1 Gradient Descent in Action: Solving Linear Systems

Suppose we have a linear system of equations of the form

$$\mathbf{Ax} = \mathbf{b} \tag{1}$$

where $\mathbf{A}$, for simplicity, is an invertible and positive semidefinite n by n matrix and $\mathbf{x}$ and $\mathbf{b}$ are length n vectors. A and b are given to us and we wish to find some setting of x which solves this equation. We could just invoke our knowledge of linear algebra and invert A to solve for x, but inverting a matrix takes $O(n^3)$ time with standard methods[1]. This could be prohibitively expensive if n is large. Fortunately, we can use gradient descent to solve this!

It may not be immediately apparent how we can use gradient descent to approach this, though. Overall, our goal is to make the function $g(\mathbf{x}) = \mathbf{Ax} - \mathbf{b}$ equal to the all-zeros vector. Note that g is a function from $\mathbb{R}^n$ to $\mathbb{R}^n$. However, gradient descent works only for functions of the form $\mathbb{R}^n \to \mathbb{R}$. Instead of g, we consider the following function $f : \mathbb{R}^n \to \mathbb{R}$:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{Ax} - \mathbf{b}^\top \mathbf{x} \, . \tag{2}$$

This formulation has the following nice property. If we take the gradient of f, we obtain

$$\nabla f(\mathbf{x}) = \mathbf{Ax} - \mathbf{b} \tag{3}$$

Recall that in gradient descent our gradient converges to 0. If we find where the gradient is 0, then we solve our original problem, so we can just use gradient descent on the function in equation 2! The takeaway lesson here is if you have a minimization problem on f(x), you'll want to run gradient descent on it. If you're trying to find a zero of some function, run gradient descent on the integral of it!

2 Newton's Method Introduction and Motivation

We now proceed to a different iterative minimization technique called Newton's method. Newton's method essentially attempts to find a zero of a function. In a single dimension, it takes a linear approximation of a function at a certain point and finds where this approximation intercepts the x-axis as depicted in Figure 1. In particular, if we start with a guess x^0 for the location of a zero, then we iteratively apply the update

$$x^k = x^{k-1} - \frac{f(x^{k-1})}{f'(x^{k-1})} \tag{4}$$

We can now solve many numerical problems by rephrasing them as a zero-finding scheme. For instance, if we want to find the value of the cube root of a, we'd want to find a zero of the function

$$f(x) = x^3 - a \tag{5}$$

We will proceed with a common application of this method to square roots.

[1]There are more advanced algorithms for computing a matrix inverse, but many of them are unfortunately not practical or work only for special matrices.

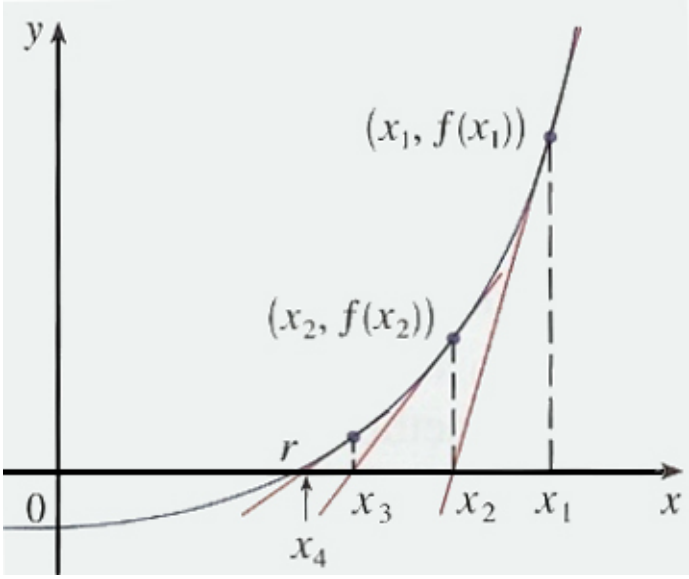

Figure 1: An example of a few iterations of Newton's method. Figure adapted from http://tutorial.math.lamar.edu/Classes/CalcI/NewtonsMethod.aspx

3 Computing Square Roots

For computing the square root of a, we start with a guess x_0 for $\sqrt{a}$. Then we repeatedly apply the formula:

$$x_{i+1} := x_i - \frac{x_i^2 - a}{2x_i} = x_i - \tfrac{1}{2}x_i + \tfrac{1}{2}\frac{a}{x_i} = \tfrac{1}{2}\left(x_i + \frac{a}{x_i}\right).$$

This repeated procedure is known as *Newton's Method* (for square roots; in fact, Newton's method is a much more general procedure).

Each iteration computes the next estimate x_{i+1} from x_i using one addition, one easy division by 2 (a shift), and one division by x_i. Note that a shift can be implemented in time $\Theta(d)$ by simply "moving over" each digit by the shift amount.

3.1 Termination

So far we do not have an actual algorithm because we haven't discussed termination. In general, there are two ways to do this:

1. Perform a fixed number of iterations and then return the final estimate.

2. Perform iterations until the estimates are changing very little, e.g., only beyond the d-th digit.

As with other search algorithms we have seen, one approach (1) makes it easy to check termination but not so easy to check safety, while the other approach (2) makes it easier to check safety but hard to check termination.

For method 1, we need to deduce the number of iterations we need to perform in order to have the desired precision of d digits (safety).

For method 2, both safety and termination require us to know that this method won't get stuck somewhere.

We'll now show how much each iteration of the square-root algorithm converges onto the correct answer, which will tell us how many iterations we need to perform if we use method (1) and will prove that the algorithm will terminate if we use method (2).

3.2 Correctness/Error Analysis

This all sounds great in principle, but how do we know that Newton's method really is converging to the answer?

Suppose that our current estimate x_i is off from the correct answer $\sqrt{a}$ by a factor of $1+\epsilon_i$ for some ϵ_i. (This ϵ_i is the "relative error" of our current estimate.) We can see how much error there will be in the next estimate simply from the definition:

$$\begin{aligned}
x_{i+1} &= \frac{1}{2}\left(\sqrt{a}(1+\epsilon_i) + \frac{a}{\sqrt{a}(1+\epsilon_i)}\right) \\
&= \sqrt{a}\,\frac{1}{2}\left((1+\epsilon_i) + \frac{1}{1+\epsilon_i}\right) \\
&= \sqrt{a}\,\frac{1}{2}\left(\frac{(1+\epsilon_i)^2}{1+\epsilon_i} + \frac{1}{1+\epsilon_i}\right) \\
&= \sqrt{a}\,\frac{1}{2}\frac{(1+\epsilon_i)^2+1}{1+\epsilon_i} \\
&= \sqrt{a}\,\frac{1}{2}\frac{2+2\epsilon_i+\epsilon_i^2}{1+\epsilon_i} \\
&= \sqrt{a}\,\left(\frac{1}{2}\frac{2(1+\epsilon_i)}{1+\epsilon_i} + \frac{1}{2}\frac{\epsilon_i^2}{1+\epsilon_i}\right) \\
&= \sqrt{a}\,\left(1 + \frac{\epsilon_i^2}{2(1+\epsilon_i)}\right)
\end{aligned}$$

In other words, if estimate x_i had relative error ϵ_i, then x_{i+1} has relative error $\frac{1}{2}\epsilon_i^2/(1+\epsilon_i)$.

The first question we might ask is: are we actually getting closer? In other words, do we have $\epsilon_{i+1} < \epsilon_i$? Actually, let's be a bit bolder and test whether $\epsilon_{i+1} < \frac{1}{2}\epsilon_i$. (This is the improvement we would get from binary search.)

$$\begin{aligned}
\epsilon_{i+1} &< \frac{1}{2}\epsilon_i \\
\frac{\epsilon_i^2}{2(1+\epsilon_i)} &< \frac{1}{2}\epsilon_i \\
\epsilon_i^2 &< \epsilon_i(1+\epsilon_i) \\
0 &< \epsilon_i
\end{aligned}$$

These equations are all equivalent for $\epsilon_i > 0$, and this final equation is then also true, so we deduce that we must always have $\epsilon_{i+1} < \frac{1}{2}\epsilon_i$. The case $\epsilon_i < 0$ is a little more tricky, but we get the same

result.[2] In other words, we can see that Newton's method for computing square roots is *always* converging at least as fast as binary search, for any initial guess x_0.

Linear convergence (like binary search) is nice, but we can see directly from the formula that it converges much faster than that. In particular, as long as we have $\epsilon_i > 0$, we have $1 + \epsilon_i > 1$, which means

$$\epsilon_{i+1} = \frac{\epsilon_i^2}{2(1+\epsilon_i)} < \tfrac{1}{2}\epsilon_i^2.$$

This is *quadratic* convergence.

In particular, suppose that our current estimate x_i is accurate to d digits in base 2. This means that $|x_i - \sqrt{a}| < \frac{1}{2^d}$, or in terms of our error variables, $\epsilon_i < \frac{1}{2^d}$. Plugging, this into the above relation, we get

$$\epsilon_{i+1} < \frac{1}{2}\left(\frac{1}{2^d}\right)^2 = \frac{1}{2^{2d+1}}.$$

Hence, each iteration takes us from d digits of precision to $2d + 1$ digits, which is (more than) doubling the precision on each iteration.

What does this mean in concrete terms? As long as our initial guess has 1 digit of precision (i.e, $\epsilon_i < \frac{1}{2}$), we will see a doubling of the number of digits of precision on each iteration. Hence, we can achieve a result with d digits of precision in $\lg d$ iterations. (In fact, it is sufficient to have $\epsilon_i < 1$ since we will gain one digit of precision on the first iteration.)

If our initial guess has much larger error, $\epsilon_i \gg 1$, then this analysis doesn't tell us anything useful about the rate of convergence. All we know in this case is that we do at least as well as binary search, which has linear convergence.

In fact, this analysis is not pessimistic. If you start with a very poor guess, then Newton's method really does behave like binary search (i.e., converging slowly) until you get close to the answer. At that point, you get a doubling of precision at each step.

[2] Assuming our initial guess was positive, $x_0 > 0$, we will have $-1 < \epsilon_i < 0$. It is easy to check that we then have $\epsilon_{i+1} > 0$. So for ϵ_{i+2}, we get the same analysis as above.