## Week 7: Monday, Mar 12

# Newton and Company

Suppose $f : \mathbb{R}^n \to \mathbb{R}^n$ is twice differentiable. Then

$$f(x + \delta x) = f(x) + f'(x)\delta x + O(\|\delta x\|^2),$$

where $f'(x)$ denotes the Jacobian matrix at $x$. The idea of Newton iteration in this multi-dimensional setting is the same as the idea in one dimension: in order to get $x_{k+1}$, set the linear approximation to $f$ about $x_k$ to zero. That is, we set

$$f(x_k) + f'(x_k)(x_{k+1} - x_k) = 0,$$

which we can rearrange to

$$x_{k+1} = x_k - f'(x_k)^{-1} f(x_k).$$

The basic form of Newton iteration remains the same in the multi-dimensional case, and so do the basic convergence properties. As usual, we obtain a recurrence for the error by subtracting the fixed point equation from the fixed point equation:

$$\begin{aligned} f(x_k) + f'(x_k)(x_{k+1} - x_k) &= 0 \\ f(x_k) + f'(x_k)(x_* - x_k) &= O(\|x_* - x_k\|^2) \\ \hline f'(x_k)e_{k+1} &= O(\|e_k\|^2) \end{aligned}$$

As long as $f'(x_*)$ is nonsingular (i.e. $x_*$ is a *regular* root) and the second derivatives of $f$ are continuous near $x_*$, this iteration is quadratically convergent to regular roots (where $f'$ is nonsingular) from close enough starting points. But the iteration may well diverge if the starting point is not good enough. What changes in the multi-dimensional case is the cost of a Newton step. If the dimension $n$ is large, the cost of forming and factoring the Jacobian matrix may start to dominate the other costs in the iteration. For this reason, one frequently uses modified Newton iterations in which solutions with $f'(x_k)^{-1}$ are approximated in some way.

There are several ways to approximate $f'(x_k)^{-1}$, and these provide different tradeoffs between the reduction in error per step and the cost per step. The simplest flavor of modified Newton iteration might be

$$x_{k+1} = x_k - \hat{J}^{-1} f(x_k),$$

where $\hat{J}$ is an approximation to the Jacobian (e.g. computed and factored at $x_0$, and then frozen for successive iterations). When does this converge? Remember our basic analysis method for fixed point iterations: take the iteration equation and subtract the fixed point equation in order to get an iteration for the error. Here, that gives us

$$e_{k+1} = \left(I - \hat{J}^{-1} f'(x_*)\right) e_k + O(\|e_k\|^2),$$

and taking norms gives

$$\|e_{k+1}\| \leq \left\|I - \hat{J}^{-1} f'(x_*)\right\| \|e_k\| + O(\|e_k\|^2),$$

so convergence is assured (for $x_0$ close to $x_*$) when $\|I - \hat{J}^{-1} f'(x)\| < 1$.

# Dealing with Newton

Newton iteration and closely-related variants are a workhorse in nonlinear equation solving. Unfortunately, as we have seen, Newton's method is only *locally* convergent. A good part of the art of nonlinear equation solving is in dealing with this local convergence property. In one dimension, we can combine Newton's method (or secant method, or other iterations) with bisection in order to get something that is simultaneously robust and efficient; Charlie talked about this last Wednesday. But bisection is a one-dimensional construction. In higher dimensions, what can we do?

As it turns out, there are several possible strategies:

1. *Get a good guess*: If you have some method of getting a good initial guess, Newton iteration is terrific. Getting a good guess is application-specific.

2. *Modify the problem*: There are usually many equivalent ways to write an equation $f(x) = 0$. Some of those ways of writing things may lead to better convergence. For example, if $f(x)$ has a zero close to the origin and approaches some constant value far away from the origin, we might want to look at an equation like $f(x)(\|x\|^2 + 1) = 0$. If $f(x)$ has a pole that causes problems, we might want to multiply through by a function that removes that pole. In general, it pays to have a good understanding of the properties of the functions you are solving, and to

try to minimize the effects of properties that confuse Newton iteration. Alas, this is also application-specific.

3. *Choose a specialized iteration*: Newton iteration is our workhorse, but it isn't the only horse around. Other iterations may have better convergence properties, or they may be cheap enough that you are willing to let them run for many more iterations than you would want to take with Newton. This is application-specific; but for lots of applications, you can find something reasonable in a textbook or paper.

4. *Use a line search*: What goes wrong with Newton iteration? The Newton direction should always take us in a direction that reduces $\|f(x)\|$, but the problem is that we might overshoot. We can fix this problem by taking steps of the form

$$x_{k+1} = x_k - \alpha_k f'(x_k)^{-1} f(x_k),$$

where $\alpha_k$ is chosen so that $\|f(x_{k+1})\| < \|f(x_k)\|$. Refinements of this strategy lead to iterations that converge to *some* root from almost everywhere, but even the basic strategy can work rather well. Ideally, $\alpha_k \to 1$ eventually, so that we can get the quadratic convergence of Newton once we've gotten sufficiently close to a root.

5. *Use a trust region*[1]: The reason that Newton can overshoot the mark is because we keep using a linear approximation to $f$ about $x_k$ far beyond where the linear approximation is accurate. In a *trust region* method, we define a sphere of radius $\rho$ around $x_k$ where we think linear approximation is reasonable. If the Newton step falls inside the sphere, we take it; otherwise, we find a point on the surface of the sphere to minimize $\|f(x_k) + f'(x_k)(x_{k+1} - x_k)\|^2$.

6. *Use a continuation strategy*: Sometimes there is a natural way of gradually transitioning from an easy problem to a hard problem, and we can use this in a solver strategy. Suppose $f(x; s)$ is a family of functions parameterized by $s$, where solving $f(x; 1) = 0$ is hard and solving $f(x; 0) = 0$ is easy. Then one approach to solving $f(x; 1) = 0$ is:

[1] I will not ask you about trust regions on any homework or exam! But it is a sufficiently widely-used technique that you might want to at least recognize the term.

```
xguess = 0;  % Initial guess for the easy problem
for s = 0:ds:1
  % Solve f(x; s) = 0 using the solution to f(x; s-h) = 0 as
  % an initial guess
  xguess = basic_solver(f, s, xguess);
end
x = xguess;
```

There are many, many variants on this theme.