

# Algorithmica Research AB

## **Model calibration with non-linear optimization routines** using Quantlab 3.0

## Model calibration with non-linear optimization routines

In this short document we try to show how to calibrate model parameters for any arbitrary model using the non-linear optimization routines. Code examples are all written in Qlang version 3.0 using the new feature - function pointers.

### ***Using the Levenberg-Marquardt method (min\_l2)***

The method was first published by Levenberg (1944) and was rediscovered by Marquardt (1963). It is a very fast solver for well behaved curve fitting problems like those often found in the field of yield curve modelling. To converge it needs a reasonable good initial guess and then uses an iterative algorithm to find the solution.

The min\_l2 solver is a least squares solver with the possibility to specify a weighting scheme for the observed data, as well as upper and lower bounds for the model parameters.

An objective function must return a vector of calculated function values and take the model parameters an input vector argument.

Let's look at an example.

We have created a yield curve model and want to calibrate the model to market data. In this example we use observed data as an explicitly written global variable. For an indirect version using a data object see the example below.

Since we at the end of the calibration want to plot the fitted model together with the observed data points we create a helper function the will take the scalar model version and expanding it to a vector version for the optimization routine. By doing so, the plotting function can directly use the same model that we used for calibration. Without the plotting function this step is obviously not necessary.

Code with comments will follow;

```
// Example of model calibration to market data using min_l2
// Some dummy data that in the normal case comes from a database or the real time feed
vector(number) x = [0.1,0.5, 0.75,1,2,3,7 ,10, 15];           //observed maturities
vector(number) y = [6,4.9, 5.4,5.2 ,5,5.5,6,6.5,6.6 ];       //observed yield

number n = v_size(x);           //number of observations
vector(number) p_fitted;       //global storage of calibrated model parameters

// a simple parametric function with parameters i.e. our newly developed yield curve model
number QuadModel(vector(number) p, number m)
{
    number a = p[0];
    number b = p[1];
    number c = p[2];
    number d = p[3];

    return a + b*m+ c*log(m)+ d*m^2;
}

// help function for the optimization routine returning a vector of f(x)
// the function will vector-expand over all observed maturities and return
// the vector with corresponding model yields
vector(number) QuadModelOpt(vector(number) p)
{
    return QuadModel(p, x);
}

vector(number) QuadFit()
{
    vector(number) w = one_vector(n);           //relative weights for each point (i.e. equally weighted)
    vector(number) p = [0,0,0,0];             // initial guess for parameters

    min_l2(&QuadModelOpt, p, y, w);           //the optimization call using L2-norm
    return p;                                 // return the fitted parameters p
}
```

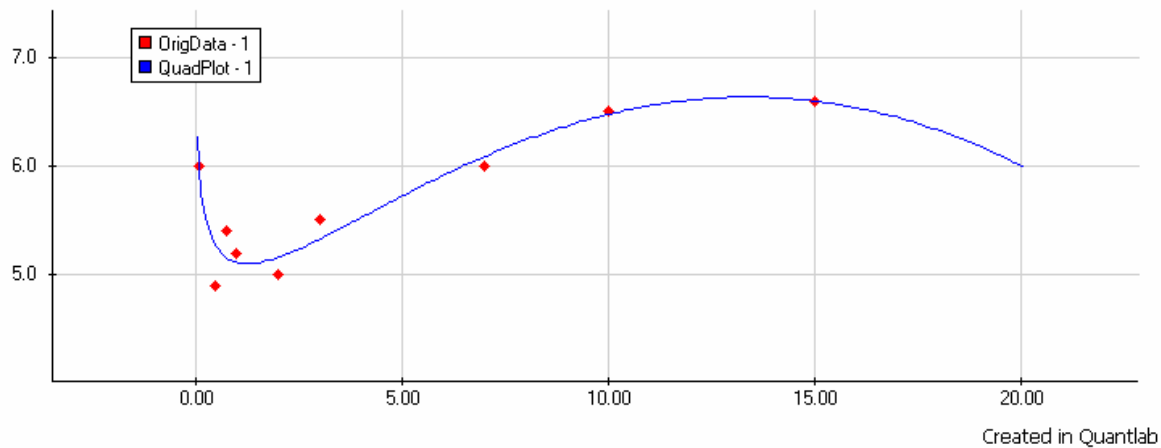
By calling the QuadFit function, the model will be calibrated and the parameters to the function returned. So let's use it in a function that will plot the modelled data in a graph together with the initial observations.

```
//plotting the original data points
out vector(point_number) OrigData() = point(x, y);

//plotting the QuadModel using the fitted parameters
out series<number>(number) QuadPlot()
{
    p_fitted = QuadFit();           //asking for the fitted params and storing in global variable

    return series(time:0.05,20,0.1;QuadModel(p_fitted, time));
}
```

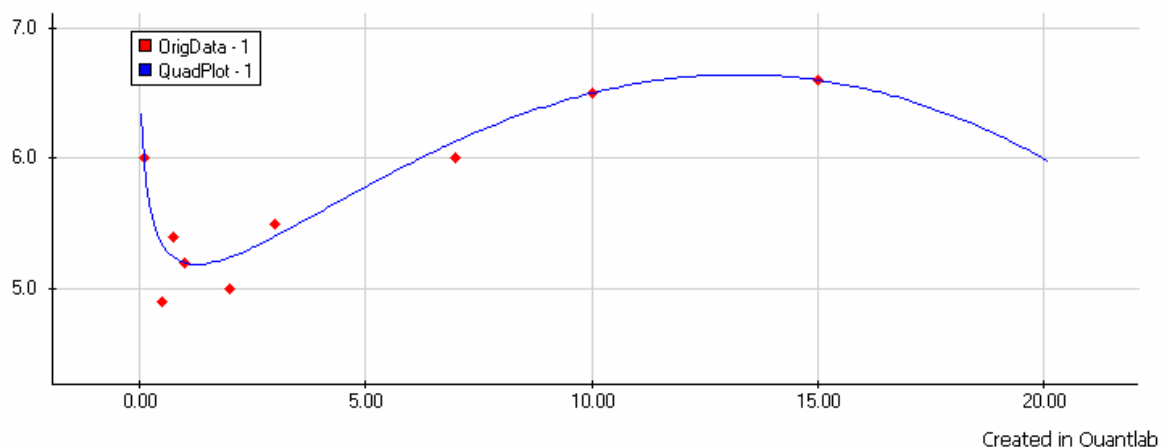
Now we can attach the two “out” declared functions in a graph.



### **Hald - min L1 optimization**

This model is adapted from the paper by J Hald and K Madsen, "Combined LP and Quasi-Newton methods for nonlinear l1 optimization", SIAM, vol 22, 1985

The example is identical to the min\_l2 case but using the min\_l1 function instead.



### **Downhill Simplex using Simulated Annealing (min\_dhsa)**

This method is also commonly referred to as the Nelder-Mead method [see Computer Journal, 1965, vol 7]. The implemented Quantlab code is an adaptation of the original article together with the simulated annealing extension as outlined in Numerical Recipes.

A difference compared to the min\_l1 and min\_l2 functions is that the min\_dhsa will minimize a *scalar* number. This means that it is up to the user to decide how the objective function should be minimized.

Here we will have a function on the form:

$\sum(f(x) - y)^n$  where  $x$  and  $y$  will be vectors and hence the norm would be  $n$ .

Let's look at the same example but using the `min_dhsa` instead.

It is not easy to choose the parameters in the `dhSA` function and to do so is more of an art than science. Below are well functioning settings.

`n_coolings = 600, max_per_cooling = 100, T0 = 0.01, dT = 0.9`

However the setting will lead to a slow fit the number `max_per_cooling` reduced to say 10 will vastly enhance speed. Speed is proportional to `n_coolings*max_per_cooling`.

```
// Example of model calibration to market data using min_dhSA
// Some dummy data that in the normal case comes from a database or in real time feed
vector(number) x = [0.1,0.5, 0.75,1,2,3,7 ,10, 15];           //observed maturities
vector(number) y = [6,4.9, 5.4,5.2 ,5,5.5,6,6.5,6.6 ];       //observed yield

number n = v_size(x);           //number of observations
vector(number) p_fitted;       //global storage of calibrated model parameters

// a simple parametric function with parameters i.e. our newly developed yield curve model
number QuadModel(vector(number) p, number m)
{
    number a = p[0];
    number b = p[1];
    number c = p[2];
    number d = p[3];

    return a + b*m+ c*log(m)+ d*m^2;
}

// help function for the optimization routine returning a vector of f(x)
// the function will vector-expand over all observed maturities and return
// the vector with corresponding model yields
number QuadModelOptDhSA( vector(number) p)
{
    return v_sum(sqr(QuadModel(p,x) - y));
}

vector(number) QuadFit()
{
    vector(number) p = [0,0,0,0];           // initial guess for parameters

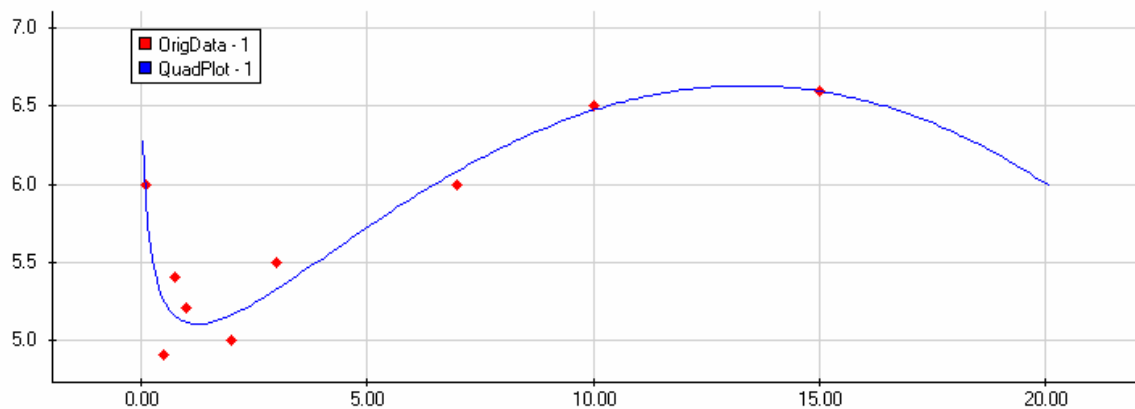
    min_dhSA(&QuadModelOptDhSA, p, 600, 100, 0.01, 0.9);
    return p;                               // return the fitted parameters p
}
```

Plotting the fitted model is of course exactly the same as for the other methods.

```
//plotting the original data points
out vector(point_number) OrigData() = point(x, y);

//plotting the QuadModel using the fitted parameters
out series<number>(number) QuadPlot()
{
    p_fitted = QuadFit();           //asking for the fitted params and storing in global variable
    return series(time:0.05,20,0.1;QuadModel(p_fitted, time));
}
```

Now we can attach the two “out” declared functions in a graph.



Created in Quantlab

## Fitting using the data object version

In applications where the data to fit will vary or come from an unknown source it is nice to be able to work with a data object instead of global variables.

The function to minimize as well as the call to optimization function will have an extra parameter containing the data object.

In this example we use the same structure and code as in the `min_11` example, but now using the data object instead. We also extend the example to fill the data object with some real-life market data using the Quantlab curve definition. This will give us the market yields in real-time with a new fit calculated each time the market updates with a tick.

```
// Example of model calibration to market data

// A data object to fill with observed data
object obs_data{

    vector(number) maturity;
    vector(number) yield;
};

// a simple parametric function with parameters i.e. our newly developed yield curve model
number QuadModel(vector(number) p, number m)
{
    number a = p[0];
    number b = p[1];
    number c = p[2];
    number d = p[3];

    return a + b*m + c*log(m) + d*m^2;
}

// help function for the optimization routine which takes the data object and params
vector(number) QuadModelOpt(obs_data o, vector(number) p)
{
    return QuadModel(p, o.maturity);
}
```

```

vector(number) QuadFit(obs_data o)
{
    vector(number) w = one_vector(v_size(o.maturity));    //relative weights for each point
    vector(number) p = [0,0,0,0];                        // initial guess for params

    // the optimization function also take the data object as input
    min_l1(&QuadModelOpt, o, p, o.yield, w);
    return p;
}

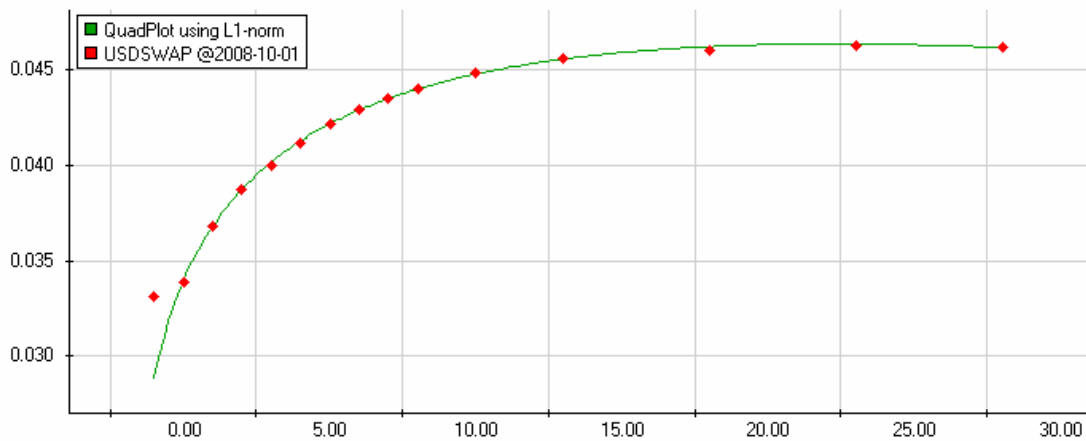
//*****
/* Functions below for plotting model in graph and table
//*****

//plotting the QuadModel using the fitted parameters
//we give the user a choice of curve to model and also for which date
out series<number>(number) QuadPlot2(curve_name c, date d)
{
    //create a new obs_data object and fill it with data
    obs_data ob = new obs_data;
    ob.maturity = curve(c,d,'mid').instruments().time_to_maturity();
    ob.yield = curve(c,d,'mid').instruments().yield();

    // call the model fit using the data object as input
    p_fitted = QuadFit(ob);

    return series(time:1,30,0.1;QuadModel(p_fitted, time));
}

```



Created in Quantlab