EN.540.635 "Software Carpentry"

Lab - Conjugate Gradient

In class, we went through a quick implementation of the Steepest Descent method in the context of optimization in Python. This method is really a trivial application of the Quasi Newton method, but instead of defining our inverse Hessian, we allow it to be a scaled version of the identity matrix (by some small α step size). To motivate this, we go into a bit more detail of the derivation of such a method. The Quasi Newton method starts from the generalized quadratic equation, as shown in Eq. 1 (generalized in N dimensions).

$$f = f_0 + \sum_{i=1}^{n} a_i x_i + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} B_{ij} x_i x_j \tag{1}$$

We will make use of the bra-ket notation to simplify notation. Again, this kind of notation is commonly used in quantum mechanics and is a useful way to notate vectors and linear operations. This is outlined as follows:

$$\begin{aligned} |x\rangle &= \text{column vector} \\ \langle x| &= \text{row vector} \\ \langle x|x\rangle &= \text{inner product (should be a scalar)} \\ |x\rangle \langle x| &= \text{outer product (should be a matrix)} \end{aligned} \tag{2}$$

Where, in either the bra or ket, we will assume that the vectors are represented as 1D arrays. Now, we can rewrite our generalized form (Eq. 1) using this notation:

$$f = f_0 + \langle a|x\rangle + \frac{1}{2} \langle x|B|x\rangle \tag{3}$$

In the last term of this equation, B is a linear operator (matrix) and the final quantity will be a scalar. The Quasi Newton method simply correlates with what we learned in basic calculus. To minimize, take the derivative and set it equal to 0. This allows us to calculate an expression for the gradient.

$$\begin{aligned} \frac{df}{dx} &= \frac{d}{dx}\left(f_0 + \sum_{i=1}^{n} a_i x_i + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} B_{ij} x_i x_j \right) \\ &= 0 + \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} \sum_{j=1}^{n} B_{ij} x_i \\ \implies |g\rangle &= |a\rangle + B|x\rangle \end{aligned} \tag{4}$$

In this case, we will make this an iterative approach! First, we can define our convergence criteria - we want our gradient to be equal to 0:

$$|0\rangle = |a\rangle + B|x_{min}\rangle \tag{5}$$

Next, we want to find a "step" to take that is in the direction of the minimum and then we will take said "step". To define this analytically, we can take the difference between equations 4 and 5 and plug in the appropriate inverse Hessian in the case of the Quasi Newton method:

$$\begin{aligned} |0\rangle - |g\rangle &= B|x_{min}\rangle - B|x\rangle \\ \implies |s\rangle = |x_{min}\rangle - |x\rangle &= -B^{-1}|g\rangle = -H|g\rangle \end{aligned} \tag{6}$$

Now we can see that to take a "step" towards a minimum, we need only change our coordinates by adding a small part of $|s\rangle$, even if we do not consider the inverse Hessian - this gives us the Steepest Descent method. An example algorithm, similar to what we went over in class is detailed below in Algorithm 1.

Algorithm 1 Steepest Descent

1: $|x^i\rangle \leftarrow$ starting_guess()
2: $H^0 = I$
3: **while** $i < N_{max}$ and $\langle g^i | g^i \rangle > \varepsilon$ **do**
4: $\quad |g^i\rangle \leftarrow$ get_gradient$(|x^i\rangle)$
5: $\quad |s^i\rangle = -H^0 |g^i\rangle$
6: $\quad |x^{i+1}\rangle = |x^i\rangle + \alpha |s^i\rangle$
7: **end while**

Step 1 - Extend to Conjugate Gradient

Now we can focus on writing an algorithm for the Conjugate Gradient method, which is somewhat similar to the Steepest Descent method. The main difference is that our step size is modified with each iteration, instead of keeping it constant! That is, instead of simply being the negative gradient, it is given by the expression shown in Eq. 7. Note, in Eq. 7 there is a β value. Many representations of this scaling factor exist; however, the one we will use is that of Fletcher-Reeves, shown in Eq. 8.

$$|s^i\rangle = -|g^i\rangle + \beta |s^{i-1}\rangle \tag{7}$$

$$\beta = \frac{\langle g^i | g^i \rangle}{\langle g^{i-1} | g^{i-1} \rangle} \tag{8}$$

Using this knowledge, write a function to run the Conjugate Gradient method, similar to the Steepest Descent method in the provided code. Test out both methods on the given test functions (2D quadratic and the sinusoidal functions).

Step 2 - Implement SciPy Optimization

Optimization algorithms are well-known methods and are already implemented in code packages across all languages. It makes more sense to implement an already written code than to spend time writting (and debugging) your own. As such, implement SciPy's optimization functionalities (see the minimize function) and run it on the same test functions as Step 1.

Step 3 - Fitting a Curve

In practice, we use optimiation algorithms for problems where the explicit function is unknown. For example, we already knew the functional forms of our objective functions, and it is not hard to just analytically solve for the minimina instead of using a numerical method. However, there are times where the function is too complex to solve analytically. A simple application of this kind of problem exists in curve fitting. When we use an application such as Excel to find the best-fit line to a sequence of data points, we are effectively asking Excel to use an optimizer to minimize some error between the line and the data points. In this part of the assignment, you will implement the minimization yourself (using SciPy)! Note, you will have to define the error function to be minimized and implement the minimizer part on your own.

At the very end, plot a scatter plot of the observed data (y_obs) as well as the best fit line you have. Make a legend that shows the $y = mx + b$ equation (fill in m and b with your discovered values), and also show what your R^2 value is in the legend - a function has been provided to calculate this. An example of what is expected is shown in Fig. 1 (note, this example is of a different plot, so yours will look different).

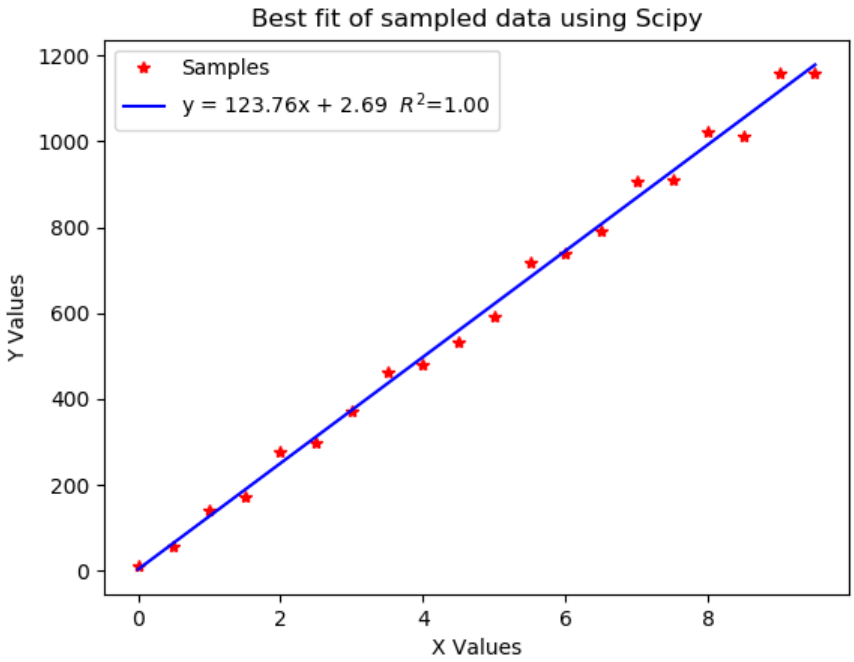

Figure 1: A best fit line using Scipy's optimizer.