BFGS vs. Conjugate Gradient Method

What considerations should I be making when choosing between BFGS and conjugate gradient for optimization? The function I am trying to fit with these variables are exponential functions; however, the actual objective function involves integration, among other things, and is very costly if that helps at all.

optimization conjugate-gradient

edited Nov 15 '13 at 10:01
Jan
2,120 8 26

asked Dec 22 '11 at 16:30
Laurbert515
852 1 7 14

3 Well, BFGS is certainly more costly in terms of storage than CG. One requires the maintenance of an approximate Hessian, while the other only needs a few vectors from you. On the other hand, both require the computation of a gradient, but I am told that with BFGS, you can get away with using finite difference approximations instead of having to write a routine for the gradient (but the version using finite differences converges a bit slower than the one using actual gradients, of course). If you have an automatic differentiation facility, then your only worry is storage. – J. M. Dec 22 '11 at 17:47

have to fit only ~7 (definitely less than 10) variables means that the hessian approximation is only (at most) a 10x10 matrix correct? in which case, is one faster than the other? – Laurbert515 Dec 22 '11 at 18:24

I don't think there'd be that much of a difference in speed; if anything, I think the part of your computation that would probably take the most time is the quadratures you have to do for function evaluation. You really should do some experiments yourself, if the number of parameters is as small as you claim. – J. M. Dec 22 '11 at 18:30

3 Answers

J.M. is right about storage. BFGS requires an approximate Hessian, but you can initialize it with the identity matrix and then just calculate the rank-two updates to the approximate Hessian as you go, as long as you have gradient information available, preferably analytically rather than through finite differences. BFGS is a quasi-Newton method, and will converge in fewer steps than CG, and has a little less of a tendency to get "stuck" and require slight algorithmic tweaks in order to achieve significant descent for each iteration.

In contrast, CG requires matrix-vector products, which may be useful to you if you can calculate directional derivatives (again, analytically, or using finite differences). A finite difference calculation of a directional derivative will be much cheaper than a finite difference calculation of a Hessian, so if you choose to construct your algorithm using finite differences, just calculate the directional derivative directly. This observation, however, doesn't really apply to BFGS, which will calculate approximate Hessians using inner products of gradient information.

In terms of convergence rates, if n is the number of decision variables in your problem, then n CG iterations approximately equals one step of Newton's method. BFGS is a quasi-Newton method, but the same sort of observation should hold; you're likely to get convergence in fewer iterations with BFGS unless there are a couple CG directions in which there is a lot of descent, and then after a few CG iterations, you restart it. CG-like methods are cheaper if matrix-vector products are cheap and your problem is so large that storing the Hessian is difficult or impossible. BFGS involves some more vector-vector products to update its approximate Hessian, so each BFGS iteration will be more expensive, but you'll require fewer of them to reach a local minimum.

I would compare the two algorithms on a small test problem for your application if you know that storage won't be an issue. Without knowing the particular specifics of your problem, my guess is that BFGS will be faster, but you should really test the two algorithms to get a better idea of which will work better.

Finally, a word about automatic differentiation: having some experience with an in house automatic differentiation (AD) facility for Fortran (DAEPACK), I can tell you that AD tools are often finicky. They may not necessarily be able to differentiate the code that they generate themselves. There are two types of AD tools:

- source-to-source AD tools
- operator overloading AD tools

Source-to-source tools are essentially modified compilers that take source code you've written, parse it, and automatically generate new source code that computes the gradient of functions in your source code. Operator overloading AD tools require you to use the overloaded AD operators in your source code so that derivatives can be calculated, which would require additional effort on your part to calculate analytical derivatives with AD.

answered Dec 22 '11 at 18:36
Geoff Oxberry ♦
21.5k 8 29 96

The associated cost of BFGS may be brought more in line with CG if you use the limited memory variants rather than the full-storage BFGS. This computes the BFGS update for the last m updates efficiently by a series of rank-one updates without needing to store more than the last m solutions and gradients.

In my experience, BFGS with a lot of updates stores information too far away from the current solution to be really useful in approximating the non-lagged Jacobian, and you can actually lose convergence if you store too much. There are "memoryless" variants of BFGS that look a lot like nonlinear conjugate gradients (see the final update described for one of these) for just these reasons. Therefore, if you're willing to do L-BFGS rather than BFGS, the memory issues disappear and the methods are related. Anecdotal evidence points to restarting being a tricky issue, as it is sometimes unnecessary and sometimes very necessary.

Your choice between the two also depends heavily on the problems you are interested in. If you have the resources, you can try both for your problems and decide which works better. For example, I personally don't do optimization with these algorithms, but instead care about the solution of systems of nonlinear equations. For these I have found that NCG works better and is easier to perform nonlinear preconditioning on. BFGS is more robust.

Frankly, my favorite method for these types of things is N-GMRES. This is especially true if your gradient evaluation is very expensive, as in my experience it gives you the most bang for your buck by solving a small minimization problem on the last m iterates to construct a new, lower-residual solution.

edited Dec 23 '11 at 5:03
J. M.
1,935 14 28

answered Dec 23 '11 at 4:12
Peter Brune
1,255 5 17

I totally forgot about L-BFGS. +1 for that. – J. M. Dec 23 '11 at 4:51

In low dimensions, a well implemented BFGS method is generally both faster and more robust than CG, especially if the function is not very far from a quadratic.

Neither BFGS nor CG need any assumption about convexity; only the initial Hessian approximation (for BFGS) resp. the preconditioner (for CG) must be positive definite. But these can always be chosen to be the identity matrix, in low dimensions without much harm. See also http://scicomp.stackexchange.com/a/3213/1117

In the absence of a programmed gradient, it is a big waste of effort to use numerical gradients, especially when function values are expensive. Rather, one should use a derivative-free algorithm. See http://archimedes.cheme.cmu.edu/?q=dfocomp for a recent survey.

edited Sep 13 '12 at 16:30

answered May 13 '12 at 18:32
Arnold Neumaier
9,577 9 31

The link gives me a "404 Not Found", could you fix it? – Stiefel Sep 13 '12 at 15:49

@Stiefel: I fixed it. The new link points to a much improved version. – Arnold Neumaier Sep 13 '12 at 16:31