Lecture 3: Introduction to numerical methods for optimization

Antoine Chapel

- Why is numerical optimization even necessary ?
- The mathematical objects of numerical optimization
- The basic methods: gradient descent and coordinate descent
- The limits of numerical methods

When could solving FOC by hand not be sufficient ?

Unconstrained Optimization

Let us forget about constraints for today.

- Function: $f(x)$, $x \in \mathbb{R}^n$
- Gradient: $\nabla f(x)\big|_{x_k} = \left(\frac{\partial f}{\partial x_1}\Big|_{x_k}, \frac{\partial f}{\partial x_2}\Big|_{x_k}, \dots, \frac{\partial f}{\partial x_n}\Big|_{x_k}\right)$, that we denote g_k
- Hessian $\nabla^2 f(x)\big|_{x_k}$, that we denote H_k
- the Newton-Raphson algorithm: $x_{k+1} = x_k - \lambda H_k^{-1} \cdot g_k$
- Where $\lambda \in (0, 1)$ is a step size.

Computing gradient and hessian

- Suppose you have a function $f : \mathbf{R}^n \to \mathbf{R}$
- The most computationally efficient option is to compute yourself the gradient and hessian functions and input them into the algorithm.
- Otherwise, you will need to approximate numerically the gradient and hessian
- Element j of the gradient is the partial derivative of f w.r.t x_j: $\frac{\partial}{\partial x_j} f \approx \frac{f(x+h \cdot e_j) - f(x-h \cdot e_j)}{2h}$, where h is a small value and e_j is a vector of zeros with value 1 at index j.
- To find the hessian, think that it is simply the gradient of the gradient: $\frac{\partial}{\partial x_i} \frac{\partial}{\partial x_j} f$

The numerical gradient

Start with the function $f(x,y) = x^2 + y^2$, and evaluate its gradient and Hessian at point $(x,y) = (2,3)$

You know that $\nabla f(x,y)\Big|_{(2,3)} = (2x, 2y)\Big|_{(2,3)} = (4,6)$, and $\nabla^2 f(x,y) = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$

For the gradient, our code will perform the following computation:

$$g_{(2,3)}[0] = \frac{f(2+0.001,3) - f(2-0.001,3)}{2 \cdot 0.001} = \frac{(2.001^2 + 3^2) - (1.999^2 + 3^2)}{0.002} = 4$$
$$g_{(2,3)}[1] = \frac{f(2,3+0.001) - f(2,3+0.001)}{2 \cdot 0.001} = \frac{(2^2 + 3.001^2) - (2^2 + 2.999^2)}{0.002} = 6$$

Note that, since our function is particularly smooth, we get exactly the value of the gradient. In general, what you get is an approximation.

The numerical Hessian

$$H_{(2,3)}[0,0] = \frac{g_{(2+0.001,3)}[0] - g_{(2+0.001,3)}[0]}{2 \cdot 0.001}$$
$$H_{(2,3)}[1,0] = \frac{g_{(2+0.001,3)}[1] - g_{(2+0.001,3)}[1]}{2 \cdot 0.001}$$
$$H_{(2,3)}[0,1] = \frac{g_{(2,3+0.001)}[0] - g_{(2,3+0.001)}[0]}{2 \cdot 0.001}$$
$$H_{(2,3)}[1,1] = \frac{g_{(2,3+0.001)}[1] - g_{(2,3+0.001)}[1]}{2 \cdot 0.001}$$

Let us compute the first one by hand:

$$H[0,0] = \frac{\frac{(2.002^2+3^2)-(2^2+3^2)}{2*0.001} - \frac{(2^2+3^2)-(1.998^2+3^2)}{2*0.001}}{2 \cdot 0.001} = 2$$

This confirms our analytical results.

The Newton-Raphson algorithm

Suppose you want to minimize our function $f(x, y) = x^2 + y^2$. You start from a random point: $(x_0, y_0) = (2, 3)$. You set a step size $\lambda = 0.1$

1. Compute the gradient at current point: $\begin{bmatrix} 4 \\ 6 \end{bmatrix}$
2. Compute the Hessian: $\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$ and invert it: $\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$
3. $$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} - 0.1 \cdot \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \cdot \begin{bmatrix} 4 \\ 6 \end{bmatrix} = \begin{bmatrix} 1.8 \\ 2.7 \end{bmatrix}$$
4. Go back to step 1, until the gradient is close enough to 0

Let us code this algorithm in Python

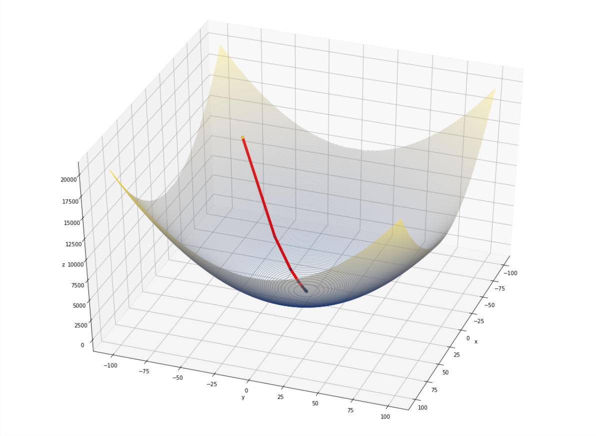

Figure 1: Gradient Descent

The coordinate descent algorithm

This alternative algorithm is computationally much lighter than the first: it does not require the Hessian.

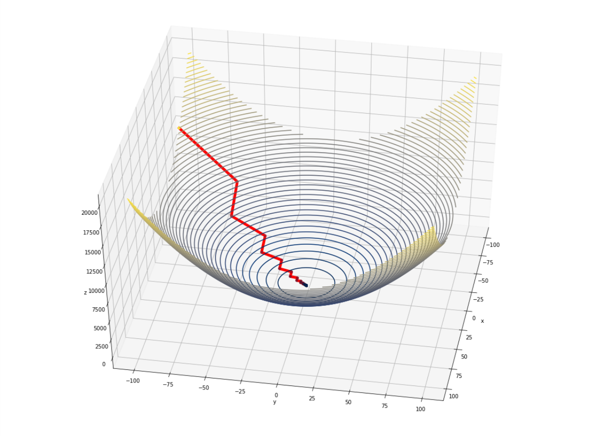

Figure 2: Coordinate Descent

Coordinate Descent Algorithm

For that algorithm, given an initial point $x^0 = (x_1^0, x_2^0 ..., x_n^0)$, you only need to compute one partial derivative per iteration.

1. $x^{k+1} = x^k - \lambda \cdot \begin{bmatrix} 0 \\ ... \\ 0 \\ \frac{\partial f}{\partial x_i} \\ 0 \\ ... \\ 0 \end{bmatrix}$
2. $i = i + 1$, go back to step 1

Yet, this algorithm may fail if the function is not smooth. You will encounter both gradient and coordinate descent in math+econ+code

When these methods may fail

One can think of two main sort of failures

1. The procedure takes ages to reach a minimum
2. The procedure gets stuck in a local minimum due to nonconvexity

What solutions do you know/can you think of ?