Optimization II: Numerical Algorithms and Newton's Method

Nathan Ratliff

Dec 12, 2014

Abstract

Newton's method take center stage in this document as a workhorse for generating numerical optimization algorithms for both unconstrained nonlinear problems and constrained variants. This preliminary (unfinished) document on that body of work summaries some of the main result, building toward Sequential Quadratic Programming for equality constrained nonlinear problems which can be viewed as an application of Newton's method to solving the problem's KKT conditions.

1 Logistic regression: An example of convex optimization in machine learning

1.1 Convexity

1.2 Convex functions

2 Algorithms for unconstrained optimization

2.1 Basic iterative algorithms

One very popular rapidly converging algorithm for nonlinear optimization is Newton's method. It's instructive and intuitive to think of Newton's method as approximating the objective at each iteration by a second-order Taylor expansion, solving that approximation to get the next point, and iterating that process until convergence. Mathematically, suppose our objective is $f : \mathbb{R}^n \to \mathbb{R}$ and our current iterate is $\boldsymbol{x}_t$. The second-order Taylor expansion around $\boldsymbol{x}_t$ is

$$f(\boldsymbol{x}_t + \delta\boldsymbol{x}) \approx f(\boldsymbol{x}_t) + \nabla f(\boldsymbol{x}_t)^T \delta\boldsymbol{x} + \frac{1}{2}\delta\boldsymbol{x}^T \nabla^2 f(\boldsymbol{x}_t)\delta\boldsymbol{x}.$$

Solving that approximation gives a step of the form

$$\delta\boldsymbol{x}_t = -\Big(\nabla^2 f(\boldsymbol{x}_t)\Big)^{-1}\nabla f(\boldsymbol{x}_t), \tag{1}$$

so that the new point becomes

$$\begin{aligned}\boldsymbol{x}_{t+1} &= \boldsymbol{x}_t + \delta\boldsymbol{x}_t \\ &= \boldsymbol{x}_t - \Big(\nabla^2 f(\boldsymbol{x}_t)\Big)^{-1}\nabla f(\boldsymbol{x}_t).\end{aligned}$$

This algorithm is very effective as long as f is sufficiently curved and positive definite. It's especially nice for strongly convex functions. Those conditions don't always hold, though, so there are a number of well-studied tricks for adapting over time how much we trust both the Hessian information and the region of applicability of the quadratic approximation, but we won't go into detail about those here. For in-depth discussions and analyses of those tricks, see Nocedal & Wright (2006). Here we just assume that the function f under consideration is "nice" enough to use the "pure" form of this Newton's method update.

2.2 Newton's method for nonlinear equations

The above intuition describing Newton's method as a sequential quadratic approximation optimizer is nice and visual, but it's less useful for extending the ideas to a form of Newton's method appropriate to constrained problems. Section 4 talks about how that intuition can be misleading when constraints are involved. It's best, instead, for generalization to think of Newton's method as not iterative solutions to quadratic approximations of the objective, but as an approach to simply solving n nonlinear equation in n variables applied to directly solving the problem's first-order optimality conditions.

Remember that the first-order optimality conditions for unconstrained nonlinear objectives state that any local optimizer must be at a point where the gradient is zero. So why not just directly try to find such a point

$$\nabla f(\boldsymbol{x}) = \mathbf{0}, \tag{2}$$

where that gradient is zero?

This nonlinear system given by Equation 2 is a set of n nonlinear equations in n inputs. We can't solve arbitrary differentiable nonlinear equations very easily, but we can solve linear ones using our techniques from linear algebra. So a good approach might be to linearize the system by taking a first-order Taylor approximation of the left hand side, solve that, and then iterate. That, indeed, is a good approach to iteratively solving nonlinear systems of equations in general, and in this case, it's precisely what Newton's method is doing. Indeed, it, too, is called **Newton's method for nonlinear equations**. The resulting first-order approximation at each iteration is

$$\nabla f(\boldsymbol{x}_t) + \nabla^2 f(\boldsymbol{x}_t)\delta\boldsymbol{x}_t = \mathbf{0},$$

which means that the update is again

$$\delta\boldsymbol{x}_t = -\Big(\nabla^2 f(\boldsymbol{x}_t)\Big)^{-1}\nabla f(\boldsymbol{x}_t),$$

which is precisely what we derived in Equation 1.

3 Nonlinear least squares, Gauss-Newton approximations, and ambient space walks

4 Newton's method for equality constrained nonlinear optimization

Newton's method, as presented in Section 2.2, is a generic method for iteratively solving nonlinear equations of the form $\boldsymbol{b}(\boldsymbol{x}) = \boldsymbol{0}$ for $\boldsymbol{b} : \mathbb{R}^n \to \mathbb{R}^n$. If we have a full set of n nonlinear equations in n dimensions, we can often iterate $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{J}_{\boldsymbol{b}}^{-1}\boldsymbol{b}(\boldsymbol{x}_t)$ until convergence to solve the problem. And we saw that when the set of nonlinear equations was the first-order optimality condition $\nabla \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ for an unconstrained nonlinear objective $f : \mathbb{R}^n \to \mathbb{R}$, this method solves the optimization problem (in most cases, as long as the objective is "nice" enough). Local optima must satisfy the first-order optimality conditions, so this application of Newton's method just directly tries to find a point that does.

But now what happens when we have constraints? For the purpose of illustration, we focus here only on equality constraints since the complementarity conditions of inequalities make them more cumbersome. But in practice, inequalities can often be handled by estimating which are *active* and which aren't, and treating the active constraints simply as *equality* constraints.

Suppose we're solving a problem of the form

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}, \end{aligned}$$

for $f : \mathbb{R}^n \to \mathbb{R}$ and $\boldsymbol{h} : \mathbb{R}^n \to \mathbb{R}^k$. Section 2.1 presented a view that Newton's method works by reducing the problem to something we can solve more easily using linear algebraic techniques, namely quadratic approximations, and solving those repeatedly rather than attacking the original complicated objective directly. That's nicely intuitive, but it's actually slightly misleading in this case. If we try to heuristically do the same thing here—simply create a simpler proxy that's somehow representative of the original problem and solve that—we might guess that the right thing to do is convert this problem into the one of simplest forms of equality constrained problem we know: quadratic programs. And how might we do that? The most natural and direct transformation would be to linearly approximate the constraints and quadratically approximate the objective. The result is the desired equality constrained quadratic program, and we know we can solve those analytically, so we're good, right? No. Unfortunately, there are a number of counterexamples where iterating such a process doesn't converge properly to the minimum. So this sort of naïve simplification heuristic breaks when we start adding constraints.

Instead of doing that, we instead take exactly the same brute force Newton's method for nonlinear equations approach summarized in Section 2.2. But now, the first-order optimality conditions aren't just that the gradient must be zero.

They state that the gradient must lie in the linear span of the rows of the constraint function's Jacobian, *and* that the original constraints, themselves, must hold. Those two equations are the KKT conditions restricted to just equality constraints. We can summarize them as

$$\boldsymbol{F}(\boldsymbol{x},\boldsymbol{\lambda}) = \left[\begin{array}{c} \nabla f(\boldsymbol{x}) + \boldsymbol{J}_{\boldsymbol{h}}^T\boldsymbol{\lambda} \\ \boldsymbol{h}(\boldsymbol{x}) \end{array}\right] = \boldsymbol{0}, \tag{3}$$

for some vector of Lagrange multipliers $\boldsymbol{\lambda} \in \mathbb{R}^k$, one for each constraint. The first constraint supplies n equations, and the second supplies k more, so even though we've added k extra variables, the system is fully constrained (as long as it's a "nice" problem in the sense that the Jacobian $\boldsymbol{J}_{\boldsymbol{h}}$ doesn't lose rank).

So we have a fully specified set of $n+k$ equations in $n+k$ variables. Why not now just apply Newton's method for nonlinear equations to these KKT conditions as we did before?

4.1 Using Newton's method to solve the KKT conditions

The generic Newton's method procedure for this problem is to simply solve linear approximations to $\boldsymbol{F}(\boldsymbol{x},\boldsymbol{\lambda})$ repeatedly until convergence. Specifically, if we're currently at $\boldsymbol{x}_t$ and $\boldsymbol{\lambda}_t$, we improve our solution by solving the problem

$$\begin{aligned} &\boldsymbol{F}(\boldsymbol{x}_t + \delta\boldsymbol{x}_t, \boldsymbol{\lambda}_t + \delta\boldsymbol{\lambda}_t) \\ &\qquad\approx \boldsymbol{F}(\boldsymbol{x}_t,\boldsymbol{\lambda}_t) + \left[\begin{array}{cc} \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} & \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{\lambda}} \end{array}\right] \left(\begin{array}{c} \delta\boldsymbol{x}_t \\ \delta\boldsymbol{\lambda}_t \end{array}\right) = \boldsymbol{0}, \end{aligned} \tag{4}$$

and our new solution approximation is

$$\begin{aligned} \boldsymbol{x}_{t+1} &= \boldsymbol{x}_t + \delta\boldsymbol{x}_t \\ \boldsymbol{\lambda}_{t+1} &= \boldsymbol{\lambda}_t + \delta\boldsymbol{\lambda}_t. \end{aligned}$$

So lets do that. All we need to do is plug in the definition of $\boldsymbol{F}(\boldsymbol{x},\boldsymbol{\lambda})$ to the generic form of the linearization above. That'll give us a linear equation whose solution gives the updates $\delta\boldsymbol{x}_t$ and $\delta\boldsymbol{\lambda}_t$. The only tricky part is perhaps the Jacobian computation for $\boldsymbol{F}(\boldsymbol{x},\boldsymbol{\lambda})$, but that just amounts to calculation. To perform the calculation, it's easiest to notice that $\boldsymbol{J}_{\boldsymbol{h}}^T\boldsymbol{\lambda} = \sum_{i=1}^k \lambda_i \nabla h_i(\boldsymbol{x})$, so $\frac{\partial}{\partial \boldsymbol{x}}\left[\boldsymbol{J}_{\boldsymbol{h}}^T\boldsymbol{\lambda}\right] = \sum_{i=1}^k \lambda_i \nabla^2 h_i(\boldsymbol{x})$ since $\frac{\partial}{\partial \boldsymbol{x}}\nabla h_i(\boldsymbol{x}) = \nabla^2 h_i(\boldsymbol{x})$. The two pieces of this Jacobian together are then

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{F}(\boldsymbol{x},\boldsymbol{\lambda}) &= \left[\begin{array}{c} \nabla^2 f(\boldsymbol{x}) + \sum_{i=1}^k \lambda_i \nabla^2 h_i(\boldsymbol{x}) \\ \boldsymbol{J}_{\boldsymbol{h}} \end{array}\right] \\ \text{and}\quad \frac{\partial}{\partial \boldsymbol{\lambda}}\boldsymbol{F}(\boldsymbol{x},\boldsymbol{\lambda}) &= \left[\begin{array}{c} \boldsymbol{J}_{\boldsymbol{h}}^T \\ \boldsymbol{0} \end{array}\right]. \end{aligned}$$

Combining them into a single matrix gives

$$\begin{aligned}\left[\begin{array}{cc}\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} & \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{\lambda}}\end{array}\right] &= \left[\begin{array}{cc}\nabla^2 f(\boldsymbol{x}) + \sum_{i=1}^{k} \lambda_i \nabla^2 h_i(\boldsymbol{x}) & \boldsymbol{J}_{\boldsymbol{h}}^T \\ \boldsymbol{J}_{\boldsymbol{h}} & \boldsymbol{0}\end{array}\right] \\ &= \left[\begin{array}{cc}\nabla^2_{\boldsymbol{x}\boldsymbol{x}}\mathcal{L} & \boldsymbol{J}_{\boldsymbol{h}}^T \\ \boldsymbol{J}_{\boldsymbol{h}} & \boldsymbol{0}\end{array}\right], \end{aligned} \tag{5}$$

were in that last line we've made the connection that the upper left block is just the Hessian of the Lagrangian for this problem taken with respect to $\boldsymbol{x}$:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{h}(\boldsymbol{x})$$

$$\text{with } \nabla^2_{\boldsymbol{x}\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \nabla^2 f(\boldsymbol{x}) + \sum_{i=1}^{k} \lambda_i \nabla^2 h_i(\boldsymbol{x}).$$

Note that below we often declutter notation by dropping the dependence of $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ on $\boldsymbol{x}$ and $\boldsymbol{\lambda}$ in its derivatives to denote them as $\nabla_{\boldsymbol{x}}\mathcal{L}$ and $\nabla^2_{\boldsymbol{x}\boldsymbol{x}}\mathcal{L}$ since the dependence is clear from the context.

Putting this all together and plugging these expressions back into the generic form of the linearization in Equation 4 gives

$$\underbrace{\left[\begin{array}{cc}\nabla^2_{\boldsymbol{x}\boldsymbol{x}}\mathcal{L} & \boldsymbol{J}_{\boldsymbol{h}}^T \\ \boldsymbol{J}_{\boldsymbol{h}} & \boldsymbol{0}\end{array}\right]}_{\boldsymbol{A}_t} \left(\begin{array}{c}\delta \boldsymbol{x}_t \\ \delta \boldsymbol{\lambda}_t\end{array}\right) = \underbrace{\left[\begin{array}{c}-\nabla f(\boldsymbol{x}_t) - \boldsymbol{J}_{\boldsymbol{h}}^T \boldsymbol{\lambda}_t \\ -\boldsymbol{h}(\boldsymbol{x}_t)\end{array}\right]}_{\boldsymbol{b}_t}. \tag{6}$$

Newton's method applied to the KKT conditions simply calculates $\boldsymbol{A}_t$ and $\boldsymbol{b}_t$ using the current iterates $\boldsymbol{x}_t$ and $\boldsymbol{\lambda}_t$, solves the above linear equation for $\delta \boldsymbol{x}_t$ and $\delta \boldsymbol{\lambda}_t$, and then computes the updated iterates as $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \delta \boldsymbol{x}_t$ and $\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \delta \boldsymbol{\lambda}_t$.

4.2 Sequential quadratic programming

Now that we've established a correctly convergent algorithm for solving such nonlinear equality constrained optimization problems, it's intuitively nice and valid to ask what's the equivalent iterative problem approximation interpretation for this problem analogous to the one we discussed for the unconstrained case in Section 2.1. We saw in the introduction to Section 2.2 that the naïve guess at what the approximation might be didn't work out, but now we have an explicit linear equation that we're solving at each iteration constructed from gradients and Hessians of the objective and constraints. Surely we should be able to reinterpret these equations as solving some sort of quadratic program.

Consider a generic equality constrained quadratic program of the form

$$\begin{aligned} \min_{\boldsymbol{u}} \ & \frac{1}{2}\boldsymbol{u}^T \boldsymbol{H} \boldsymbol{u} - \boldsymbol{v}^T \boldsymbol{u} + c \\ \text{s.t.} \ & \boldsymbol{C}\boldsymbol{u} = \boldsymbol{d}. \end{aligned} \tag{7}$$

The KKT conditions of this problem are already linear, and they take the form

$$\begin{aligned} \boldsymbol{H}\boldsymbol{u} - \boldsymbol{v} + \boldsymbol{C}^T\gamma &= \boldsymbol{0} \\ \boldsymbol{C}\boldsymbol{u} - \boldsymbol{d} &= \boldsymbol{0}, \end{aligned} \tag{8}$$

where $\boldsymbol{\gamma}$ is the vector of Lagrange multipliers. We can rewrite them as

$$\begin{bmatrix} \boldsymbol{H} & \boldsymbol{C}^T \\ \boldsymbol{C} & \boldsymbol{0} \end{bmatrix} \begin{pmatrix} \boldsymbol{u} \\ \gamma \end{pmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{d} \end{bmatrix}.$$

In this form, we can directly match the generic coefficients up to the components of Equation 6. Doing so gives correspondences

$$\begin{array}{rcl} \boldsymbol{u} & \leftarrow & \delta\boldsymbol{x}_t \\ \gamma & \leftarrow & \boldsymbol{\lambda} \\ \boldsymbol{H} & \leftarrow & \nabla^2_{\boldsymbol{xx}}\mathcal{L} \\ \boldsymbol{C} & \leftarrow & \boldsymbol{J_h} \\ \boldsymbol{v} & \leftarrow & -\nabla f(\boldsymbol{x}_t) - \boldsymbol{J}_{\boldsymbol{h}}^T\boldsymbol{\lambda}_t \\ \boldsymbol{d} & \leftarrow & -\boldsymbol{h}(\boldsymbol{x}_t). \end{array}$$

Substituting these values into the generic quadratic program of Equation 7 tells us that that the problem we're looking for takes the form

$$\begin{aligned} \min_{\delta\boldsymbol{x}} \quad & \big(f(\boldsymbol{x}_t) + \boldsymbol{\lambda}_t^T\boldsymbol{h}(\boldsymbol{x}_t)\big) + \big(\nabla f(\boldsymbol{x}_t) + \boldsymbol{J}_{\boldsymbol{h}}^T\boldsymbol{\lambda}_t\big)^T\delta\boldsymbol{x} + \frac{1}{2}\delta\boldsymbol{x}^T\,\nabla^2_{\boldsymbol{xx}}\mathcal{L}\,\delta\boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{x}_t) + \boldsymbol{J_h}\delta\boldsymbol{x} = \boldsymbol{0}, \end{aligned} \tag{9}$$

where we've added the constant value $\boldsymbol{\lambda}_t^T\boldsymbol{h}(\boldsymbol{x}_t)$ to the objective to make this constant term the original problem's Lagrangian evaluated at $\boldsymbol{x}_t$ and $\boldsymbol{\lambda}_t$

$$\mathcal{L}(\boldsymbol{x}_t, \boldsymbol{\lambda}_t) = f(\boldsymbol{x}_t) + \boldsymbol{\lambda}_t^T\boldsymbol{h}(\boldsymbol{x}_t).$$

Notice also that the linear term's coefficients in the objective are just the gradient in $\boldsymbol{x}$ of the Lagrangian

$$\nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \nabla f(\boldsymbol{x}_t) + \boldsymbol{J}_{\boldsymbol{h}}^T\boldsymbol{\lambda}.$$

All of these observations tell us that the objective is just the second-order Taylor expansion of the Lagrangian in terms of the variable $\boldsymbol{x}$. Putting all of this together, we get that a quadratic program whose KKT conditions are the linearized KKT conditions of the original problem is

$$\begin{aligned} \min_{\delta\boldsymbol{x}} \quad & \mathcal{L}(\boldsymbol{x}_t, \boldsymbol{\lambda}_t) + \nabla_{\boldsymbol{x}}\mathcal{L}^T\delta\boldsymbol{x} + \frac{1}{2}\delta\boldsymbol{x}^T\,\nabla^2_{\boldsymbol{xx}}\mathcal{L}\,\delta\boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{x}_t) + \boldsymbol{J_h}\delta\boldsymbol{x} = \boldsymbol{0}, \end{aligned} \tag{10}$$

In other words, the right thing to do isn't to quadratically approximate the objective and linearly approximate the constraints. We need to form a quadratic approximation of the entire Lagrangian. The second-order information encoded in the constraint function Hessians is just as important to the problem as the second-order information encoded in the objective's Hessian. Solving this quadratic program gives $\delta \boldsymbol{x}_t$ and $\delta \boldsymbol{\lambda}_t$. The former is the quadratic program's solution variable, and the latter is its optimal Lagrange multipliers; together they form the updates to both the original problem's optimization variable $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \delta \boldsymbol{x}_t$ and its corresponding the Lagrange multipliers $\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \delta \boldsymbol{\lambda}_t$.

It's sometimes useful to consider the version of the problem that combines $\delta \boldsymbol{\lambda}_t$ and $\boldsymbol{\lambda}_t$ itself into a single variable $\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \delta \boldsymbol{\lambda}_t$ to form the following reformulation of the system in Equation 6:

$$\begin{bmatrix} \nabla^2_{\boldsymbol{xx}} \mathcal{L} & \boldsymbol{J}_{\boldsymbol{h}}^T \\ \boldsymbol{J}_{\boldsymbol{h}} & \boldsymbol{0} \end{bmatrix} \begin{pmatrix} \delta \boldsymbol{x}_t \\ \boldsymbol{\lambda}_{t+1} \end{pmatrix} = \begin{bmatrix} -\nabla f(\boldsymbol{x}_t) \\ -\boldsymbol{h}(\boldsymbol{x}_t) \end{bmatrix}. \tag{11}$$

These equations are equivalent to those given by Equation 6 since replacing $\boldsymbol{\lambda}_{t+1}$ with its definition $\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \delta \boldsymbol{\lambda}_t$ in this new set gives us back exactly those original equations. Solving this version, though, directly finds an estimate of $\boldsymbol{\lambda}_{t+1}$, the new full set of Lagrange multipliers, rather than just the update $\delta \boldsymbol{\lambda}_t$ to the existing multipliers.

Under these reformulated equations, the corresponding quadratic programming approximation that has these linearized equations as its KKT conditions is

$$\begin{aligned} \min_{\delta \boldsymbol{x}} \quad & f(\boldsymbol{x}_t) + \nabla f(\boldsymbol{x}_t)^T \delta \boldsymbol{x} + \frac{1}{2} \delta \boldsymbol{x}^T \, \nabla^2_{\boldsymbol{xx}} \mathcal{L} \, \delta \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{x}_t) + \boldsymbol{J}_{\boldsymbol{h}} \delta \boldsymbol{x} = \boldsymbol{0}. \end{aligned} \tag{12}$$

This version of the quadratic program solves for $\delta \boldsymbol{x}_t$, which is the *update* to $\boldsymbol{x}_t$, and the full next Lagrange multiplier $\boldsymbol{\lambda}_{t+1}$ because of the reformulation above. The objective shows that we really only need to account here for the *second-order* information of the constraint in the objective, since the first-order information is already encoded in the linearized constraint.

This method is called Sequential Quadratic Programming (SQP). Most commonly the iterative quadratic programming approximation used in practice is the one given by Equation 12. We know from the above discussion that deep down it's just Newton's method on the KKT conditions of the original equality constrained nonlinear programming problem, but its interpretation as iterative solutions to quadratic programming approximations is very alluring and leads to some effective practical generalizations that extend its applicability beyond this "pure" Newton's method mode. For more information on SQP techniques, including methods for integrating inequality constraints and expanding the radius of convergence, see Nocedal & Wright (2006).

References

Nocedal, Jorge and Wright, Stephen. *Numerical Optimization.* Springer, 2006.