1 Solving systems of equations

Given a function $f : \mathbb{R} \rightarrow \mathbb{R}$, we find a point $x$ satisfying $f(x) = 0$ by an iterative procedure that approximates $f$ linearly: from a point $x_k$, we approximate

$$f(x) \approx f(x_k) + f'(x_k)(x - x_k)$$

and find $x_{k+1}$ by setting this approximation equal to 0.

When we have a function $g : \mathbb{R}^n \rightarrow \mathbb{R}$, we can try to find a point $x \in \mathbb{R}^n$ satisfying $g(x) = 0$ by the same procedure, but doing this is slightly awkward. We can approximate $g(x)$ at a point $x^{(k)}$ by

$$g(x) \approx g(x^{(k)}) + \nabla g(x^{(k)}) \cdot (x - x^{(k)})$$

but if we set this approximation equal to 0, we will find many solutions. It’s not clear which one of them is the best one to pick for $x^{(k+1)}$.

Instead, the natural generalization of Newton’s method to $n$ dimensions is for solving a system of equations:

$$\begin{align*}
  g_1(x) &= 0, \\
  g_2(x) &= 0, \\
  \vdots \\
  g_n(x) &= 0
\end{align*} \iff g(x) = 0$$

where $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a function with components $g_1, g_2, \ldots, g_n$.

In this case, a linear approximation of $g(x)$ at $x^{(k)}$ is given by combining the linear approximations of $g_1, g_2, \ldots, g_n$:

$$g(x) \approx \begin{bmatrix} g_1(x) \\ g_2(x) \\ \vdots \\ g_n(x) \end{bmatrix} \approx \begin{bmatrix} g_1(x^{(k)}) \\ g_2(x^{(k)}) \\ \vdots \\ g_n(x^{(k)}) \end{bmatrix} + \begin{bmatrix} \nabla g_1(x^{(k)})^T \\ \nabla g_2(x^{(k)})^T \\ \vdots \\ \nabla g_n(x^{(k)})^T \end{bmatrix} (x - x^{(k)}).$$

This has the form $g(x) \approx b + Ax$, where $b$ is a vector of the values of $g$ at $x^{(k)}$ and $A$ is a matrix of the derivatives of $g$ at $x^{(k)}$. The matrix, which we wrote above in terms of its rows (its $i^{th}$ row...
is the gradient of \( g_i \) at \( x^{(k)} \) can also be written as

\[
\begin{bmatrix}
\nabla g_1(x^{(k)})^T \\
\nabla g_2(x^{(k)})^T \\
\vdots \\
\nabla g_n(x^{(k)})^T
\end{bmatrix}
= 
\begin{bmatrix}
\frac{\partial g_1}{\partial x_1}(x^{(k)}) & \frac{\partial g_1}{\partial x_2}(x^{(k)}) & \cdots & \frac{\partial g_1}{\partial x_n}(x^{(k)}) \\
\frac{\partial g_2}{\partial x_1}(x^{(k)}) & \frac{\partial g_2}{\partial x_2}(x^{(k)}) & \cdots & \frac{\partial g_2}{\partial x_n}(x^{(k)}) \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial g_n}{\partial x_1}(x^{(k)}) & \frac{\partial g_n}{\partial x_2}(x^{(k)}) & \cdots & \frac{\partial g_n}{\partial x_n}(x^{(k)})
\end{bmatrix}.
\]

The \((i, j)\) entry of this matrix is the derivative of \( g_i \) with respect to \( x_j \). We write \( \nabla g(x^{(k)}) \) for this matrix, and it is called the Jacobian matrix of \( g \) at \( x^{(k)} \).

Altogether, the \( n \)-dimensional Newton’s method for solving \( g(x) = 0 \) starting with a guess \( x^{(0)} \) is to compute a sequence \( x^{(1)}, x^{(2)}, \ldots \) by letting \( x^{(k+1)} \) be the solution to

\[
g(x^{(k)}) + \nabla g(x^{(k)})(x^{(k+1)} - x^{(k)}) = 0.
\]

In theory, we can solve for \( x^{(k+1)} \) to get the nice-looking formula

\[
x^{(k+1)} = x^{(k)} - \nabla g(x^{(k)})^{-1}g(x^{(k)})
\]

which resembles the one-dimensional Newton’s method. But when \( n \) is large, it’s usually not efficient to solve the system of equations by computing the inverse.

It is not really required to have the number of variables be the same as the number of equations, it’s just the nicest case. When we have fewer equations than variables, we’ll end up making arbitrary choices at each step, because we’re likely to have infinitely many solutions for \( x^{(k+1)} \). (It might make sense to use least-squares optimization to pick \( x^{(k+1)} \) as close to \( x^{(k)} \) as possible, but that’s just me making things up.)

### 1.1 Example

For example, if we have \( g(x, y) = (x^2 + y^2 - 4, xy - 1) \) then

\[
\nabla g(x, y) = \begin{bmatrix} 2x & 2y \\ y & x \end{bmatrix}.
\]

If we want to iterate from the point \((1, 0)\), then we have

\[
g(1, 0) = \begin{bmatrix} -3 \\ -1 \end{bmatrix}, \quad \nabla g(1, 0) = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}
\]

and to find the next point \((x_1, y_1)\) we solve the system of equations

\[
\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 - 1 \\ y_1 - 0 \end{bmatrix} = \begin{bmatrix} -3 \\ -1 \end{bmatrix},
\]

or \(2(x_1 - 1) = 3\) and \(y_1 = 1\), getting the point \((\frac{5}{2}, 1)\).
2 Newton’s method for minimization

To build intuition, let’s return to the one-dimensional case. We have a function $f : \mathbb{R} \rightarrow \mathbb{R}$. But now, we want to minimize (or maximize) $f$. There’s two equivalent descriptions of our approach.

First, we can try find a critical point of $f$: a place where $f’(x) = 0$. This is something that ordinary Newton’s method can do. Starting at some guess $x_0$, we follow the iteration

$$x_{k+1} = x_k - \frac{f’(x_k)}{f''(x_k)}$$

which is Newton’s method applied to the function $f’$.

Second, we can imagine working with $f$ by approximating it by a quadratic function, minimizing that, and having that be our iterative procedure at each step. (Why a quadratic? Because that’s the smallest degree at which we could, in principle, have a minimizer or maximizer.) From the point $x_k$, we approximate

$$f(x) \approx f(x_k) + f’(x_k)(x - x_k) + f''(x_k)\frac{(x - x_k)^2}{2}.$$ 

A quadratic equation $y = ax^2 + bx + c$ has its vertex at $x$-coordinate $-\frac{b}{2a}$. In this case, thinking of this parabola in terms of $x - x_k$, we get $-\frac{b}{2a} = -\frac{f’(x_k)}{2 \cdot f''(x_k)}$. So if our next iterative point $x_{k+1}$ is the vertex of the parabola, it must satisfy

$$x_{k+1} - x_k = -\frac{f’(x_k)}{f''(x_k)}$$

which gives us the same recurrence as before.

These are the two ways to think about Newton’s method for minimization:

1. Apply Newton’s method to $f’$.

2. Approximate $f$ by a parabola.

Any way we look at it, this method is not very intelligent: it only finds critical points of $f$, so it cannot distinguish minimizers from maximizers. So when we use this method, we will want to use the second derivative test to classify the limiting point as a local minimizer or local maximizer, and if we got the wrong one, we can start over from some other point far away and try again.

Exactly the same approach works for minimizing $f : \mathbb{R}^n \rightarrow \mathbb{R}$. We can apply Newton’s method to the $n$-variable, $n$-dimensional function $\nabla f$, to find a critical point $\mathbf{x}$ where $\nabla f(\mathbf{x}) = \mathbf{0}$. If we do what we did earlier today, we get $\mathbf{x}^{(k+1)}$ from $\mathbf{x}^{(k)}$ by solving

$$\nabla f(\mathbf{x}^{(k)}) + Hf(\mathbf{x}^{(k)})(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = \mathbf{0}.$$ 

This can also be phrased as a quadratic approximation: we approximate

$$f(x) \approx f(\mathbf{x}^{(k)}) + \nabla f(\mathbf{x}^{(k)}) \cdot (x - \mathbf{x}^{(k)}) + \frac{1}{2}(x - \mathbf{x}^{(k)})^T Hf(\mathbf{x}^{(k)})(x - \mathbf{x}^{(k)})$$

and find the critical point of this approximation.