Quasi Newton Optimization
via Levenber Marquardt and Camera Triangulation

Henrik Aanæs Vesselin Perfanov

May 17, 2008

The aim of this exercise is to illustrate the use of Quasi Newton optimization schemes, particularly the very popular Marquardt or Levenberg-Marquardt algorithm, [2, 3]. The task to which this algorithm is to be applied is that of camera triangulation, i.e. given n-cameras, which observe a point, estimate the 3D location of that point. This is a fundamental problem in computer vision and photogrammetry, and has a complexity which should allow you to complete this exercise within the given time frame.

This exercise is run in MatLab. If you are not acquainted with this tool, an introductory primer can be found at:

- `http://www2.imm.dtu.dk/~jmc/02501/exercises/introduction/introduction.pdf`
- `http://www2.imm.dtu.dk/~jmc/02501/exercises/introduction2/introduction2.pdf`

1 Getting Started

In order to get started you are to download a version of the Marquardt algorithm from `http://www2.imm.dtu.dk/~hbn/immoptibox/` and unzip it into the directory in which you wish to work.

In order to illustrate how this algorithm works please run the script `TrialRun.m`. This script applies the Marquardt method to the well-known (in optimization circles) Rosenbrock function.

Task: *Insert comments into the script, such that you can convince yourself that you know what is going on. Try typing* `help marquardt`.

There is also an out-commented section, which calculates the numerical gradient and compares it with the analytical. This is an important way to validate your objective functions, because experience shows, that errors often occur in computing the analytical gradient. Try it out and comment.

2 Generating an Initial Guess

Since the Quasi Newton methods, and the Marquardt method in particular, work via an iterative scheme, a starting guess or point is required. The performance of the algorithm typically depends highly on this starting guess, both in regards to speed of convergence, and more importantly if it finds the global or a local minimum.

A good way for finding a starting point is via a so called algebraic method [1], i.e. a method that minimizes a non-interpretable error measure, but is linear (meaning easily solvable). To derive this method, assume a perspective camera model, where the cameras are represented by 3 by 4 matrix. Denote the cameras by their columns, i.e.

$$\mathbf{C}_i = \begin{bmatrix} C_{i1}^T \\ C_{i2}^T \\ C_{i3}^T \end{bmatrix} \ , \tag{1}$$

and the 3D point by a homogeneous vector of length 4, denoted Q. The 2D coordinate of Q projected into camera i is then given in homogeneous coordinates, by:

$$q_i = \begin{bmatrix} s_i x_i \\ s_i y_i \\ s_i \end{bmatrix} = \mathbf{C}_i Q \ . \tag{2}$$

Thus the x-coordinate is given by:

$$x_i = \frac{C_{i1}^T Q}{C_{i3}^T Q} \ , \tag{3}$$

which can be transformed into a linear constraint in Q (which is our unknown) by:

$$x_i = \frac{C_{i1}^T Q}{C_{i3}^T Q} \Rightarrow x_i C_{i3}^T Q = C_{i1}^T Q \Rightarrow x_i C_{i3}^T Q - C_{i1}^T Q = 0 \Rightarrow (x_i C_{i3}^T - C_{i1}^T) Q = 0 \ . \tag{4}$$

A similar constraints can be calculated for y_i, yielding two times the number of views linear constraints on Q. These linear constraints can be arranged in a matrix, B, and Q can be found via, SVD in MatLab:

```
...
B(cObs*2,:)=qs{cObs}(2)*Cams{cObs}(3,:)-Cams{cObs}(2,:);
...
[u,s,v]=svd(B);
Q0=v(:,end)/v(end,end);
```

Please refer to Section 4, if you want to see why this method is 'faulty'.

Task: *Write MatLab functionality that calculates an initial estimate via the above described method.*

The data you should use is located in CamData.mat, and can be retrieved via `load CamData`. It consists of three cameras, `Cams`, with corresponding three 2D coordinates, `qs`.

3 Non-Linear Optimization

To run the actual Marquardt algorithm or non-linear optimization, you have to supply an objective function as described in the help to the supplied algorithm. This function should supply the observation errors f_i for a given Q:

$$f_{2i-1} = \frac{C_{i1}^T Q}{C_{i3}^T Q} - \frac{q_x}{q_s} \tag{5}$$

$$f_{2i} = \frac{C_{i2}^T Q}{C_{i3}^T Q} - \frac{q_y}{q_s} \ , \tag{6}$$

and their derivatives wrt. Q, J_{ij}:

$$J_{2i,j} = \frac{C_{i2,j} C_{i3}^T Q - C_{i3,j} C_{i2}^T Q}{(C_{i3}^T Q)^2}$$

Task: *Construct the objective function and test it by comparing your analytical derivatives with the corresponding numerical derivatives. What is the effect of setting* `delta`?

Task: *Run the Marquardt algorithm with your start guess.*

4 Where did the Algebraic Method Go Wrong?

To see where the 'fault' is in the algebraic method, consider once more (3):

$$x_i = \frac{C_{i1}^T Q}{C_{i3}^T Q} \ ,$$

which should have been

$$x_i = \frac{C_{i1}^T Q}{C_{i3}^T Q} + \epsilon_i \ , \tag{7}$$

since there is a measurement error or noise. A more correct algorithm would solve the problem of:

$$\min_Q \sum_i ||\epsilon_i||_? \ ,$$

where $||\cdot||_?$ denotes some norm, usually the two norm squared, i.e. $||\cdot||_2^2$. Redoing the calculations of (2) with this more correct observation model, (7), would then result in

$$(x_i C_{i3}^T - C_{i1}^T) Q = \epsilon (C_{i3}^T Q) \ . \tag{8}$$

So when minimizing

$$\min_Q || \sum_i (x_i C_{i3}^T - C_{i1}^T) Q + (y_i C_{i3}^T - C_{i2}^T) Q ||_2^2 \ ,$$

as is done with the linear method, as described above, the observation errors ϵ_i are 'erroneously' weighted by $C_{i3}^T Q$. Where $C_{i3}^T Q$ is closely related to the distance of the camera $\mathbf{C}_i$ to the 3d point Q.

References

[1] R. I. Hartley and A. Zisserman. *Multiple View Geometry – 2nd edition.* Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 2003.

[2] K. Levenberg. A method for the solution of certain problems in least-squares. *Quart. J. of Appl. Math.*, 12:164–168, 1944.

[3] D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.