Optimisation TD 2
Levenberg-Marquardt method

C. Frindel

4 Novembre 2016

1 Introduction

The goal of this exercise is to implement the Levenberg-Marquardt (LM) optimisation method. It's a "clever" method which combines advantages of gradient descent and of Newton's method we saw in the previous exercise.

Reminder: At first order, descent direction d_1 of a function f at point x is given by the following formula : $d_1 = -\nabla_f(x)$.
At second order, descent direction d_2 of a function f at point x is given by the following formula : $d_2 = -\frac{\nabla_f(x)}{\nabla_f^2(x)}$.
LM method consists in the introduction of a parameter $\lambda > 0$ which encourages the use of first order over second order, when it is appropriate to to so. It does that by ponderating the influence of the diagonal terms of the hessian matrix, $H = \nabla_f^2(x)$, by defining a new matrix H^{LM} :

$$\begin{aligned} H_{ii}^{LM} &= H_{ii}(1+\lambda) \\ H_{ij}^{LM} &= H_{ij} \end{aligned}$$

It is this matrix H^{LM} used to compute the descent direction : $d_{LM} = -\frac{\nabla_f(x)}{H_f^{LM}}$. If λ is close to zero, H^{LM} tends towards the expression of a classic Hessian and so $d_{LM} \to d_2$. If λ is very big, diagonal terms of H^{LM} become preponderant, which means the descent direction will tend to be the steepest slope (following the gradient) : $(d_{LM} \to d_1)$. The value of λ will evolve as a function of the proximity to the minimum of the cost function.

2 Non linear regression

In this exercise, this method will be applied to non-linear regression. We suppose we have data (observations) in the form of couples (x_i, y_i). We're looking for the parameters of a given function g which will approximate as closely as possible this data. We suppose that g is known and depending on a parameter vector $\mathbf{a} \in \mathbb{R}^n$. To obtain the non-linear regression, we want to minimize (by optimizing $\mathbf{a}$) the squared sum error between observations y_i and function $g(\mathbf{a})$ estimated in points x_i.

$$f(\mathbf{a}) = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - g(x_i, \mathbf{a})\right)^2$$

The gradient terms of f with respect to $\mathbf{a}$ are :

$$\frac{\partial f}{\partial a_k} = -\sum_{i=1}^{N}\left(y_i - g(x_i, \mathbf{a})\right)\frac{\partial g}{\partial a_k}(x_i, \mathbf{a})$$

Second order derivatives of f with respect to $\mathbf{a}$ are :

$$\frac{\partial^2 f}{\partial a_l \partial a_k} = \sum_{i=1}^{N}\left[\frac{\partial g}{\partial a_k}(x_i, \mathbf{a})\frac{\partial g}{\partial a_l}(x_i, \mathbf{a}) - \left(y_i - g(x_i, \mathbf{a})\right)\frac{\partial^2 g}{\partial a_l \partial a_k}(x_i, \mathbf{a})\right]$$

In the non-linear regression context, it is possible to use the Gauss-Newton approximation which allows us to avoid the computing of second derivatives. Indeed, this approximation enables to deduce second order derivatives from the first order ones.

Approximation of Gauss-Newton : We'll suppose function g regular enough and term $(y_i - g(x_i, \mathbf{a}))$ small enough to approximate the second derivatives by :

$$\frac{\partial^2 f}{\partial a_k \partial a_l} = \sum_{i=1}^{N} \frac{\partial g}{\partial a_k}(x_i, \mathbf{a}) \frac{\partial g}{\partial a_l}(x_i, \mathbf{a})$$

Notice how the secondary derivatives are expressed only as a product of first order derivatives. Full computing of the Hessian will therfore not be necessary in this exercise. Do note, however, that this approximation is valid only here in the context of a sum of squared error minimization (like here, for non linear regression).

3 Mono-exponential case

We suppose $g(x) = e^{-ax}$ with $a \in \mathbb{R}$. We will peform a simple non-linear regression in a space of dimension 1.

1. Briefly remind the advantages and drawbkacks of descent methods seen in the previous exercise.
2. Write function g which takes x and a as arguments.
3. Create a noisy dataset on the intervall $x \in [0, 3]$ with $a = 2.0$, st $y = g(a, x) + bN(0, 1)$ with b a parameter controlling the amplitude of a normally distributed noise. Use a step of 0.01 for x. Use function `randn` from `numpy.random` library.
4. Plot your dataset with function `pyplot` of `matplotlib` library.
5. Write cost function f which takes x, y and a as arguments.
6. Write the function which returns the gradient of f wrt a.
7. Write the function which returns the second order derivaties of f. Use Gauss-Newton approximation.
8. Implement LM algorithm.
    - Initialisation of a at 1.5
    - Initialisation of λ at 0.001
    - While [stopping conditions (get ideas from previous exercise)] :
        - Compute gradient for current a
        - Compute secondary derivatives
        * Compute H^{LM}
        - Compute descent direction d_{LM}
        - Compute new cost $f(a^{(k+1)}) = f(a^{(k)} + d_{LM})$
        - If this cost is smaller than current cost, do $a^{(k+1)} = a^{(k)} + d_{LM}$ and divide λ by 10
        - Otherwise, multiply λ by 10 and start again at $(*)$
9. Study the evolution of the algorithm's parameters (λ, gradient norm, value of f) as iterations go. Comment.
10. Try different amplitudes for the noise in your data. Study its influence on convergence of the algorithm.
11. Plot on the same plot your dataset and function $g(x, a)$ optimale for the different amplitudes you tried.

4 Bi-exponential case

We'll study now the case where f is from $\mathbb{R}^2$ to R, i.e. a function g with two parameters in entry. Let $g(x, \mathbf{a}) = x^{a_1} e^{-a_2 x}$ with $\mathbf{a} = [a_1, a_2]$.

1. Define like before funtion g, a noisy dataset (for $x \in [0, 5]$, $a_1 = 2.0$ and $a_2 = 3.0$) and function f.
2. Compute gradient of f wrt to $\mathbf{a}$.
3. Compute second order derivative matrix with Gauss-Newton approximation. Indication : use function dot of `numpy` to compute a scalar product.
4. Run the LM algorithm to compute optimal vector $\mathbf{a}$, with an initialisation of $\mathbf{a}$ at [1.5,1.5]. Indication : To compute the inverse of a matrix, use function `inv()` from the `numpy.linalg` package.
5. Study the evolution of the algorithm's parameters (λ, gradient norm, value of f) as iterations go. Comment and test for several noise amplitudes.
6. Plot results with `matplotlib` for several noise amplitudes.
7. Conclude on advantages of the LM method. Compare it with traditionnal descent methods.