Notes for 2016-04-18

Lay of the Land

In the landscape of continuous optimization problems, there are two axes in which things can be hard. In the first axis, we have the global structure of the problem:

- Easiest: Quadratic functions.
- Harder: General convex functions.
- Harder: Nonconvex functions with special structure.
- Harder: Nonconvex functions with some smoothness.
- Hardest: Nonconvex functions with wild variations.

Along the second axis, we have the local structure that is readily available:

- Easy: Hessians and gradients are available.
- Harder: Gradients are available, but Hessians are not (or are too costly to use).
- Hardest: Gradients are not available; only function evaluations.

For the past two weeks, we have discussed the related problems of optimization and nonlinear equation solving. Our standard method (Newton iteration) applies to both problems. In the categorization above, this is "easy" in terms of assuming we know lots of local structure: gradients and Hessians. Newton converges quickly from good guesses, and can be effectively globalized, making it useful not only for convex problems but also for harder non-convex problems (given a good initial guess). But Newton uses many derivatives, and factorizations that may be expensive. We have discussed a few methods (e.g. Broyden) that require fewer derivatives, but we can go further.

Today we discuss optimization in one of the hard cases. For this class, we will not deal with the case of problems with very hard global structure, other than to say that this is a land where heuristic methods (simulated annealing, genetic algorithms, and company) may make sense. But there are some useful methods that are available for problems where the global structure is not so hard as to demand heuristics, but the problems are hard in that they are "black box" we are limited in what we can compute to looking at function evaluations.

Before describing some methods, I make a plea that you consider these only after having thoughtfully weighed the pros and cons of gradient-based methods. If the calculus involved in computing the derivatives is too painful, consider a computer algebra system, or look into a tool for automatic differentiation of computer programs. Alternately, consider whether there are numerical estimates of the gradient (via finite differences) that can be computed more quickly than one might expect by taking advantage of the structure of how the function depends on variables. But if you really have to work with a black box code, or if the pain of computing derivatives (even with a tool) is too great, a gradient-free approach may be for you.

Model-based methods

The idea behind Newton's method is to successively minimize a quadratic *model* of the function behavior based on a second-order Taylor expansion about the most recent guess, i.e. $x^{k+1} = x^k + p$ where

$$\operatorname{argmin}_p \phi(x) + \phi'(x)p + \frac{1}{2}p^T H(x)p.$$

In some Newton-like methods, we use a more approximate model, usually replacing the Hessian with something simpler to compute and factor. In simple gradient-descent methods, we might fall all the way back to a linear model, though in that case we cannot minimize the model globally – we need some other way of controlling step lengths. We can also explicitly incorporate our understanding of the quality of the model by specifying a constraint that keeps us from moving outside a "trust region" where we trust the model to be useful.

In derivative-free methods, we will keep the basic "minimize the model" approach, but we will use models based on interpolation (or regression) in place of the Taylor expansions of the Newton approach. There are several variants.

Finite difference derivatives

Perhaps the simplest gradient-free approach (though not necessarily the most efficient) takes some existing gradient-based approach and replaces gradients with finite difference approximations. There are a two difficulties with this approach:

- If $\phi : \mathbb{R}^n \rightarrow \mathbb{R}$, then computing the $\nabla \phi(x)$ by finite differences involves at least $n+1$ function evaluations. Thus the typical cost per step ends up being $n+1$ function evaluations (or more), where methods that are more explicitly designed to live off samples might only use a single function evaluation per step.

- The finite difference approximations depends on a step size h, and their accuracy is a complex function of h. For h too small, the error is dominated by cancellation, revealing roundoff error in the numerical function evaluations. For h large, the error depends on both the step size and the local smoothness of the function.

Linear models

A method based on finite difference approximations of gradients might use $n+1$ function evaluations per step: one to compute a value at some new point, and n more in a local neighborhood to compute values to estimate derivatives. An alternative is to come up with an approximate linear model for the function using $n+1$ function evaluations that may include some "far away" function evaluations from previous steps.

We insist that the $n+1$ evaluations form a simplex with nonzero volume; that is, to compute from evaluations at points $x_0, \ldots, x_n$, we want $\{x_j - x_0\}_{j=1}^n$ to be linearly independent vectors. In that case, we can build a model $x \mapsto b^T x + c$ where $b \in \mathbb{R}^n$ and $c \in \mathbb{R}$ are chosen so that the model interpolates the function values. Then, based on this model, we choose a new point.

There are many methods that implicitly use linear approximations based on interpolation over a simplex. One that uses the concept rather explicitly is Powell's COBYLA (Constrained Optimization BY Linear Approximation), which combines a simplex-based linear approximation with a trust region.

Quadratic models

One can build quadratic models of a function from only function values, but to fit a quadratic model in n-dimensional space, we usually need $(n+2)(n+1)/2$ function evaluations – one for each of the $n(n+1)/2$ distinct second partials, and $n+1$ for the linear part. Hence, purely function-based methods that use quadratic models tend to be limited to low-dimensional spaces. However, there are exceptions. The NEWUOA method (again by Powell) uses $2n+1$ samples to build a quadratic model of the function with a diagonal matrix at second order, and then updates that matrix on successive steps in a Broyden-like way.

Response surfaces

Polynomial approximations are useful, but they are far from the only methods for approximating objective functions in high-dimensional spaces. One popular approach[1] is to use *radial basis functions*; for example, we might write a model

$$s(x) = \sum_{j=1}^{m} c_j \phi(\|x - x_j\|)$$

where the coefficients c_j are chosen to satisfy m interpolation conditions at points $x_1, \ldots, x_m$. Another option is to use a Gaussian process model to predict how the objective function behaves between objectives; this is used, for example, in an optimizer called EGO. There are a variety of other surfaces one might consider, though.

In addition to fitting a surface that interpolates known function values, there are also methods that use *regression* to fit some set of known function values in a least squares sense. This is particularly useful when the function values have noise.

Pattern search and simplex

So far, the methods we have described are explicit in building a model that approximates the function. However, there are also methods that use a

[1]At least, it is popular that I've gotten pulled into working on it. Your TA does, too! See `https://github.com/dme65/pySOT`.

systematic search procedure in which a model does not explicitly appear. These sometimes go under the heading of "direct search" methods.

Nelder-Mead

The Nelder-Mead algorithm is one of the most popular derivative-free optimizers around. For example, this is the default algorithm used in MATLAB's `fminsearch`. As with methods like COBYLA, the Nelder-Mead approach maintains a simplex of $n + 1$ function evaluation points that it updates at each step. In Nelder-Mead, one updates the simplex based on function values at the simplex corners, the centroid, and one other point; or one contracts the simplex.

Visualizations of Nelder-Mead are often quite striking: the simplex appears to crawl downhill like some sort of mathematical amoeba. But there are examples of functions where Nelder-Mead is not guaranteed to converge to a minimum at all.

Hook-Jeeves and successors

The basic idea of *pattern search* methods is to test points in a pattern around the current "best" point. For example, in the Hook-Jeeves approach (one of the earliest pattern search methods), one would at each move evaluate $\phi(x^{(k)} \pm \Delta e_j)$ for each of the n coordinate directions e_j. If one of the new points is better than $x^{(k)}$, it becomes $x^{(k+1)}$ (and we may increase Δ if we already took a step in this direction to get from $x^{(k-1)}$ to $x^{(k)}$. Of $x^{(k)}$ is better than any surrounding point, we decrease Δ and try again. More generally, we would evaluate $\phi(x^{(k)} + d)$ for $d \in \mathcal{G}(\Delta)$, a *generating set* of directions with some scale factor Δ.

Summarizing thoughts

Direct search methods have been with us for more than half a century: the original Hook-Jeeves paper was from 1961, and the Nelder-Mead paper goes back to 1965. These methods are attractive in that they require only the ability to compute objective function values, and can be used with "black box" codes – or even with evaluations based on running a physical experiment!

Computing derivatives requires some effort, even when automatic differentiation and related tools are available, and so gradient-free approaches may also be attractive because of ease-of-use.

Gradient-free methods often work well in practice for solving optimization problems with modest accuracy requirements. This is true even of methods like Nelder-Mead, for which there are examples of very nice functions (smooth and convex) for which the method is guaranteed to mis-converge. But though the theoretical foundations for these methods have gradually improved with time, the theory for gradient-free methods is much less clear-cut than the theory for gradient-based methods. Gradient-based methods also have a clear advantage at higher accuracy requirements.

Gradient-free methods do *not* free a user from the burden of finding a good initial guess. Methods like Nelder-Mead and pattern search will, at best, converge to local minima. Methods such as simulated annealing may have better luck in finding global minima, but it is still a hard problem in general. Gradient-free methods may also have difficulty with functions that are discontinuous, or that have large Lipschitz constants.

In many areas in numerics, an ounce of analysis pays for a pound of computation. If the computation is to be done repeatedly, or must be done to high accuracy, then it is worthwhile to craft an approach that takes advantage of specific problem structure. On the other hand, sometimes one just wants to do a cheap exploratory computation to get started, and the effort of using a specialized approach may not be warranted. An overview of the options that are available is useful for approaching these tradeoffs intelligently.

References

Our textbook does not have much discussion of gradient-free optimization. For further reading at the same level as these notes (though by a much more knowledgable authority), I recommend "A view of algorithms for optimization without derivatives" by M. J. D. Powell (2007). There is also a beautiful survey of direct search methods by Kolda, Lewis, and Torczon from 2003 ("Optimization by direct search: new perspectives no some classical and modern methods," *SIAM Review*, vol 45, pp. 385–482). For more detail, try *Introduction to Derivative-Free Optimization* by Conn, Scheinberg, and Vicente (SIAM, 2009). The Cornell library subscribes to SIAM's eBook service, so if you are on campus, you can get to the electronic version of this book.