1. Basic knowledge beforehand
2. Gradient (of a scalar over a vector)

Note: is the direction that **increases** *f* the fastest!

1. Hessian matrix

Note:

1. we can usually express a quadratic function in the term of:

In the above expression, *A* (a N by N symmetric matrix) would be the hessian matrix (that also explains why quadratic programming is faster than fmincon: no need to calculate Hessian again!).

Note: If *f* is a vector instead of scalar, the Hessian matrix becomes a 3D matrix.

1. Sometimes we also use to denote the Hessian matrix.
2. Jacobian matrix

A Jacobian matrix is the gradient of a vector (M by 1) over another vector (N by 1).

In fact the gradient in (1) is just a special case for Jacobian.

1. Taylor Series
2. Newton method in function solving

Based on Taylor series if a function is differentiable then

So we can iteratively solve the problem:

Usually we also add a coefficient *α* to control the rate of iteration.

1. Chain rule of derivative

First derivative:

Second derivative:

Note: the above equation was not intuitive at first sight. However I must think about the **product (calculus) rule** carefully. (https://en.wikipedia.org/wiki/Product\_rule)

1. Convex optimization

An optimization problem can be expressed by:

Where *X* is a set of all possible values of *x*.

An optimization problem is convex if and only if both following conditions are met:

1. is a convex set (Jensen inequality: for any , there must have , where ).
2. is a convex function (Jensen inequality: , where ).

Note: in many realistic cases the subset can be expressed as linear equality/inequality conditions. **These conditions ALL belong to convex subsets**!

**For following discussions we define the optimization target as .**

1. Gradient descent

Gradient descent (steepest descent) method tries to move the variable vector in the **opposite** direction of the Jacobian matrix, such that the value of *f* keeps decreasing in each iteration:

Here *α­k* is also called “learning rate”. Some algorithms will adjust this rate to make the process fast in the beginning and avoid zig-zag in the end.

**Note: pay attention to the difference between gradient descent and Newton’s method !**

1. Newton’s method in optimization (also called Gauss-Newton method)

Again, based on Taylor series, this time we can expand the target function to the second order:

If we fix and take the derivative of the function over and set it to zero:

As with gradient descent method, we also usually use “learning rate” to control the step size.

Note: intuitively Newton’s method is better than simple gradient descent because it uses second-order derivative information. However, **not every Hessian function is invertible**. Also the inverting can take extra time. Some also argue that if the initial value is too far from the optimal point, the Taylor series may not be close enough to the target function hence the method can fail.

1. Levenberg-Marquadt

Before the derivation, we must emphasis that (a) Levenberg-Marquadt method could be seen as an adaptive method between the Gradient descent and the Gauss-Newton method; (b) we hope to use L-M method in SRF fitting. Hence it is important to recite the Gradient descent method and Gauss-Newton method in the form of SRF fitting.

Statement of problem: for the simplest case, in a single-pixel event, we have a series of samples. Assuming we already knew the system response matrix (*P*) at 662 keV, how do we estimate the energy and triggering time of event?

We know that the measured signal can be described as:

*P* is the SRF matrix measured at 662 keV, z is the depth of interaction, E is the energy of interaction, T is time offset, is the noise (subject to normal distribution but **can be improved by weighting**). For each event we try to find the best fit of vector. Re-forming the mathematical model:

We use least-squares as a measure of goodness of fit (assuming that noise at each cell is i.i.d normal).

We must make two assumptions: (a) the SRF at different energies are the same (charge cloud effect is small), (b) Jacobian matrix can be approximated by evaluating the SRFs (material has reasonable continuity).

For all the three methods we iteratively update the estimate of:

1. Gradient descent method: directly use the first derivative of *S* over.

Note J is the Jacobian matrix of over.

1. Newton method

Note: Initially I was very confused by the form of iteration here. In my attempt, I tried to directly apply Taylor expansion on S over. This generated a somewhat similar form of iteration, but not exactly the same. In the derivation I also had to apply the chain rule of second derivative which was very challenging.

Later I realized the approximation was done by first order approximation of:

Now we already have the evaluation of, we should take the derivative of over and set it to zero. Then we have the form:

1. Levenberg-Marquadt method (trust-region)

As we mentioned, Levenberg-Marquadt method is the adaptive method varying between gradient descent and Newton method. The adaptive way is added as:

Note: must not be negative. We also call the “damping coefficient”. There are several ways to update this coefficient (for example, one method is to normalize it with the maximum of).

Note: For our detector problem, we know that the noise is not i.i.d (for example the noise in cathode channel is much higher than anode). We can add the weights as following:

1. Conjugate gradients