Chapter 5

Optimization

The field of optimization is vast and complex, and much of it lies beyond the scope of this course. We will focus on the core techniques for optimization commonly encountered in the context of robotics. If you are interested in more on optimization, please see the bibliography for this section — most of the resources referenced go into much more depth than we will in these notes.

5.1 Calculus for Optimization

Many common optimization techniques are rooted in calculus. As such, we will begin our study of optimization with a review of some concepts from this area. We will assume basic knowledge of Calculus I through III (roughly, derivatives, integrals, and techniques for computing both, as well as the basics of vectors and vector-valued functions). For a more complete introduction to calculus, we recommend the excellent resources available in Dawkins [1, 2, 3].

5.1.1 Partial Derivatives

Most optimization problems encountered in robotics pertain to functions of multiple variables. As such, we often need to be able to compute a derivative of one of these functions with respect to a single variable. We have already seen partial derivatives in earlier sections (such as in the Euler-Lagrange equations), but they are especially critical for solving optimization problems of many kinds. We will now briefly review the definition and computation of a partial derivative:

Definition 5.1 (The Partial Derivative of a Real-Valued Function) For some function $f : \mathbb{R}^n \to \mathbb{R}$, i.e. $f(x_1, \ldots, x_n)$, we define $\frac{\partial f}{\partial x_i}$, the **partial derivative of f with respect to x_i**, as $\frac{\partial f}{\partial x_i} = \frac{d}{dx_i} g(x_i)$, where $g(x_i)$ is defined as f with

$x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ held constant. In other words, $\frac{\partial f}{\partial x_i}$ is the derivative of f with every input variable other than x_i treated as though it were constant.

Given this definition, we can see that computing a partial derivative is nearly equivalent to computing an ordinary derivative. The partial derivative of a function *is* the ordinary derivative of that function with all but one variable treated as constant, so computing a partial derivative is as simple as imagining that every variable other than some x_i is a constant and computing the ordinary derivative of f with respect to x_i.

As an example, consider the following:

Example 5.1 Take $f(x, y, z) = 3\sin(x)\tan^2(y) + \frac{4x^2}{\log(z)}$. We wish to compute $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial z}\left(\frac{\partial f}{\partial x}\right)$.

To begin, we will compute $\frac{\partial f}{\partial x}$ by taking y and z to be constants and computing as follows:

$$\frac{d}{dx}g(x) = \frac{d}{dx}\left(3\sin(x)\tan^2(y) + \frac{4x^2}{\log(z)}\right)$$

$$= \frac{d}{dx}\left(3\sin(x)\tan^2(y)\right) + \frac{d}{dx}\left(\frac{4x^2}{\log(z)}\right) \qquad \text{The derivative of a sum is the sum of the derivatives}$$

$$= 3\tan^2(y)\frac{d}{dx}\sin(x) + \frac{4}{\log(z)}\frac{d}{dx}x^2 \qquad \text{Pull out the constants}$$

$$= 3\tan^2(y)\cos(x) + \frac{8x}{\log(z)}$$

Computing $\frac{\partial f}{\partial z}\left(\frac{\partial f}{\partial x}\right)$ is very similar. We begin with $\frac{\partial f}{\partial x}$ as computed above and proceed as follows:

$$\frac{d}{dz}h(z) = \frac{d}{dz}\left(3\cos(x)\tan^2(y) + \frac{8x}{\log(z)}\right)$$

$$= \frac{d}{dz}\left(3\cos(x)\tan^2(y)\right) + \frac{d}{dz}\left(\frac{8x}{\log(z)}\right) \qquad \text{The derivative of a sum is the sum of the derivatives}$$

$$= 0 + 8x\frac{d}{dz}\frac{1}{\log(z)} \qquad \text{The first term is all constants, with a derivative of zero}$$

$$= \frac{-8x}{z\log^2(z)}$$

As you may be able to see from this example, the order in which we evaluate partial derivatives **does not** change the final result. The intuition for this property is that the other variables in the equation play no role in the computation of a partial derivative with respect to some specific variable, so the partial derivatives each have a sort of independence. To better see this for yourself, try computing $\frac{\partial f}{\partial x}\left(\frac{\partial f}{\partial z}\right)$ and comparing your result to the result of $\frac{\partial f}{\partial z}\left(\frac{\partial f}{\partial x}\right)$ in the above example. For more on the partial derivative, see Dawkins [3].

5.1.2 The Gradient

The gradient is core to a large number of optimization techniques. It can be considered as a more general version of the derivative, expanding the notion of the change of a function with respect to some variable to multiple dimensions. Computing the gradient is quite simple: For a function with input variables $x_1, \ldots, x_n$, the gradient is the vector consisting of the function's partial derivative with respect to each of the x_i in turn. More formally:

Definition 5.2 (Gradient of a Real-Valued Function) Given a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, i.e. $f(x_1, \ldots, x_n)$, we define **the gradient of f to be $\nabla f = \langle \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \rangle$, the vector of all first-order partial derivatives of f.**

The gradient is useful because it tells us the direction of increase of a function at every point. If we imagine placing a ball at a point $(x_1, \ldots, x_n)$ on the surface created by the values of a function f, then $(-\nabla f)(x_1, \ldots, x_n)$ tells us which way that ball would roll (obviously, this intuition makes the most sense when $n = 2$ or 3). The gradient's value is a vector with magnitude equivalent to the slope of f and direction equivalent to the direction in which f is increasing fastest.

The utility of the gradient to optimization is thus fairly straightforward: If we want to solve an optimization problem, we typically want to find the point(s) where a function is at its extrema — where it is largest or smallest. Given that the gradient tells us the direction of increase of a function (and thus also the direction of decrease of a function, by negating the gradient), it seems intuitive that we could "follow" the gradient to find the extrema of the function. We will see some techniques for doing so in Section 5.2.

Finally, we have already encountered the **Jacobian** matrix in a number of contexts. The Jacobian can be thought of as the further generalization of the gradient to vector-valued functions. It is the matrix of first order derivatives of a vector-valued function. In short, each row of the Jacobian of a vector-valued function $\vec{f}$ is the gradient of each element of the column vector which comprises $\vec{f}$, in order.

5.1.3 The Hessian Matrix

The Hessian matrix for a function is a measure of the function's local curvature. It has many applications, some of which we will see in later sections. For now, we will simply define the Hessian.

Definition 5.3 (The Hessian) For a twice-differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, the **Hessian matrix H** is defined as:

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{5.1}$$

In other words, the Hessian is the matrix of all second-order partial derivatives of f.

5.1.4 Critical Points & Critical Point Tests

Before we begin considering more advanced methods of optimizing functions, we will briefly review one of the most fundamental methods of finding the extrema of a function: computing its critical points and evaluating their second-order properties.

Definition 5.4 (Critical Points of a Real-Valued Function) Given a continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, a point $x = (x_1, \ldots, x_n)$ is considered a **critical point of** f iff $f(x)$ exists and either $\nabla f(x) = \vec{0}$ or $\nabla f(x)$ does not exist.

In this section, we will only consider functions for which we can fairly easily compute the critical points algebraically. Section 5.2 and much of the rest of this chapter discusses methods for approximating the extrema of functions, which in some cases includes approximating their critical points.

In general, the procedure for finding the critical points of a function f is as follows: Compute ∇f, and then use algebra to find the roots of ∇f, as well as any points where at least one component of ∇f does not exist. The resulting set of points is the set of critical points for f.

Once we have the critical points of f, we need some way to find out whether or not they are extrema. Recall that while all extrema are critical points, not all critical points are extrema. If f is twice differentiable (meaning that we can compute its second-order gradient), then we can use the **second derivative test** to check its critical points.

For f of the form we have been considering thus far (i.e. $f : \mathbb{R}^n \to \mathbb{R}$), we can perform the second derivative test for a critical point c by the following procedure:

First, compute the Hessian matrix $\mathbf{H}$ (see Section 5.1.3) of f at c, and then the eigenvalues[1] of $\mathbf{H}$. If all of the eigenvalues λ_i of $\mathbf{H}$ are positive, then f has a local

[1]Recall that an eigenvalue λ of a matrix $\mathbf{A}$ is a root of the characteristic polynomial of $\mathbf{A}$, $|\mathbf{A} - \lambda \mathbf{I}| = \mathbf{0}$, where $\mathbf{I}$ is the identity matrix.

minimum at c, If all λ_i are negative, then f has a local maximum at c. If none of the λ_i are zero, but there are some negative and some positive, then c is a saddle point of f. Finally, if none of these conditions are met, then the second derivative test is inconclusive and we cannot classify c.

This test only gives us information about the local extrema of a function, whereas we are often more interested in the global extrema. In order to find these, we must take a slightly different approach. We have no general means of finding the global extrema for a function; we do not even have a guarantee that global extrema exist. As such, we will restrict our attention to a closed, bounded region of $B \in \mathbb{R}^n$.

Given B, we can find the region-global extrema by finding the critical points of our function f within B and the minimum and maximum points of f on the border of B. With these values, it is fairly straightforward to use comparison to find the true extrema of f in B.

5.2 Iterative Methods

Unfortunately, the exact methods provided by calculus are not always applicable to optimization problems. It may be intractable or otherwise infeasible to exactly find the critical points of a function, the second derivative test may be inconclusive, we may be unable to set bounds necessary to find the global extrema of a function, or some other property of the problem may prevent us from using exact methods. In these cases, it is useful to be able to fall back on a **numerical approximation** of the optima for the problem. Thus, we will now introduce a subset of the **iterative methods** of optimization. As their name implies, these methods iteratively improve on an approximation of the optimal value for a function. The key difference between the methods is their **update rule** — the means by which they determine how to change their estimate.

5.2.1 Newton's Method

Newton's method is the most basic method of iterative optimization. You have probably encountered Newton's method in the context of approximating function roots in a calculus class; in this section, we will discuss the applications of this method to the problem of function optimization.

Method

As we saw earlier (specifically, in Section 5.1.4), the minima of a function occur at its critical points — the input points for which the first derivative of the function is zero. This fact is the crux of Newton's method in optimization: with this in mind, we can use Newton's method to approximate the roots of the first derivative of a twice differentiable function, thus approximating the function's critical points. We can then use these points to find the minima of the function.

To put this more formally, given a function $f : \mathbb{R} \to \mathbb{R}$ such that f is twice-differentiable, we can pick an initial "guess" x_0 and compute:

$$x_{n+1} = x_n - \frac{\dot{f}(x_n)}{\ddot{f}(x_n)} \tag{5.2}$$

You will note that this equation is recursively defined. We use our guess value x_0 to initialize the computation. Iteration continues until x_n converges. We can test for convergence by examining the value of $\Delta = x_n - x_{n-1}$; when we have $\Delta \leq \epsilon$ for some $\epsilon \in \mathbb{R}$, we can stop the iteration because the value of x_n is not going to change significantly in future iterations. We elide the proof of convergence for Newton's method in these notes, but it is easily found online.

Newton's method is not limited to single-dimensional functions. For a function $f : \mathbb{R}^m \to \mathbb{R}$ where f is twice-differentiable, we can generalize the recurrence relation given in (5.2) as follows:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - (\mathbf{H}(\mathbf{x}_n))^{-1} \nabla f(\mathbf{x}_n) \tag{5.3}$$

where $\mathbf{H}$ is the Hessian of f (see Section 5.1.3) and ∇f is the gradient of f. Intuitively, all we are doing in this generalization is replacing the 1-dimensional notions of first and second derivatives with their multi-variable equivalents. Note also that $\mathbf{x}_n$ and $\mathbf{x}_{n+1}$ are both vectors, as is necessary for higher-dimensional functions.

Analysis

Although Newton's method *usually* converges to a low-error approximation of the desired result, there are some conditions which may cause it to diverge or otherwise fail to succeed. Some of the most pernicious of these conditions stem from improper selection of the initial guess, x_0. In this section, we will use notation corresponding to the single-dimensional case, but the principles we discuss are common to applications of Newton's method with any dimensionality.

For example, if we pick a value of x_0 for some function f such that $\ddot{f}(x_0) = 0$, then the update term $\frac{\dot{f}(x_0)}{\ddot{f}(x_0)}$ will be undefined from the first iteration, and the method cannot make progress toward converging. Such an x_0 is known as a **stationary point** of f.

Similarly, it may be possible that we pick x_0 for some function f such that x_0 is part of a **cycle**, e.g. for $x_1 = x_0 - \frac{\dot{f}(x_0)}{\ddot{f}(x_0)}$, we have $x_1 - \frac{\dot{f}(x_1)}{\ddot{f}(x_1)} = x_0$. This choice of x_0 will obviously lead to the method iterating forever, as it will keep making "progress" around the cycle without ever getting to the extrema of f. This is effectively the method overshooting the real solution.

Sometimes, properties of the function f can affect the ability of Newton's method to converge. In particular, we rely on $\dot{f}$ and $\ddot{f}$ being "well-behaved", at least in a neighborhood around each extremum. If this is not the case — for example, if $\ddot{f}$ does not exist at some extremum, then Newton's method will diverge instead of converging to this extremum. Even if $\ddot{f}$ does exist at a given extremum, we might also run into trouble if it is discontinuous at the extremum in question. In general, for Newton's method to succeed, f needs to be smooth in a neighborhood around each extremum.

Finally, Newton's method may be computationally expensive. It uses the exact, directly computed forms of both $\dot{f}$ and $\ddot{f}$, which may be difficult or infeasible to find. We can imagine that this might be the case for certain choices of f where f is very complicated, estimated from data, or otherwise not easily differentiated analytically. In these cases, we sometimes use **Quasi-Newton methods**, which work around the need for exact derivative computation. These methods are not covered in this class, however.

For more on Newton's method, please see Section 8.6.1 of Goodfellow et al. [4].

5.2.2 Gradient Descent

Gradient descent is a method of optimizing a function by **greedily following its gradient downward**. It is one of the most common methods of optimization used, particularly for machine and deep learning. This popularity stems from the simplicity and effectiveness of the technique.

Method

The core of gradient descent is the notion that a function decreases fastest in the opposite direction of its gradient. While this does not guarantee that following the negative gradient will lead to a global minimum, it does mean that we will converge to a local minimum[2]. The formula for basic gradient descent is as follows:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla f(\mathbf{x}_n) \tag{5.4}$$

[2]Assuming certain properties of f, which we won't discuss here.

where $f : \mathbb{R}^m \to \mathbb{R}$ is the function we are optimizing, x_n is a previous estimate of the optimum of f, and γ_n is a scaling coefficient. As in Section 5.2.1, we begin iteration with an arbitrary estimate x_0 and conclude when some stopping condition is met — usually when the rate of change of the x_i drops below some threshold.

Analysis

Gradient descent is simpler and slower but more general than Newton's method. Whereas Newton's method requires f to be twice-differentiable, we only need to compute the first derivatives of f for gradient descent. Further, computing the gradient is a cheaper operation than computing the update term for Newton's method[3]. However, Newton's method will usually converge quadratically to a minimum, while gradient descent tends to be much slower. Additionally, gradient descent has more of a tendency to get "stuck" near saddle points and near minima, as the magnitude of the gradient decreases.

5.3 Constrained Optimization

5.3.1 Lagrange Multipliers

All of the techniques we have seen thus far have been applicable to **unconstrained** optimization problems, wherein we simply want to minimize some function and have no constraints on the region of input values we can use for this task. In this section, we will discuss a method of solving **constrained** optimization problems known as the method of **Lagrange multipliers**.

Constrained Optimization

First, we define a constrained optimization problem as follows:

Definition 5.5 Given a function f, we wish to find the input $\mathbf{x}$ such that $f(\mathbf{x})$ is minimized, while satisfying the equations $g_i(\mathbf{x}) = c_i$ and inequalities $h_j(\mathbf{x}) \leq d_j$, for constants c_i and d_j, $1 \leq i \leq n$, $1 \leq j \leq m$.

In other words, we are solving an optimization problem as usual, but some possible solutions might not be acceptable (if they violate the constraints). While we could conceivably use an unconstrained optimization method and check each possible solution against the constraints, this is not a very efficient approach. It is better to use a method (such as Lagrange multipliers or linear programming —

[3] This follows because the gradient is only one part of the update term for Newton's method.

which we will discuss in Section 5.3.2) which takes into account the constraints in the search for optima.

Method

We can only apply the method of Lagrange multipliers to a subset of constrained optimization problems: those where we have only **equality** constraints. The intuition behind Lagrange multipliers can be considered to stem from two insights. First, because we are only working with equality constraints, we know that the solution to the optimization problem must lie in the intersection of the **level sets** of all of its constraints.

Definition 5.6 A **level set** of a function $g(\mathbf{x})$ is the set $G = \{\mathbf{x} | g(\mathbf{x}) = c\}$ for some constant c. In other words, it is the set of points which g maps to the same value. For $g : \mathbb{R}^2 \to \mathbb{R}$, this set is also called a **contour** of g.

Second, we know that the optimum for f will be found at a point where f is unchanging. This will happen either where $\nabla f = \mathbf{0}$ or on a contour of f.

Putting these two facts together and noting that the gradient of a function is perpendicular to its contours[4], we can construct a function which will have a zero gradient at the possible optima of f.

Specifically, we will construct the **Lagrangian** as follows. For a function $f : \mathbf{R}^n \to \mathbf{R}$ and constraints $g_i : \mathbf{R}^n \to \mathbf{R}$ for $i = 1, \ldots, n$, the Lagrangian is

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) - \sum_{i=1}^{n} \lambda_i (g_i(\mathbf{x}) - c_i) \tag{5.5}$$

We include the vector λ to account for the fact that although the gradients of f and the g_i must be parallel, they need not be of the same magnitude. Then, by computing $\nabla \mathcal{L}$ and solving $\nabla \mathcal{L} = \mathbf{0}$, we can find the candidate optima for f with respect to the n constraints g_i. To solve this equation, we will note that for f and the g_i as given, this is equivalent to solving the $n + m$ equations described by the gradient vector of $\mathcal{L}$. This system will be in the variables $x_1, \ldots, x_n$ (for the input to f) and $\lambda_1, \ldots, \lambda_m$ (for the Lagrange multipliers), and thus will be solvable (as the number of variables and number of equations in the system are equal).

Analysis

The key properties to note for Lagrange multipliers are as follows. First, this method is only applicable for **equality constraints**; problems with other kinds of

[4]We are using terminology for the case of $f : \mathbb{R}^2 \to \mathbb{R}$ for convenience; the formulae and notions generalize to higher dimensions.

constraints must either be transformed into the equality constraint form or solved through another method. Second, as with most of the methods we have discussed thus far, the function f must be differentiable for the method to be applicable. Finally, note that the solutions to the system of equations discussed above must be checked for optimality; they may not necessarily be minima or maxima. For more on Lagrange multipliers, see Klein [5] or Dawkins [3].

5.3.2 Linear Programming

We will conclude our brief overview of optimization techniques by looking at another technique for constrained optimization: **linear programming**.

Method

Linear programming is a means of solving the subset of constrained optimization problems where both the objective and all of the constraints are linear functions. At a high level, linear programming constructs a **feasible region** where solutions must lie in the set of constraints, and then it uses one of several techniques to search this region for the optima.

If an objective function $f : \mathbf{R}^n \to \mathbf{R}$ and set of constraints $g_i : \mathbf{R}^n \to \mathbf{R}$ are all linear, then we can express the constrained optimization problem in terms of a set of matrix products of the matrices corresponding to the systems of linear inequalities and equations comprising the problem. For f and g_i as given, where $f(\mathbf{x}) = c_0x_0 + \ldots + c_nx_n$ and $g_i(\mathbf{x}) = a_{i,0}x_0 + \ldots + a_{i,n}x_n \leq b_i$, we can write the constrained optimization problem in the form:

$$\text{minimize } \mathbf{c}^T\mathbf{x} \tag{5.6}$$
$$\text{maintaining } \mathbf{Ax} \leq \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0} \tag{5.7}$$

where $\mathbf{c}$ is the vector composed of the c_i from f, A is the matrix composed of the $a_{i,j}$ from g_i, and $\mathbf{b}$ is the vector composed of the b_i.

Taken together, the inequalities in (5.7) describe the convex region composed by intersecting the regions that each individual constraint $g_i \leq b_i$ describes. Once we have this feasible region and the appropriate description of f, which is the expression in (5.6), we can use one of several methods to find the optimal input for f. In these notes, we will discuss at a high level the **simplex algorithm**, which was among the first algorithms for solving linear programs.

The basic method of the simplex algorithm is straightforward: Starting at some vertex of the region described by the constraints, we walk along the edges of the region following the direction of increase of f (assuming that we are solving a maximization problem) until we reach a vertex from which no edge exists along which f increases.

The viability of this simple algorithm is based on two clever insights, which we will state without proof here. First, it can be shown that the optimal value for the objective function of a linear program in the above form will occur on a vertex of the region described by the constraints, if any optimal value exists. Second, it can also be shown that all vertices of the region which do not have the optimal value for f have an edge along which f increases and which leads to a vertex with a greater value of f. Thus, by following edges until f cannot increase, we ensure that we can find the optimal value of f with respect to the constraints.

Analysis

Linear programming is very powerful, and there exist many good tools (and more sophisticated algorithms) for solving linear programs. The main consideration in its use is that both the objective function and all constraints must be completely linear, which limits the class of constrained optimization problems to which it is applicable. Additionally, although there exist guaranteed polynomial-time algorithms for solving linear programs, simplex is only usually polynomial-time and may run longer.

Bibliography

[1] P. Dawkins. Calculus I, 2016. URL http://tutorial.math.lamar.edu/Classes/CalcI/CalcI.aspx.

[2] P. Dawkins. Calculus II, 2016. URL http://tutorial.math.lamar.edu/Classes/CalcII/CalcII.aspx.

[3] P. Dawkins. Calculus III, 2016. URL http://tutorial.math.lamar.edu/Classes/CalcIII/CalcIII.aspx.

[4] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. URL http://www.deeplearningbook.org.

[5] D. Klein. Lagrange multipliers without permanent scarring. URL https://people.eecs.berkeley.edu/~klein/papers/lagrange-multipliers.pdf.