Lecturer: Nicholas Boumal Scribe: Kiran Vodrahalli

# Contents



# 1 Introduction

I prepared a story with two parts. One is optimization on manifolds. The second part is optimization on manifolds to solve semidefinite relaxations.

Optimization on manifolds is nonlinear programming, and is concerned with solving minimization for smooth $f$. What is particular about the domain of $f$ is that it is defined on a manifold $M$. To understand how we might do this, let's look at classical gradient descent for unconstrained optimization. Can we do this on a more complicated space. Everything here will be nonconvex. The first part will be a general framework for nonconvex optimization. I'll show particular cases where we can actually get a global optimizer, for particular examples.

# 2 Optimization on Manifolds

Let's say we can get at least critical points. If you want to think about $M$, think about the sphere.

What do we do in gradient descent? We take steps along the gradient. If you are smart about how you pick step sizes, this will converge to a point where the gradient is zero. This may be a local optimizer. But at least you go to a critical point.

Now we do the same thing on a sphere (just the shell). Then I will argue you can do the same thing even on an abstract manifold.

We have a cost function over the sphere. The idea is exactly the same. Locally pick the best one. Let's restrict our attention to tangential directions to the manifold. Among all tangent vectors, we will pick what looks like the best direction. We follow this direction. You could follow the generalization of line on a curved space, a geodesic. As long as you move in right direction at first, I will be happy. I want to move in direction I chose. Then you iterate this and it may converge to a critical point. The theory around manifolds say yes.

We only used notions of directions along which it is ok to move: tangent space, tangent vector. The second thing I did is look at all tangent vectors in that space and compared them, since we wanted a notion of "steepest". So you might want an inner product on the tangent spaces. You're actually transforming the set into a Riemannian manifold. There's

a notion of linearization (tangent space). We're asking that on each tangent space there's a metric, so there's no discontinuity in the way you pick inner products. This is a backstory for why it makes sense to consider optimization on manifolds, it's very nice.

We're trying to not bring in curvature into the mix since it is not easy to work with, so you just use line search. This is different from projected gradient - why is it different? Why do we do approximation by hyperplane? When you do projected gradient, you use the ambient space, you have a notion of gradient in the embedding space. If you do the same thing in the framework of optimization on manifolds, there is an intrinsic notion of gradient on the sphere, we separate it from the notion from the gradient in the embedding space, which may or may not exist. If you try to use the same metric on the tangent space, you can just project the Euclidean gradient to the tangent space. The advantage of the manifold approach is becasue the gradient is defined intrinsically, you can do this if there's no ambient space. Well the Whitney embedding theorem guarantees there is an embedding, but that's not practical to work with. This allows us to work with the manifold directly. For optimization on manifolds, you have retractions (approximate geodesics); the requirements there are very lax.

A geodesic is a generalization of a line. You want to extend acceleration of the line to weird manifolds; acceleration is 0 means it's a geodesic (FROM THE POINT OF VIEW OF SOMEONE WHO LIVES ON THE MANIFOLD). The geodesics will be the great circles. How do you compute a geodesic locally? A geodesic satisfies a differential equation. Either I integrate the equation (very costly), or I think of the formula (which usually doesn't exist). The next best thing as a computer scientist is I'll do something which looks like it. I'll use that definition of 'close to' and will prove some things with it. Our definition is very lax. If I am at the current point and follow geodesic with velocity 0, I should stay at my current point. The second requirement is that locally, up to first order, the retraction and the geodesic should agree, locally (in the Taylor polynomial sense). This will be enough to ensure convergence, since eventually you will make very small steps. You can say something about projected gradient descent. Intuitively this should be a reasonable approximation of the geodesic, and it's true. There are some conditions and there is a paper, and this will give you something which agrees up to second-order. Most retractions in our toolbox use exactly this. The step-sizes matter in the analysis, so you may have to do extra work to get the same convergence rates.

Here we want to think of optimization on a manifold in the same way we think of unconstrained optimization. We want to think of the manifold as the only thing that exists. You can define all of these things even if the manifold is the only thing that exists. To optimize, we only use that the search space is a Riemannian manifold. This works for abstract manifolds and quotient spaces too. This could come up in practice as follows: Consider sensor localization. You measure distances between pairs of people, and given these measurements, you can try to figure out where everyone is. There is a fundamental indeterminacy in this problem; namely, translational. There's also rotational indeterminancy. If you look at my unknowns which are $x, y$ for everyone, they are redundant since there are extra degrees of freedom you cannot account for. Now if you try to write this as an optimization problem, the cost function will never have isolated optimizers (since points are equivalent to other points). If you open a book about nonlinear programming, then these all start with assuming you're over an isolated optimizer. So these don't apply. We're not talking about a global convergence rate, we're talking about local convergence rate, which says that eventually gradient descent will enter the neighborhood at a local minimum (converges exponentially fast here).

These require you have an isolated optimizer; this is NP-hard in general.

If you try to formulate sensor-network localization, and you want to invoke a theorem giving fast local convergence, you can't: There's fundamental indeterminancy, there's no local strong convexity.

Well you can solve this by just quotienting out rotation and translation which happens to be a quotient space which is a manifold (for instance, matrix completion: $UV^T$, can just insert an orthogonal matrix in there. You want to mod out by orthogonal matrices). You end up with abstract space where each point is an equivalence class.

At the end of the day you represent everything with matrices anyways. You prove on theorem once, and it applies to everything. Check out the book "Optimization algorithms on matrix manifolds". This was written by my PhD advisor; we wrote a program to solve many optimization problems over manifolds.

**Example 2.1.** Max-Cut relaxation SDP.

$$\min_{Y:n\times p} \operatorname{Tr}(AYY^T)$$

with $\operatorname{diag}(YY^T) = 1$ (each row of $Y$ has unit norm), where $A$ is the adjacency matrix of a graph. This is an optimization problem over a manifold. This is just a Cartesian product of spheres. The tangent space is given by $\{Y^{\cdot} : \operatorname{diag}(Y^{\cdot}Y^T + Y(Y^{\cdot})^T) = 0\}$: a linear subspace. If you want to optimize, you can get this function classically. Once you have your point and your direction, you need a way to retract: Well, you can just normalize each row. The only difference with projected gradient is the intermediate step of projecting to tangent space. In practice, you may want to do something with fast gradient descent. There's something called Riemannian trust regions.

Now, note that this problem is solveable in a convex setting. However, I will argue that optimizing over the manifold is faster than solving the SDP. In particular, for a very very large matrix, the interior point method blows up (memory and gives nonsense!) For this manopt library, it takes five seconds on my laptop and we have a proof that it converges quickly.

## 2.1 Convergence guarantees for Riemannian Gradient Descent

The rule of thumb for optimization on manifolds is you can expect as much as you would get in unconstrained optimization (smooth unconstrained in $\mathbb{R}^n$, do gradient descent; in the limit you will get a critical point and locally you will get linear convergence rate (gradient descreases exponentially)). You get the same things on the manifold. You can also control the gradient with the number of iterations. In the Euclidean case, you need $1/\epsilon^2$ and Lipschitz assumption. Similarly, you can get $\|\operatorname{grad}(f)\| \leq \epsilon$ in $\mathcal{O}(1/\epsilon^2)$ iterations under Lipschitz assumptions. It's proved very similarly to the other case. It gets tricky when you have to assume Lipschitzness in the gradient. To compare gradients at different points on the manifold is tricky, because there are different tangent spaces. So you have to parallel transport them to each other. These are annoying concepts, so we instead used an equivalent definition of Lipschitz gradient in $\mathbb{R}^n$, which isn't equivalent for manifolds. You basically say the function is well approximated by first-order Taylor expansion. We started this way and worked backwards. The Lipschitz assumption is the Taylor approximation one. We made

that assumption not on $f$, but rather on the retract of $f$. The retraction is a map from the tangent space at $X$ to the manifold. If you compose $f$ with the retraction is that you get a pullback (the cost function to the tangent space), now you get a function defined on the tangent space and that's a linear space. you assume this kind of Lipschitzness on all of tangent spaces (you need global Lipschitz on all tangent spaces, but we control that). After that the proof is amazingly similar.

You can also compute global convergence to second order critical points (Hessian is close to semidefinite; second-order necessary conditions). You get a quadratic convergence rate locally. As long as the gradient is large, you don't need second order information. When the gradient is small, now you need to use second-order. While $\text{Hess}(x) \geq -\epsilon I$, you iterate.

# 3 Max-Cut Optimality

Given a graph, split nodes into two classes and select a subset of nodes of the graph and split them. Some of the edges will go from left to right, count how many there are. We want to choose how to put nodes on the left and right, so that there are as many edges as possible. This is NP hard. You probably know also about the semidefinite relaxation for max cut: $\min_X Tr(AX)$ such that $\text{diag}(X) = 1$ for $X$ PSD. IF this has a solution of rank 1, you get a true global optimizer. By Goemans-Williamson and randomized projection you can get the best cut within 87% of optimal $X$ to $\{\pm 1\}^n$. If you want the actual value of the cut, you replace $A$ with the Graph Laplacian and divide by 4. This is convex, but SDPs don't really work when the problem is big. If you do this with a 2000-node graph on a laptop, memory issues will come into play. But manopt solves this with 6 seconds and no memory issues. A fair comparison would be to gradient descent: You will have to project to the cone though, and that will kill you. You can look at Lagrange relaxation kind of things (iterate eigenvalue). But as long as you output $2000 \times 2000$ matrix, you're doomed to be slow. But you can use Multiplicative Weights method, but then you will get $1/\epsilon$ instead of $\log(1/\epsilon)$. If you want to solve this SDP and you add that the rank must be bounded by $\sqrt{2n}$, it's not NP hard. Max-Cut SDP has a low-rank solution. The original search space is the cone of PSD matrices, and intersect with affine plane; this is convex set. It also happens to be compact. Now, if you optimize a linear function (concave) over a compact convex set, one of the extreme points will be a global optimizer (linear programming: vertices of polyhedron). So you know at least vertex is a global optimizer. Then, all exterem points of the set there is an optimal $X$ whose rank $r$ satsifies $r(r+1)/2 \leq n$. Thus, $r \leq \sqrt{2n}$ is fine. In general, $n$ is the number of constraints. This is not for general SDP.

If there are only one or two constraints and your search space is compact, then the SDP always has a solution of rank 1. Consider the trust region problem:

$$\min x^T A x + b^T x + c$$

with the constraint $x^T x = 1$. Now this you can write as $[x^T 1][A, b; b, c][x1]^T$. Call the middle matrix $C$. Introduce matrix $Z = [x1]^T[x1] = [xx^T, x; x^T, 1]$. Then we do

$$\min_Z \text{Tr}(CZ)$$

with $\text{Tr}(Z) = 1$, and $Z_{n+1,n+1} = 1$, with $Z$ PSD. Recall our matrix is $n \times p$. So $p = 2$ means $p(p+1) = 6 > 4$. Now $\frac{p(p+1)}{2} \leq 2$, then $p(p+1) \leq 4$, implying $p = 1$ exists. Rank 1 would

be very useful. 2 is believable. One of the optimizers has rank 1, but there are others which are not rank 1. Interior point methods will not find this typically maybe. There's also a polynomial way of reducing rank 2 solutions to rank 1, since this can only happen when the solution is the midpoint between two extreme points, and you can use linear algebra tricks to get the rank 1 solution.

Now, for a max-cut partial relaxation, you can relax the rank constraint partially, by saying that $\text{rank}(X) \leq p$. This is a hard problem to look at. Parametrize $X = YY^T$ with $Y$ size $n \times p$. You can now explore this set of matrices. This is called the Burer-Monteiro approach works fantastically well in practice, but there wasn't much theory to speak of. The best result I could come up with Bandeira is if you look at the problem on a manifold, and take $p$ big enough to include all extreme points, for almost all $A$, all SOCP's are optimal. Saddle points are escapable since Hessians must have negative eigenvalues. That is: $\min_{Y:n\times p} \text{Tr}(AYY^T)$ such that $\text{diag}(YY^T) = 1$. If $p(p+1)/2 > n$, you get an explicit dual certificate for the SDP, now you just have to show that second-order conditions imply the other. And it works for almost $A$, we don't actually have an example of a bad $A$. Before we got this result, I tried to come up with a deterministic result, and the best is $p > n/2$; for all $A$, all SOCPs are optimal.

# 4 Conclusion

The take home message is that optimization on manifolds applies to a wide range of applications, and it's easy to try with Manopt. It comes with the same guarantees as unconstrained nonlinear optimization. For some problems you get global optimality.