

**University of  
Newcastle upon Tyne**



***DEPARTMENT OF CHEMICAL  
AND PROCESS ENGINEERING***

# **UNCONSTRAINED MULTIVARIABLE OPTIMISATION**

Lecture 3: Steepest descent, Levenberg-Marquardt and quasi-Newton methods

Written: Mark Willis  
Date: May / June 1999  
Contact: [mark.willis@ncl.ac.uk](mailto:mark.willis@ncl.ac.uk)

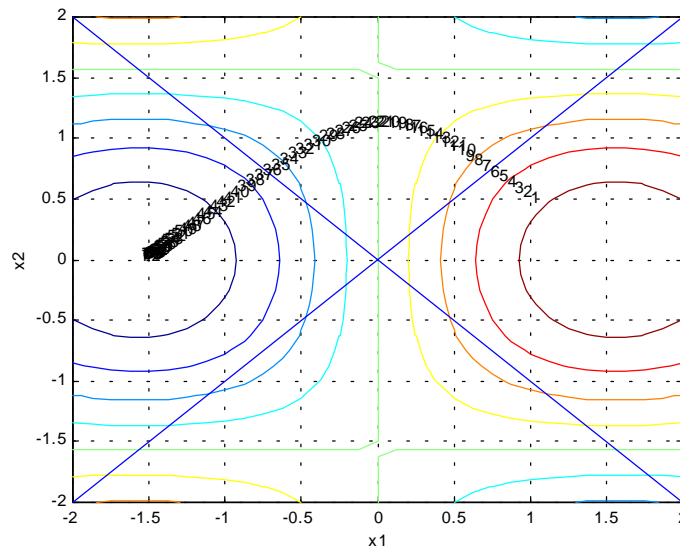
## Steepest descent

The steepest descent method uses the negative of the gradient vector as the direction of minimisation<sup>1</sup> and the gradient vector as the direction for maximisation. In other words the search direction for minimisation is,

$$\mathbf{s}^k = -\mathbf{J}(\mathbf{x}^k) \quad (17)$$

When compared to Newton's method, a steepest descent algorithm can be considered as one that approximates  $\mathbf{H}(\mathbf{x}^k)$  with the identity matrix  $\mathbf{I}$  (this is positive definite therefore ensuring movement towards a minimum) however as much information regarding the curvature of the function has been discarded the method is expected to be inefficient.

Figure (6) shows the trajectory of the steepest descent algorithm along the actual surface initiated at  $\mathbf{x} = [1 \ 0.5]$  with a fixed step size,  $\alpha = 0.1$ . The termination criteria used was  $|f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| < \varepsilon_1$  where  $\varepsilon_1 = 1e^{-4}$ . The numbers (which admittedly are difficult to see after the first few) mark the point in the 2-dimensional space which represents the new  $\mathbf{x}$  after the iteration. The total number of iterations taken was 72.



**Figure (6). Optimisation of the function  $y = \sin(x_1) * \cos(x_2)$  using steepest descent initiated at  $\mathbf{x} = [1 \ 0.5]$  with a fixed step size,  $\alpha = 0.1$ .**

Note,

---

<sup>1</sup> Intuitively it may be expected that this would be a reasonable direction. When coming down a hill, the shortest path is the one with the steepest gradient. In numerical optimisation this (*local*) direction is also the one which will decrease the function at the fastest rate.

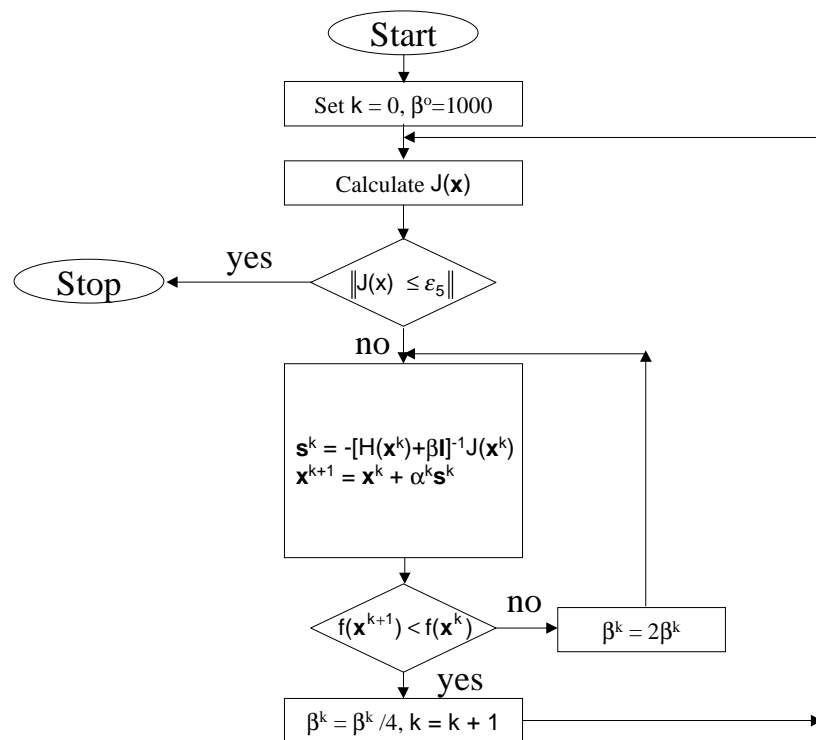
(1) The steepest descent method is inefficient (a relatively large number of iterations are required before the solution is obtained). However it is robust. Furthermore, a fixed step size was used. Picking a “good” step length should improve the rate of convergence.

### **Forcing the Hessian to be positive definite**

The general procedure is to modify the Hessian as follows,

$$A(\mathbf{x}) = [H(\mathbf{x}) + \beta \mathbf{I}] \quad (18)$$

Where  $\mathbf{I}$  is the identity matrix and  $\beta$  is a positive constant large enough to make  $H(\mathbf{x})$  positive definite when it is not. Levenberg and Marquardt first proposed algorithms of this type. The relatively straightforward procedure is summarised in Figure (7), where it is assumed that  $\|J(\mathbf{x})\| \leq \varepsilon_5$  on the 1<sup>st</sup> iteration,



**Figure (7) Marquardt’s algorithm (for minimisation).**

The basic idea is to start with a direction that approximates steepest descent and as the number of iterations increase gradually shift the emphasis to a Newton search by using Hessian information. It is assumed that after a number of iterations the algorithm should be close to the minimum and therefore the Hessian will be positive definite). This idea is to reduce the number of iterations required to find the minimum of a function.

Consider one iteration of the Marquardt algorithm used to optimise the function  $y = \sin(x_1) * \cos(x_2)$  starting at  $\mathbf{x}^0 = [1 \ 0.5]$  with  $\beta^0 = 1000$ . The search direction is,

$$\mathbf{s}^0 = -A(\mathbf{x}^0)^{-1}J(\mathbf{x}^0) \text{ where } J(\mathbf{x}^0) = \begin{bmatrix} 0.4742 \\ -0.4034 \end{bmatrix} \text{ and } A(\mathbf{x}^0) = \begin{bmatrix} 999.2615 & -0.259 \\ -0.259 & 999.2615 \end{bmatrix}$$

The two eigenvalues of  $A(\mathbf{x}^0)$  are 999.5206 and 999.0025 (i.e. they are both positive indicating that the algorithm will move towards a minimum) and the search direction is,

$$\mathbf{s}^0 = 1e-3 \begin{bmatrix} -0.4744 \\ -0.4036 \end{bmatrix}$$

**Note,**

- (1) The direction of search is similar to that of the standard steepest descent algorithm.
- (2) Using a fixed  $\alpha$ , i.e.  $\alpha = 0.1$  etc. will make only a small increment to  $\mathbf{x}$ . It would appear (from direct comparison to the use of the steepest descent algorithm on this example) that a larger value of  $\alpha$  could be tolerated. Systematic 1-dimensional optimisers are used to conduct a ‘line search’ for the optimal value of  $\alpha$ .

### ***Line search when optimising functions of more than one variable***

The methodology is to construct an ‘artificial’ function of one variable. At iteration  $k$ , it is known that,  $f(\mathbf{x}^{k+1}) = f(\mathbf{x}^k + \alpha \mathbf{s}^k)$ . The value of  $\mathbf{x}^k$  is known and the value of  $\mathbf{s}^k$  is assumed fixed. The objective of the line search is to optimise  $f(\mathbf{x}^k + \alpha \mathbf{s}^k)$  for  $\alpha$ , i.e. we minimise  $f(\alpha)$ .

A line search proceeds by iteration. Numerical recipes (Press et al, 1988) recommend the following methods,

- (1) Brent’s method when derivative information is not available.
- (2) Newton-Raphson (with book-keeping on the bounds) when derivative information is available.

Both methods require the minimum to be “bracketed”.

### **Bracketing**

A simple procedure to bracket a minimum is to “go down-hill” taking steps of increasing size (often,  $h$ ,  $2h$ ,  $4h$ , etc. where  $h$  is the step length) until the function value increases. The initial point and final point (e.g. at  $mh$ ) are then two positions in the one-dimensional space that ‘bracket’ a minimum.

### **Brents method.**

This combines bracketing, bisection and quadratic interpolation to find the minimum of a one-dimensional function. Quadratic interpolation involves a data fit to the *univariate* function of the form,

$$f(\alpha) = a\alpha^2 + b\alpha + c \quad (19)$$

The coefficients (a, b and c) are found from the solution of three simultaneous equations, e.g. via a least squares procedure. The values of  $\alpha$  and corresponding  $f(\alpha)$  are the two extremes of the bracket and the mid-point of the interval. Once the quadratic model has been parameterised the minimum is found by differentiation and setting to zero to give,

$$\alpha^* = \frac{-b}{2a} \quad (20)$$

If the solution does not lie within the initial bracket, or the function is not decreasing at a fast enough rate, the bisection method is used.

### Newton's (or Newton-Raphson) method

The necessary condition for  $f(\alpha)$  to have a minimum is that  $f'(\alpha) = 0$ . Suppose  $f(\alpha)$  is approximated by a quadratic function at  $\alpha_i$  (the subscript 'i' is used to avoid confusion between the iterations for the line search and the 'global' iteration of the multi-dimensional optimiser),

$$f(\alpha) \approx f(\alpha_i) + f'(\alpha_i)(\alpha - \alpha_i) + \frac{1}{2}f''(\alpha_i)(\alpha - \alpha_i)^2 \quad (21)$$

A stationary point of this quadratic function is given by  $f'(\alpha) = 0$ , i.e.,

$$f'(\alpha_i) + f''(\alpha_i)(\alpha - \alpha_i) = 0 \quad (22)$$

Re-arranging gives (as with the multi-dimensional algorithm the subscript 'i+1' has been added to indicate that for general functions equation 23 is a recursive formula),

$$\alpha_{i+1} = \alpha_i - \frac{f'(\alpha_i)}{f''(\alpha_i)} \quad (23)$$

As with the multi-dimensional counter-part, the advantage of Newton's method is that for a quadratic function the minimum (or maximum) will be obtained in a single iteration. The disadvantages of the technique are that the first and second derivatives must be calculated and the method can be slow, especially if the second derivative is small.

If the solution does not lie within the initial bracket or the function is not decreasing at a fast enough rate the bisection method<sup>2</sup> is used.

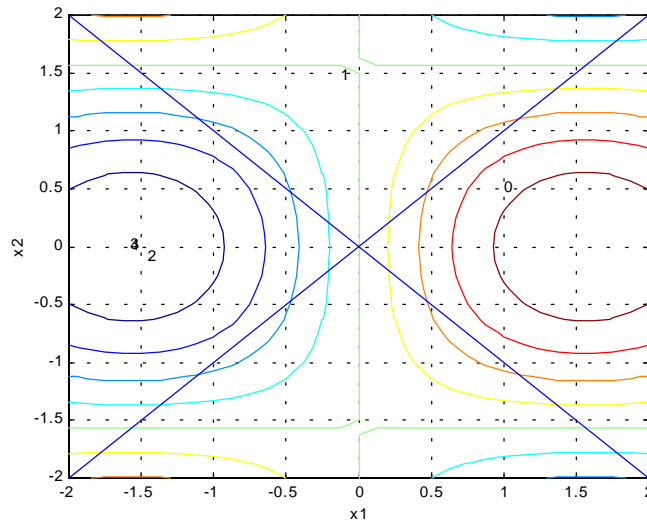
---

<sup>2</sup> A basic methodology here would be to half the interval given by the bracketing procedure, locate the section where the minimum lies and then repeat the procedure until a minimum point is found.

### Worked example

Optimise  $f(\mathbf{x}) = \sin(x_1)\cos(x_2)$  starting at  $\mathbf{x}^0 = [1 \ 0.5]$  using the Marquardt algorithm and a line search to determine the optimal values of  $\alpha$ .

Figure (8) shows the trajectory of the Marquardt algorithm initiated at  $\mathbf{x} = [1 \ 0.5]$ . The termination criteria had a tolerance of  $\epsilon_5 = 1e^{-4}$ . The numbers mark the point in the 2-dimensional space, which represent the new  $\mathbf{x}$  after the iteration. The total number of iterations taken was 4.



**Figure (8). Optimisation of the function  $y = \sin(x_1) * \cos(x_2)$  using Marquardt's algorithm initiated at  $\mathbf{x} = [1 \ 0.5]$  with a line search for  $\alpha$ .**

Table 2 lists the values of  $f(\mathbf{x})$ ,  $\mathbf{x}$ ,  $\mathbf{J}(\mathbf{x})$ ,  $\mathbf{A}(\mathbf{x})^{-1}$  and the step size  $\alpha$  for each stage of the minimisation.

k	$f(\mathbf{x}^k)$	$\mathbf{x}^{k+1}$	$\mathbf{J}(\mathbf{x}^k)$	$\mathbf{A}(\mathbf{x}^k)^{-1}$	$\alpha$
0	0.7385	$\begin{bmatrix} -0.1291 \\ 1.4606 \end{bmatrix}$	$\begin{bmatrix} 0.4742 \\ -0.4034 \end{bmatrix}$	$\begin{bmatrix} 999.2615 & -0.259 \\ -0.259 & 999.2615 \end{bmatrix}$	2380
1	-0.0142	$\begin{bmatrix} -1.4583 \\ -0.0963 \end{bmatrix}$	$\begin{bmatrix} 0.1091 \\ 0.1280 \end{bmatrix}$	$\begin{bmatrix} 250.0142 & -0.9857 \\ -0.9857 & 250.0142 \end{bmatrix}$	3032
2	-0.9891	$\begin{bmatrix} -1.5710 \\ 0.00 \end{bmatrix}$	$\begin{bmatrix} 0.118 \\ -0.0946 \end{bmatrix}$	$\begin{bmatrix} 63.4891 & 0.0108 \\ 0.0108 & 63.4891 \end{bmatrix}$	64
3	-1.0	$\begin{bmatrix} -1.57 \\ 0.00 \end{bmatrix}$	$\begin{bmatrix} 0.00 \\ 0.00 \end{bmatrix}$	$\begin{bmatrix} 16.625 & 0.0 \\ 0.0 & 16.625 \end{bmatrix}$	17

**Table 2. Optimisation of the function  $y = \sin(x_1) * \cos(x_2)$  using Marquardt's algorithm initiated at  $\mathbf{x} = [1 \ 0.5]$ .**

**Note,**

(1) The line search improves the rate of convergence of the algorithm.

***Quasi-Newton methods***

The basic idea behind Quasi-Newton methods is to approximate the inverse of the Hessian using only first order partial derivatives. Various methods have been proposed in the literature and these include Broyden's update, the Davidon-Fletcher-Powell (DFP) update, and the Broyden-Fletcher-Goldfarb-Shanno (BFGS). Of the three, BFGS is probably one of the more popular and more successful relationships.

The various algorithms are developed based on the gradient of a quadratic function. Recall that from the development of Newton's algorithm we have,

$$\mathbf{J}(\mathbf{x}^k) = -\mathbf{H}(\mathbf{x} - \mathbf{x}^k) \quad (24)$$

where  $\mathbf{H}$  is used to denote the Hessian rather than the more general notation  $\mathbf{H}(\mathbf{x}^k)$  as with a quadratic this is a constant matrix. Suppose equation (24) is linearised (around the point  $\mathbf{x}^0$ ). Then,

$$\mathbf{J}(\mathbf{x}) \approx \mathbf{J}(\mathbf{x}^0) + \mathbf{H}(\mathbf{x} - \mathbf{x}^0) \quad (25)$$

Approximate  $\mathbf{H}$  at iteration  $k$  by  $\mathbf{A}^k$ . The information used to construct  $\mathbf{A}^k$  originates from gradient information at  $\mathbf{J}(\mathbf{x}^{k-1})$  and  $\mathbf{J}(\mathbf{x}^k)$  as well as the points at  $\mathbf{x}^{k-1}$  and  $\mathbf{x}^k$ . Thus,

$$\mathbf{J}(\mathbf{x}^{k-1}) \approx \mathbf{J}(\mathbf{x}^0) + \mathbf{H}(\mathbf{x}^{k-1} - \mathbf{x}^0) \quad (26)$$

$$\mathbf{J}(\mathbf{x}^k) \approx \mathbf{J}(\mathbf{x}^0) + \mathbf{H}(\mathbf{x}^k - \mathbf{x}^0) \quad (27)$$

Subtracting (26) from (27) gives (on substitution of  $\mathbf{A}^k$  for  $\mathbf{H}$ ),

$$\mathbf{J}(\mathbf{x}^k) - \mathbf{J}(\mathbf{x}^{k-1}) = \mathbf{A}^k(\mathbf{x}^k - \mathbf{x}^{k-1})$$

Defining,  $\mathbf{g}^k = \mathbf{J}(\mathbf{x}^k) - \mathbf{J}(\mathbf{x}^{k-1})$  and  $\mathbf{d}^k = \mathbf{x}^k - \mathbf{x}^{k-1}$  gives,

$$\mathbf{g}^k = \mathbf{A}^k \mathbf{d}^k \text{ or } \mathbf{d}^k = (\mathbf{A}^k)^{-1} \mathbf{g}^k = \mathbf{B}^k \mathbf{g}^k \quad (28)$$

$\mathbf{B}^k$  is an approximation to the inverse of the Hessian.  $\mathbf{d}^k$  and  $\mathbf{g}^k$  are 'n' dimensional vectors while  $\mathbf{B}^k$  is an 'n x n' dimensional matrix. The objective is to choose  $\mathbf{B}^k$  so that it is as close as possible to  $\mathbf{H}^{-1}$ .

The general update formula is,

$$\mathbf{B}^k = \mathbf{B}^{k-1} + \Phi \quad (29)$$

To initiate the recursion  $\mathbf{B}^0 = \mathbf{I}$  (the identity matrix) and the choice of  $\Phi$  determines the procedure used.

### Broyden's update

$\Phi = \mathbf{czz}^T$  where  $c = \text{scalar}$  and  $\mathbf{z} = [z_1, \dots, z_n]^T$

$$\mathbf{B}^k = \mathbf{B}^{k-1} + \Phi = \mathbf{B}^{k-1} + \mathbf{czz}^T \quad (35)$$

But,

$$\mathbf{d}^k = \mathbf{B}^k \mathbf{g}^k = [\mathbf{B}^{k-1} + \mathbf{czz}^T] \mathbf{g}^k \quad (36)$$

or,

$$\mathbf{d}^k = \mathbf{B}^{k-1} \mathbf{g}^k + \mathbf{cz}(\mathbf{z}^T \mathbf{g}^k) \quad (37)$$

Now  $\mathbf{z}^T \mathbf{g}^k$  is a scalar, therefore equation (37) may be re-arranged as,

$$\mathbf{cz} = (\mathbf{d}^k - \mathbf{B}^{k-1} \mathbf{g}^k) / (\mathbf{z}^T \mathbf{g}^k) \quad (38)$$

Taking  $c = 1/(\mathbf{z}^T \mathbf{g}^k)$  gives  $\mathbf{z} = (\mathbf{d}^k - \mathbf{B}^{k-1} \mathbf{g}^k)$

In other words, the update formula is,

$$\mathbf{B}^k = \mathbf{B}^{k-1} + [(\mathbf{d}^k - \mathbf{B}^{k-1} \mathbf{g}^k) (\mathbf{d}^k - \mathbf{B}^{k-1} \mathbf{g}^k)^T] / (\mathbf{z}^T \mathbf{g}^k) \quad (39)$$

Broyden's formula can be non-robust, i.e not work for non-quadratic functions.

### Davidon-Fletcher-Powell (DFP) update, $\Phi = c_1 \mathbf{z}_1 \mathbf{z}_1^T + c_2 \mathbf{z}_2 \mathbf{z}_2^T$

Following the procedure outlined above,

$$\mathbf{d}^k = \mathbf{B}^{k-1} \mathbf{g}^k + c_1 \mathbf{z}_1 (\mathbf{z}_1^T \mathbf{g}^k) + c_2 \mathbf{z}_2 (\mathbf{z}_2^T \mathbf{g}^k) \quad (40)$$

The following (non-unique) choices may be made:

$$c_1 = 1/(\mathbf{z}_1^T \mathbf{g}^k); c_2 = -1/(\mathbf{z}_2^T \mathbf{g}^k); \mathbf{z}_1 = \mathbf{d}^k; \mathbf{z}_2 = \mathbf{B}^k \mathbf{g}^k \quad (41)$$

This is robust when used to minimise general non-linear functions (advanced optimisation text-books should be consulted as to the reasons why).

The update expressions are known as inverse Hessian update formulas – as they approximate the inverse of the Hessian. Following exactly the same procedures as detailed above it is possible to derive direct update formulas – ones that approximate the Hessian. Following the DFP procedure this will lead to the Broyden-Fletcher-Goldfarb-Shanno (BFGS) formula.

### ***Matlab's optimisation toolbox***

Any realistic example should be tackled using a computer-based algorithm. Matlab's Optimisation toolbox provides robust software that may be used for this purpose. In Matlab the unconstrained multi-dimensional function optimiser is called `fminu`. It is a quasi-newton method that uses the BFGS formula for updating the approximation to the Hessian. The default line search algorithm is a mixed quadratic and cubic interpolation algorithm. There are various options that may be specified and these are listed in Table x (a \* indicates the default setting). The calling syntax and use of the minimisation routine will be explained in laboratory session # 1.

Option	Setting	Description
1	0*, 1	No output, table of results
2	1e-4*	Termination criteria for $\mathbf{x}$ (all $x_i$ must meet this criteria)
3	1e-4*	Termination criteria for $f(\mathbf{x})$
6	0*, 1, 2	DFGS, DFP, Steepest descent
7	0*, 1	Mixed quadratic / cubic polynomial, cubic polynomial
14	100n*	Max. no. of iterations ( $n$ = the no. of independent variables)

**Table 3. Various options available for use with function 'fminu'.**

