18.335 Problem Set 6

Due Monday, 30 November.

Problem 1: Convexity

Recall from class that a *convex function* is a function $f(x)$ such that $f(\alpha x+(1-\alpha)y) \le \alpha f(x)+(1-\alpha)f(y)$, for $\alpha \in [0,1]$, where $x \in \mathbb{R}^n$. The inequality is $\ge$ for *concave* functions, and = for *affine* functions. A *convex set* $X \subseteq \mathbb{R}^n$ satisfies the property that if x and y are in X, then so is the line connecting x and y, i.e. so is $\alpha x+(1-\alpha)y$ for $\alpha \in [0,1]$.

Show that the feasible region satisfying m constraints $f_i(x) \le 0$, $i = 1,\ldots,m$, is a convex set if the constraint functions are convex.

Problem 2: Adjoints

Consider a recurrence relation $x^n = f(x^{n-1}, p, n) \in \mathbb{R}^K$ with initial condition $x^0 = b(p)$ and P parameters $p \in \mathbb{R}^P$, as in the notes from class (see the handout on the web page). In class, and in the handout, we used the adjoint method to derive an expression for the derivatives $\frac{dg^N}{dx}$ of a function $g(x^n, p, n) \triangleq g^n$ evaluated after N steps.

In this problem, suppose that instead we want the derivative of a function G that depends on the values of x^n from *every* $n \in \{0,1,\ldots,N\}$ as follows:

$$G(p,N) = \sum_{n=0}^{N} g(x^n, p, n)$$

for some function g.

(a) One could simply use the adjoint formula from class to obtain $\frac{dG}{dp} = \sum_n \frac{dg^n}{dp}$. Explain why this is inefficient.

(b) Describe an efficient adjoint method to compute $\frac{dG}{dp}$. (Hint: modify the recurrence relation for λ^n from class to compute $\sum_n \frac{dg^n}{dp}$ via the results of a *single* recurrence.)

(c) Apply this to the example 2×2 recurrence and g function from the notes, and implement your adjoint method in Matlab. Check your derivative $\frac{dG}{dp}$ with $N = 5$ against the center-difference approximation $\frac{dG}{dp_i} = [G(p_i+\delta) - G(p_i-\delta)]/2\delta$ for $p = (1,2,3,4,5)^T$ and $\delta p = 10^{-5}$.

Problem 3: Nonlinear fitting

One well-known application of nonlinear optimization is the problem of nonlinear fitting, also called nonlinear regression: you have a bunch of data points (x_i, y_i), and you want to fit them to a function $f(x)$ with some unknown parameters, where the parameters enter the function in a nonlinear way. A common problem is to find a *least-squares fit*, the fit parameters minimizing the sum-of-squares error $\sum_i [f(x_i) - y_i]^2$. Here, you'll do nonlinear fitting using the NLopt library, via Matlab. It is installed on Athena (x86 Linux machines only)—do "add stevenj" at the Athena prompt, and then, in Matlab, type `path(path,'/mit/stevenj/Public/nlopt_i386_deb50/matlab'` to tell Matlab where to find NLopt.

In this problem you will fit a set of data points to a Lorentzian line shape (which often arises in resonant processes, e.g. looking at absorption lines in spectroscopy, or in NMR experiments). (There are actually much more sophisticated and robust techniques for the specific problem of fitting Lorentzian peaks, but we won't use them here.) That is, we have a bunch of data points (x_i, y_i) that we want to fit to a curve of the form:

$$f(x) = \frac{A}{(x-\omega)^2 + \Gamma^2}$$

in terms of some unknown "resonant frequency" ω, "lifetime" Γ, and amplitude A.

The file lorentzdata.m on the course page contains a function [x,y] = lorentzdata(N, A, ω, Γ, *noise*) that generates N random "data" points based on the Lorentzian peak with paramters A, ω, and Γ, with some random noise of amplitude *noise* added in. The file lorentzfit.m computes the sum-of-squares error given a vector p = $[A, \omega, \Gamma]$ of the fit parameters and the point arrays x and y.

(a) Try fitting 200 random data points from $A = 1$, $\omega = 0$, and $\Gamma = 1$, with noise ± 0.1, to minimize the sum-of-squares error, using:

```
[x,y] = lorentzdata(200, 1, 0, 1, 0.1);
stop.xtol_rel = 1e-8; stop.verbose = 1;
[p, errmin] = nlopt_minimize(NLOPT_LN_NEWUOA_BOUND,
                             @lorentzfit, {x,y},
                             [-inf,-inf,0], [inf,inf,inf],
                             [0,1,2], stop);
```

This is calling NLopt to do the minimization with a derivative-free algorithm called NEWUOA that constructs approximate quadratic models of the objective function (lorentzfit). Note the [-inf,-inf,0], [inf,inf,inf] arguments, which give lower and upper bounds for the parameters $[A, \omega, \Gamma]$ (which are unconstrained except that we require $\Gamma \geq 0$). The last line gives an initial "guess" $A = 0$, $\omega = 1$, $\Gamma = 2$. The optimization terminates when an estimated fractional error 10^{-8} is reached on the parameters. Plot the fit curve, which is returned in the parameters p0, and the data points to verify that the fit looks reasonable. Do the fit parameters vary significantly depending on the initial guess?

(b) Because of the line "stop.verbose=1", you can see how many function evaluations are required by the optimization algorithm to converge. Change stop.xtol_rel to zero (no tolerance), and set stop.minf_max = errmin0 * 1.0001. This will stop the optimization when the objective function gets within 10^{-4} of the previous value (errmin0), which allows us to compare different algorithms easily—we can see how many iterations are required to reach the same value of the objective function. With this new stopping criteria, compare the number of iterations for NEWUOA (NLOPT_LN_NEWUOA_BOUND) with a different algorithm that constructs only linear approximations of the objective function (NLOPT_LN_COBYLA), with the Nelder-Mead simplex algorithm (NLOPT_LN_NELDERMEAD).

(c) All of the previous algorithms were derivative-free (they don't use the gradient of the objective, only its values). However lorentzfit also optionally returns the gradient, and we can use that. With the same stopping criterion as in the previous part, try the gradient-based algorithms NLOPT_LD_MMA and NLOPT_LD_LBFGS. LBFGS iteratively constructs quadratic approximations that try to match the first and second derivative (similar in spirit to NEWUOA), while MMA uses only the first derivative; what is the impact of this on the rate of convergence?

(d) Now, modify lorentzdata and lorentzfit so that the data is the sum of *two* overlapping Lorentzians: $A = 1$, $\omega = 0$, and $\Gamma = 1$, and $A = 2$, $\omega = 1$, and $\Gamma = 1.5$, with noise = 0.01. Try fitting again with any of the algorithms from above, changing the stopping criteria to stop.xtol_rel=1e-4 and stop.minf_max=-inf. Try different starting points; do you get more than one local minimum? How close does your initial guess need to be to the "correct" answer in order to recover something like the parameters you used to generate the data?

(e) Try the previous part, but use one of the *global* optimization algorithms, like NLOPT_GN_DIRECT_L or NLOPT_GD_MLSL_LDS, or NLOPT_GN_CRS2_LM (see the NLopt manual, at `ab-initio.mit.edu/nlopt`, for what these algorithms are). You'll need to specify finite search-box constraints; use [0,0,0,0,0,0] for the lower bounds and [5,5,5,5,5,5] for the upper bounds. You'll also need to change the stopping conditions. Change stop.xtol_rel = 0 and and just vary the maximum number of function evaluations—set stop.maxeval = 1000 to start with (at most 1000 function evaluations). How big does the number of function evaluations need to be to recover (roughly) the parameters you used to generate the data? Is there a different global optimization algorithm in NLopt that requires significantly fewer evaluations for similar accuracy?

(f) Most global-optimization methods spend most of their time searching the parameter space and not much time "polishing" the local optima, so it is better to finish them off by running a local optimization at the end. If you do this for DIRECT_L, running LBFGS at the end to get the final local optimum more accurately, can you significantly reduce the maxeval from the global search while still recovering the original parameters accurately?