Week 6: Monday, Mar 7

Problem du jour

Show that for any initial guess $x_0 > 0$, Newton iteration on $f(x) = x^2 - a$ produces a decreasing sequence $x_1 \geq x_2 \geq \ldots \geq x_n \geq \sqrt{a}$. What is the rate of convergence if $a = 0$?

Answer: First, note that for $x_k > \sqrt{a}$, we have

$$\frac{f(x_k)}{f'(x_k)} > 0.$$

Therefore, if $x_k > \sqrt{a}$, then $x_{k+1} < x_k$.

Now, recall the error recurrence

$$e_{k+1} = \frac{1}{2}\frac{f''(\xi_k)}{f'(x_k)}e_k^2.$$

In the case of $f(x) = x^2 - a$, this boils down to

$$e_{k+1} = \frac{e_k^2}{2x_k}$$

Note that if $x_k > 0$, then $e_{k+1} > 0$, i.e. $x_{k+1} > x_*$. So the iterates (except possibly x_0) are always greater than $\sqrt{a}$.

There is a theorem from analysis that guarantees that any bounded, monotone sequence converges to some limit, so we know the Newton iteration converges. As for the *rate* of convergence, in the case $a = 0$, we have the iteration:

$$x_{k+1} = x_k/2.$$

This converges to zero, but the convergence is *linear*.

Logistics

1. HW 4 was due this morning.
2. Project 2 (ray casting) will be due Wednesday, March 16.
3. There will be no homework or projects over Spring Break.

Ray tracing

The next project will involve ray tracing an *implicit surface* defined by an equation $f(x, y, z) = 0$. We will assume that f is a nice, smooth function, and that there is a MATLAB function available to compute function values and gradients. Your job is to find where rays originating from the viewer first intersect this surface. I have already provided some framework code to do the rendering once you find where the surface is.

Ivo will talk about the project in section, but I want to emphasize two points now:

1. Part of the project involves understanding the performance of your code. I can render a sphere at 640-by-480 resolution in about 1.5 seconds; while I don't necessarily expect your code to be as fast, if your code takes ten minutes to do the same thing, it had better be producing *really* nice pictures. You should use MATLAB's profiler to help you understand where there are bottlenecks in your code that you might want to improve.

2. In addition to the code, I want a good write-up that describes what testing you have done, what shortcomings and failure modes your code may have, and where there are performance bottlenecks. The prompt has a list of items that I would like in your report. The write-up will be worth a substantial fraction of the project grade.

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 α_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*

[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.

method, we define a sphere of radius ρ 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:

```
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.

Unconstrained optimization: a refresher

Suppose $f : \mathbb{R} \to \mathbb{R}$ has two continuous derivatives. From a first calculus class, we know that any local minimizer or maximizer must be a *stationary point*, i.e.

$$f'(x_*) = 0.$$

This is sometimes called the *first derivative test*. There is also a *second derivative test* that we can use to distinguish whether a stationary point x_* is a minimum, maximum, or something else:

1. If $f''(x_*) > 0$, then x_* is a minimum.
2. If $f''(x_*) < 0$, then x_* is a maximum.
3. If $f''(x_*) = 0$, then we have no information.

We call x_* a *strong local minimizer* if $f'(x_*) = 0$ and $f''(x_*) > 0$.

The first and second derivative tests capture the idea that a local minimum point is a point where there are no "downhill" directions. If $f'(x)$ is

nonzero, then $f(x+\delta) \approx f(x) + f'(x)\delta < 0$ if δ is a small value whose sign is opposite the sign of $f'(x)$, so the first derivative test is necessary. Similarly, if x_* is a strong local minimizer, then

$$f(x_* + \delta) \approx f(x_*) + f''(x_*)\delta^2 > f(x_*)$$

for any sufficiently small δ.

Usually, we are interested in optimizing functions of more than one variable, and so we would like appropriate extensions of the first and second derivative tests. This is something you probably saw in a multivariate calculus class; but you may have forgotten it, or learned it with a different notation than I use, so let me briefly review here. We again want to find conditions that correspond to "no downhill direction"; that means we need to reason about the local behavior of f in different directions.

For a multi-variable function $f : \mathbb{R}^n \to \mathbb{R}$, recall that the directional derivative in some direction[2] u is

$$\frac{\partial f}{\partial u}(x) = \left.\frac{d}{ds}\right|_{s=0} f(x+su) = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(x) u_i,$$

where the second equality follows by the multivariable version of the chain rule. We can rewrite the summation in terms of vectors and dot products as

$$\sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(x) u_i = f'(x)u = \nabla f(x)^T u$$

where the derivative $f'(x)$ is a row vector of partial derivatives and the gradient $\nabla f(x)$ is a column vector of partials:

$$\begin{aligned} f' &= \begin{bmatrix} \partial f/\partial x_1 & \partial f/\partial x_2 & \dots & \partial f/\partial x_n \end{bmatrix} \\ \nabla f &= (f')^T. \end{aligned}$$

If $\nabla f(x) \neq 0$, that means that for $u = -\nabla f(x)$, the directional derivative is $\partial f/\partial u = -\|\nabla f(x)\|^2 < 0$, and so u is a downhill direction. In fact, the direction $-\nabla f(x)$ is the direction of *steepest descent*. Therefore, if x_* is a minimum, we require $\nabla f(x_*) = 0$ (i.e. x_* is a stationary point). This is the multivariable first derivative test.

[2] In some sources, you will see a convention that the direction u is always normalized to unit length. We will not have this requirement.

In order to generalize the second derivative test, we need to reason about second derivatives. We begin with directional second derivatives:

$$\frac{\partial^2 f}{\partial u^2}(x) = \left.\frac{d^2}{ds^2}\right|_{s=0} f(x+su) = \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial^2 f}{\partial x_i \partial x_j}(x) u_i u_j,$$

where the last equality comes by applying the multivariable chain rule to

$$\left.\frac{d^2}{ds^2}\right|_{s=0} f(x+su) = \left.\frac{d}{ds}\right|_{s=0} \left(\sum_{j=1}^{n} \frac{\partial f}{\partial x_j}(x+su) u_j \right).$$

We can write the directional second derivative in terms of summations over indices, but there is also a convenient matrix formulation:

$$\frac{\partial^2 f}{\partial u^2}(x) = u^T H_f(x) u$$

where $H_f(x)$ is the *Hessian matrix*:

$$H_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{bmatrix}.$$

If x_* is a stationary point and f has enough derivatives, then for every direction u we have

$$f(x_* + su) = f(x_*) + \frac{1}{2} s^2 u^T H_f(x_*) u + O(s^3),$$

where the first-order term drops out because $f'(x_*) = 0$. A sufficient condition for minimality is that we satisfy the second derivative test for any direction, i.e.

$$\forall v \neq 0, \quad v^T H_f(x_*) v > 0.$$

If the matrix $H_f(x_*)$ satisfies this condition, it is called *positive definite*.