R}^l$ are Lagrange multipliers for the inequality constraints. Note that Condition 4 below states that Lagrange multipliers for inactive inequality Lagrange must be zero, so their contributions to this equation vanish.

2. The solution must be **feasible** in the sense that all equality and inequality constraints must be satisfied: $\boldsymbol{h}(\boldsymbol{x}^*) = \mathbf{0}$ and $\boldsymbol{g}(\boldsymbol{x}^*) \leq \mathbf{0}$.

3. **Inequality Lagrange multipliers are non-negative** since they only work in one direction. Specifically, $\mu_j \geq 0$ for all j. In other words, the inequality constraint surface don't prevent the objective from pushing further into the feasible region. They only prevent the objective from pushing out of the region.

4. **Complimentary slackness.** We typically write the condition of inequality activation as $\mu_j g_j(\boldsymbol{x}^*) = 0$ for all j, meaning that μ_j strictly positive implies that $\boldsymbol{x}^*$ lies on the constraint surface with $g_j(\boldsymbol{x}^*) = 0$. And $g_j(\boldsymbol{x}^*) < 0$ ($\boldsymbol{x}^*$ is strictly in the interior of the feasible region) implies $\mu_j = 0$.

Figure 4: KKT conditions.

function has its unique minimum value on this negative side, then the constraint really does nothing, it's vacuous. We can add a million more inequality constraints that are already satisfied at the objective's global minimum and they don't affect the problem at all.

On the other hand, if the minimum falls on the positive side of the constraint, then that unconstrained global minimum is invalid. We need to address that discrepancy to correctly solve the constrained problem. Fortunately, integrating the constraint is quite easy given what we know about equality constraints. If we know an inequality constraint makes a difference, we can just treat it like an equality constraint. There's never a benefit (as measured by the objective function) to not be on the surface of the constraint if the objective function is trying to push the minimum point past that constraint boundary. So if we know a constraint is *active* in this sense, we can simply include it in the problem as an equality constraint.

The more general case, where f isn't just quadratic and the constraint functions g_j aren't linear, is similar. For any local minimum, either a constraint matters or it doesn't. If it doesn't matter (i.e. the objective isn't trying to push through the constraint surface), then we can just ignore the constraint. If it does matter (the objective *is* actively trying to push the point through the constraint surface), we can treat it as an equality constraint. This document just treats these problems analytically, so we're primarily interested in this theoretical observation. But in practice, we often devise effective algorithms by estimating which inequality constraints are *active* and which aren't. Given such an estimate, we can solve the resulting equality constrained problem, check if our activation guesses were right, and then iterate if necessary. Such an algorithm is called an *active set* method.

These observations about the activation of inequality constraints lead to a general set of first-order optimality criteria (necessary conditions) for nonlinear equality and inequality constrained problems. These conditions are collectively called the Karush-Kuhn-Tucker (KKT) conditions Nocedal & Wright (2006). Figure 4 lists the KKT conditions in full.

These KKT conditions, together, give us theoretical conditions that can both help us find a local minimum analytically for relatively easy problems, and derive iterative algorithms for addressing more complicated constrained problems.

5.1 Steepest descent revisited

[This section is will be completed once the homework is done.]

5.2 Control as a quadratic program

We saw above in Equation 20 that we can describe the constrained dynamics of a robot in "equality" contact with the environment as a pair of constraints relating the system's acceleration $\ddot{\boldsymbol{q}}$ to the forces/torques $\boldsymbol{\tau}$ and workspace forces $\boldsymbol{\lambda}$ at the contacts. More generally, we might have inequality kinematic constraints stating, for instance, that the end-effector can accelerate in the workspace away from

the contact surface, but not in the opposite direction through the constraint. For this example, if $\boldsymbol{v}$ is a vector normal to the surface, we can require that $\boldsymbol{v}^T\dot{\boldsymbol{x}} \geq 0$. Differentiating the right hand side once gives $\boldsymbol{v}^T\boldsymbol{J}_\phi\ddot{\boldsymbol{q}} + \boldsymbol{v}^T\dot{\boldsymbol{J}}_\phi\dot{\boldsymbol{q}} \geq 0$. This is an inequality constraint, but that's okay. We just saw above that the inequality constraints are either vacuous, or they hold with equality (which, in this case, would reduce to the situation we analyzed in Section 4.3).

In general, given any collection of equality or inequality constraints representing kinematic feasibility relations—physical constraints induced by, for instance, friction, or anything else that would make our model more plausible—we can now try to control the robot by optimizing any sort of quadratic function of the form $Q(\ddot{\boldsymbol{q}}, \boldsymbol{\tau}, \boldsymbol{\lambda})$ subject to the first contact dynamics constraint of Equation 20 (relating the variables $\ddot{\boldsymbol{q}}$, $\boldsymbol{\tau}$, and $\boldsymbol{\lambda}$) and all of these extra kinematic constraints. This quadratic objective may, for instance, try to get $\ddot{\boldsymbol{q}}$ to accelerate in some desired direction $\ddot{\boldsymbol{q}}_d$ while keeping $\boldsymbol{\tau}$ small, or it could penalize components of $\boldsymbol{\lambda}$ tangential to the contact surface since forces in those directions might induce slippage. Generally, anything that models the problem well that can be written as a quadratic function of these variables can be packed into the objective.

The resulting problem is

$$\begin{aligned} \min_{\ddot{\boldsymbol{q}},\boldsymbol{\tau},\boldsymbol{\lambda}} \quad & Q(\ddot{\boldsymbol{q}}, \boldsymbol{\tau}, \boldsymbol{\lambda}) \\ \text{s.t.} \quad & \boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{h} = \boldsymbol{\tau} + \boldsymbol{J}_\phi^T\boldsymbol{\lambda} \\ & \boldsymbol{h}_{\text{lin}}(\ddot{\boldsymbol{q}}, \boldsymbol{\tau}, \boldsymbol{\lambda}) = \boldsymbol{0} \\ & \boldsymbol{g}_{\text{lin}}(\ddot{\boldsymbol{q}}, \boldsymbol{\tau}, \boldsymbol{\lambda}) \leq \boldsymbol{0}, \end{aligned}$$

where $\boldsymbol{h}_{\text{lin}}$ and $\boldsymbol{g}_{\text{lin}}$ are both linear maps. This method is the state-of-the-art. Optimizers for solving problems of this form, built around the KKT optimality principles outlined above, are very fast, and can be solve this sort of problem in real time up to 1000 times per second if appropriate measures are applied to exploit structure in the problem. The most sophisticated robots in the world today, especially humanoid robots that need to balance on their own two feet by controlling the forces exerted on each joint, exploit quadratic programming at the core of their control system, solving problems of this sort in real time. Two good resources for further reading in this area are Kuindersma et al. (2014) and Herzog et al. (2014). See Ratliff (2014b) for a more pedagogical introduction to the ideas.

6 Constrained second-order optimality conditions

The second-order optimality conditions for constrained problems are more difficult to describe than those of unconstrained problems. And truthfully, they aren't discussed as much in the applications literature in many areas relevant to intelligent systems. But they're important to the the derivation and analysis of algorithms so it's good to have at least an intuitive understanding of how

they work. Just as we did for equality constrained problems, we can form a Lagrangian for the full problem given in Equation 4, too, as

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{\mu}^T \boldsymbol{g}(\boldsymbol{x}).$$

This representation is a convenient compact way to remember the KKT conditions: the first two conditions, for instance, are given by setting the gradients of this Lagrangian w.r.t. $\boldsymbol{x}$, $\boldsymbol{\lambda}$, and $\boldsymbol{\mu}$ to $\mathbf{0}$. But of theoretical importance, now we have a representation that we can study for second-order optimality.

At a stationary point (i.e. one that satisfies the first-order optimality criteria given by the KKT conditions), we have specific fixed values for $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$. We can view the resulting Lagrangian with respect to $\boldsymbol{x}$ as a proxy to constrained problem. The equation $\nabla_{\boldsymbol{x}} \mathcal{L} = \mathbf{0}$ tells us that the "potential forces" produces by negative gradients of f, $\boldsymbol{h}$, and $\boldsymbol{g}$ (for active inequality constraints), must balance. The Lagrange multipliers $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ tell us how each constraint should weigh into creating this unconstrained proxy, but, ultimately, we do have an unconstrained proxy function that we can now analyze with respect to second-order optimality.

Second-order optimality in this case is effectively that the Hessian of the Lagrangian is positive definite $\nabla^2_{\boldsymbol{xx}} \mathcal{L} \succeq 0$. However, now we're typically resting on a constraint surface consisting of both equality and inequality constraints, so the set of plausible perturbations away from the current point is restricted. Effectively, the positive definiteness requirement is restricted to only needing to hold for the set of directions consistent with the first-order (tangent) representation of the constraints. In other words, we require that $\boldsymbol{w}^T \nabla^2_{\boldsymbol{xx}} \mathcal{L} \boldsymbol{w} > 0$ for all nonzero $\boldsymbol{w}$ for which $\boldsymbol{w}^T \nabla h_i(\boldsymbol{x}) = 0$ and $\boldsymbol{w}^T \nabla g_j(\boldsymbol{x}) \leq 0$ for all i and j. See Nocedal & Wright (2006) for a more in depth discussion of the second-order conditions and their application.

7 Other forms of optimization

So far, we've assumed that our objective and constraints are all second-order differentiable, and that assumption gave us enough structure to analytically solve some otherwise complicated problems. But that isn't the only structure we could have exploited. Applications exist for practically any type of optimization problem you might think of. Sometimes we only have access to function values, or just function values and gradients. Sometimes we can't differentiate the function everywhere, but we know that the function is bowl-like (convex), and other times our variables can only take on integer values, which really complicates things. Even more, we sometimes can't even say with certainty that the function value is deterministic. We might get a different value every time we evaluate the function. But even in this seemingly hopeless cases, if we can garner some structure by characterizing the *distribution* of values we might get, then there's still hope that we can optimize.

For each variant, there are huge bodies of literature available on the topic describing what additional structure we might have to assume and how to exploit

those structures to derive algorithms and get guarantees on their convergence or convergence rates. This document presents some analytical techniques for studying and solving second-order differentiable nonlinear programming problems, but such problems are only a small fraction of what you might encounter in practice. Nonetheless, these techniques have many applications in the sub-fields of intelligent systems and physics, so learning them thoroughly as a starting point can't go wrong. For more information and theoretical insight into these methods, along with a thorough discussion and analysis of associated algorithms for iterative optimization, see Nocedal & Wright (2006) and Boyd & Vandenberghe (2004).

References

Boyd, Stephen and Vandenberghe, Lieven. *Convex Optimization*. Cambridge University Press, 2004.

Herzog, A., Righetti, L., Grimminger, F., Pastor, P., and Schaal, S. Balancing experiments on a torque-controlled humanoid with hierarchical inverse dynamics. In *Proceedings of the IEEE International Conference on Intelligent Robotics Systems*, 2014. URL `http://www-clmc.usc.edu/publications/H/herzog-IROS2014.pdf`.

Kuindersma, S., Permenter, F., and Tedrake, R. An efficiently solvable quadratic program for stabilizing dynamic locomotion. In *In Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2014.

Nocedal, Jorge and Wright, Stephen. *Numerical Optimization*. Springer, 2006.

Ratliff, Nathan. Analytical dynamics and contact analysis, 2014a. Lecture notes: Advanced Robotics series.

Ratliff, Nathan. Controlling floating-based robots, 2014b. Lecture notes: Advanced Robotics series.

Ratliff, Nathan. Multivariate calculus I: Derivatives and local geometry, 2014c. Lecture notes: Mathematics for Intelligent Systems series.

Ratliff, Nathan. Multivariate calculus II: The geometry of smooth maps, 2014d. Lecture notes: Mathematics for Intelligent Systems series.

Susskind, Leonard. *The Theoretical Minimum: Classical Mechanics*. Stanford: Continuing Studies, 2011. URL `http://theoreticalminimum.com/courses/classical-mechanics/2011/fall`.