These notes draw heavily on *Numerical Recipes*, a valuable resource for entry-level understanding of numerical methods relevant to physics.

1. Minimization

Optimization, either minimization or maximization of functions, is a close relation of root-finding, but it has some fundamental differences. There is a sense in which they are the same problem, because finding a stationary point of $f(x)$ is like finding a root of $\mathrm{d}f/\mathrm{d}x$. Nevertheless, they do differ (particularly in multiple dimensions, due fundamentally to the fact that the gradient of a function has special properties helpful to finding its minimum).

In the game of optimization, the most important categorization is between *convex* and *non-convex* problems. Convex functions are ones where any line segment between two points on the function's surface lie above the function. These functions have a single optimum — e.g. a single minimum. So if you find a minimum, you have found the minimum. Nonconvex problems can have multiple optima — i.e. more than one local minimum, only one of which is a global minimum (unless there's a tie!).

We start with the simplest example of a convex problem, that of a quadratic optimization. We've already seen this problem in the context of SVD, but let's connect it explicitly to the idea of minimization.

2. Minimization of quadratic functions

If you have a one-dimensional parabola, it has exactly one extreme. We will write the parabola as:

$$f(x) = \alpha x^2 + \beta x + \gamma \tag{1}$$

In this case, what is the extremum of the function?

Typically, we will be handed a function, not given an explicit set of quadratic parameters (and we'll see later we actually want to use the quadratic form as an approximation to more general functions).

How many function evaluations do I need to do to determine the parameters of the quadratic?

In fact, one can work through this matrix to find α, β, nd γ (you don't really use γ, but you do need to calculate it) and thus the minimum (this is the same as Brent's method). Analytically you get the result:

$$x = b - \frac{1}{2}\frac{(b-a)^2\left[f(b)-f(c)\right] - (b-c)^2\left[f(b)-f(a)\right]}{(b-a)\left[f(b)-f(c)\right] - (b-c)\left[f(b)-f(a)\right]} \tag{2}$$

This method works great if the function is actually quadratic (not something else, and not linear). But is very rare for a function to be known to be quadratic, and not have the ability to calculate its derivative directly. The real use of this method is as a way to iterate twoards a solution.

3. Golden Section Search

For functions about which you know nothing, there is a more fool-proof method, akin to bisection, which is *golden section* search.

Imagine you start with a bracketing set of points a, b, and c, such that $f(b) < f(a)$ and $f(b) < f(c)$. Define the fractional position of b as:

$$w = \frac{b-a}{c-a} \tag{3}$$

Now we will pick a next trial point x. This will define some new triplet, depending on whether it is higher or lower than $f(b)$. Let's call the fraction position beyond b:

$$z = \frac{x-b}{c-a} \tag{4}$$

If the new triplet is the first three points, it has fractional length $w+z$. If it is the last three points, it has fractional length $1-w$. It seems reasonable to want these to be the same:

$$w + z = 1 - w \quad \rightarrow \quad z = 1 - 2w \tag{5}$$

which leaves each length as $1-w$.

If you are applying this iteratively, the fractional position of x within the new bracket should be the same as b was in the old bracket (since the latter was created by the same process):

$$\frac{z}{1-w} = w \tag{6}$$

and then plugging in for z, and solving for w we can find:

$$w = \frac{3-\sqrt{5}}{2} \approx 0.382 \tag{7}$$

This is the *golden section*.

This method will continually bracket the minimum further and further down.

4. Brent's one-dimensional minimization

A workable one-dimensional minimization routine combines these two methods, and is called Brent's Method. Like the root-finding Brent, it uses parabolic approximations, but it keeps track of a bracketing interval, and under certain conditions it reverts to golden section search.

These conditions are that:

- The parabolic step falls outside the bracketing interval.
- The parabolic step is greater than the step before last (!).

You shall, for your homework, implement this method.

5. N-dimensional minimization

Things get trickier in higher numbers of dimensions. A phenomenally bad way to minimize in high dimensions is to iteratively minimize along different coordinate directions. This will be extremely slow even in pretty reasonable cases. It will get caught in some degeneracy valleys. Another bad way that at first sounds good is *steepest descent*, where you minimize in the gradient direction, successively. It is easy to draw situations where most starting points will lead to very slow convergence.

Modern methods are robust to this problem; at times you will hear physicists describe this as if it were a mysterious, unassailable problem, but computer science has had it solved in a huge number of practical cases for decades.

The key idea for improving convergence is the same as that used in parabolic minimization. Near the minimum of the function, it will be quadratic at lowest order. Let's put the minimum at the origin for simplicity. If you expand it in Taylor series:

$$f(\vec{x}) = f(\vec{0}) + \sum_i \left[\frac{\partial f}{\partial x_i}\right]_{\vec{0}} x_i + \frac{1}{2}\sum_{ij} \left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]_{\vec{0}} x_i x_j \tag{8}$$

What is the gradient of this Taylor expansion?

If you had to choose successive directions to minimize one-dimensionally in, which would they be?

There is one sort of case where you may know the Hessian matrix. That is when your function f is quadratic in the parameters. A classic example is least squares minimization to a linear model (sometimes called χ^2 minimization, which is only appropriate to say if you are fitting data to a model with Gaussian noise). In this case if you have data points g_i, and a model $m_i = \sum_j a_j M_{ij}$ which is a linear function of unknowns a_j, then the function you want to minimize is:

$$f = \sum_i \left(g_i - \sum_j a_j M_{ij}\right)^2 \tag{9}$$

and your Hessian matrix can be constructed and the problem above solved. Always remember that if your function is quadratic in its parameters than minimizing it should be easy — it is convex and you can solve it in one matrix operation!

6. Conjugate Gradient

In more general cases, we do not know the Hessian matrix explicitly and need to estimate its structure based on evaluations of f (and its derivatives). How to estimate the Hessian matrix efficiently with function and derivative evaluations is the subject of the *conjugate gradient* method and its variants. In general, you will want to use one of the modern variants on conjugate gradient, which are good for many low-enough-dimensional and smooth problems. A good, extensive introduction is given by Shewchuk (1994).

The idea of conjugate gradient is simple, but its implementation is tricky. Basically, the idea is that if you knew the eigenvectors of the Hessian matrix, you could successively minimize along its eigenvectors and reach the minimum in N steps. This is clear when you remember that the eigenvectors form a basis in which the matrix is diagonal. Thus, if you construct a rotation matrix $\mathbf{R}$ from the eigenvectors:

$$\mathbf{A} = \mathbf{R}^T \cdot \mathbf{D} \cdot \mathbf{R} \tag{10}$$

where $\mathbf{D}$ is a diagonal matrix. Thus, any quadratic form can be written in the eigenbasis $\hat{e}'$ as:

$$\sum_i a_i (x'_i - \bar{x}_i)^2 \tag{11}$$

that is, without any cross-terms between different dimensions. Furthmore, we can define a more general transformation $\mathbf{T}$ so that:

$$\mathbf{A} = \mathbf{T}^T \cdot \mathbf{I} \cdot \mathbf{T} \tag{12}$$

just by renormalizing the eigenvectors. It is also worth noting at this stage that the columns of $\mathbf{T}$ satisfy:

$$\vec{t_i} \cdot \mathbf{A} \cdot \vec{t_j} = \delta_{ij} \tag{13}$$

If I transform into a new space defined by $\mathbf{T}$ then we can start in *any* direction and do steepest descent.

This fact is of limited use on its own. After all, you still need to determine the Hessian matrix and construct the eigenbasis. The magic of the conjugate gradient algorithm is that you can generate searches along the relevant directions. You start in any direction $\vec{h_0} = \vec{g_0}$ and line minimize in that direction. Then you update according to:

$$\begin{aligned} \vec{g}_{i+1} &= \vec{g}_i - \lambda_i \mathbf{A} \cdot \vec{h}_i \\ \vec{h}_{i+1} &= \vec{g}_{i+1} + \gamma_i \vec{h}_i \end{aligned} \tag{14}$$

with:

$$\begin{aligned} \lambda_i &= \frac{\vec{g}_i \cdot \vec{h}_i}{\vec{h}_i \cdot \mathbf{A} \cdot \vec{h}_j} \\ \gamma_i &= \frac{\vec{g}_{i+1} \cdot \vec{g}_{i+1}}{\vec{g}_i \cdot \vec{g}_i} \end{aligned} \tag{15}$$

Given this updating scheme, you line minimize successively on $\vec{h}_i$.

The directions $\vec{h}$ and $\vec{g}$ have the magical property that for $j < i$, $\vec{g}_i$ is orthogonal to all the previous $\vec{g}_j$ and $\vec{h}_j$, and crucially:

$$\vec{h}_i \cdot \mathbf{A} \cdot \vec{h}_j = 0 \tag{16}$$

so the resulting directions have the property we want. Showing that this is the case takes more time than I have here.

Except the λ_i still require the knowledge of $\mathbf{A}$. Luckily we can use the following trick. If $\vec{g}_i = -\nabla f(\vec{P}_i)$, then line minimize and pick the next $\vec{g}_{i+1}$ as the gradient again, this is the same as the update for $\vec{g}$ above. It does not require $\mathbf{A}$.

We can see this because in this case:

$$\vec{g}_i = -\mathbf{A} \cdot \vec{P}_i + \vec{b} \tag{17}$$

so:

$$\vec{g}_{i+1} = -\mathbf{A} \cdot (\vec{P}_i + \lambda \vec{h}_i) + \vec{b} \tag{18}$$

and so:

$$\vec{g}_{i+1} = \vec{g}_i - \lambda \mathbf{A} \cdot \vec{h}_i \tag{19}$$

with λ chosen at the line minimum. This λ turns out is the same as λ_i used above. To show this, note that at the minimum:

$$\vec{h}_i \cdot \nabla f = -\vec{h}_i \cdot \vec{g}_{i+1} = 0 \tag{20}$$

Then you can solve for λ in the above equation and you get the value for λ_i above.

This version of the algorithm is known as the Fletcher-Reeves algorithm. You start your minimization along the gradient, then you update according to how $\vec{h}_i$ evolves under the iteration above. If your function is quadratic, you converge in N steps. If not, you still proceed quite rapidly to the minimum.

You need derivatives! This is what you want `autodiff` for usually.

7. Quasi-Newtonian Methods

There are other techniques for finding the minimum through generating the Hessian matrix information, specifically building up a model for its inverse. These are known as quasi-newtonian

methods. The most commonly used of these nowadays is called Broyden-Fletcher-Goldfarb-Shanno (BFGS). This is the method of choice for minimization problems with a moderate number of dimensions.

In outline form, what BFGS does is to track an approximation to the inverse of the Hessian matrix. You can find this on the first step or just count on your iteration to converge.

- Search in the direction $-\mathbf{A}^{-1} \cdot \vec{\nabla} f(\vec{x}_i)$, and find $\Delta\vec{x}_k$.
- Update $\vec{x}_{k+1} = \vec{x}_k + \Delta\vec{x}_k$.
- Calculate $\vec{y}_k = \vec{\nabla} f(\vec{x}_{k+1}) - \vec{\nabla} f(\vec{x}_k)$.
- Then use the following approximation for the update to the inverse of the Hessian:

$$\mathbf{A}_{k+1}^{-1} = \mathbf{A}_k^{-1} + \frac{\left(\Delta\vec{x}_k \cdot \vec{y}_k + \vec{y}_k \cdot \mathbf{A}_k^{-1} \cdot \vec{y}_k\right) s_k^2}{\left(\vec{s}_k \cdot \vec{y}_k\right)^2} - \frac{\mathbf{A}_k^{-1} \cdot \vec{y}_k \vec{s}_k + \vec{s}_k \cdot \vec{y}_k \cdot \mathbf{A}_k^{-1}}{\vec{s}_k \cdot \vec{y}_k} \tag{21}$$

Showing why this is an approximation to the inverse Hessian update is well beyond our scope. The key thing here is to remember that gradients are essential and make this sort of thing possible.

`jax`, for example, has a BFGS implementation in its version of `scipy.optimize`.