A Memo on How to Use the Levenberg-Marquardt Algorithm for Refining Camera Calibration Parameters

Pradit Mittrapiyanuruk
Robot Vision Laboratory, Purdue University, West Lafayette, IN, USA
mitrapiy@purdue.edu

This short note briefly explains how to implement the Levenberg-Marquardt (LM) algorithm in MATLAB for the minimization of the geometric distance given by Equation (10) of the Zhang's report. For students who are not familiar with the LM method, we suggest you visit the Wikipedia page

http://en.wikipedia.org/wiki/Levenberg-Marquardt_algorithm

For a more elaborate discussion of the LM method, the following references may be useful
- Appendix 6 in the Hartley & Zisserman's Book
- Section 9.4 in the Chong & Zak's Book (the official textbook for ECE580)
- K. Madsen et al, "Methods for non-linear least squares problems", IMM Lecture Note 2004, available on the web at
  http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3215/pdf/imm3215.pdf

In the rest of this note, I will first demonstrate how to use the LM method to solve a simple curve fitting problem and then I will give some clues that may help you to apply the LM algorithm for the refinement of the camera calibration parameters in Homework 5.

Example: Curve fitting with the LM algorithm

Problem: Fit a set of point data $\{x_i,y_i\}$ (x_i=0, 0.1, …, 2*PI) with the function[*]

f(x) = a*cos(bx)+b*sin(ax)

where a and b are the parameters that we want to estimate by the LM method. The problem of finding the optimum can be stated as follows

$$\arg\min_{a,b} \quad \mathbf{d}^T\mathbf{d}$$

$$where \quad \mathbf{d} = [d_1\, d_2 ......d_n]^T \; and \quad d_i = [y_i - f(x_i;a,b)]$$

[*] This example, taken from Wikipedia, is interesting because the complexity of the fitted function would preclude a linear least square solution to the problem.

The LM algorithm requires that you supply it with the Jacobian of the vector **d**. The Jacobian is merely a matrix representation of all the first derivatives of the components of the vector. For the example, the Jacobian of the vector **d** is given by the matrix

$$\mathbf{J} = \begin{bmatrix} \frac{\partial d_1}{\partial a} & \frac{\partial d_1}{\partial b} \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \frac{\partial d_n}{\partial a} & \frac{\partial d_n}{\partial b} \end{bmatrix}$$

The above minimization can be carried out with the following MATLAB steps:

1. Using the MATLAB symbolic toolbox, find the analytic form of the Jacobian of one row of **d** with respect to the parameters a and b

```
syms a b x y real;
f=( a * cos(b*x) + b * sin(a*x))
d=y-f;
Jsym=jacobian(d,[a b]);
```

This MATLAB code returns the following symbolic expression

```
Jsym =[ -cos(b*x)-b*cos(a*x)*x,  a*sin(b*x)*x-sin(a*x)]
```

Note that in the above MATLAB script the variable *Jsym* is a single row of the Jacobian matrix **J** shown above. That is

$$Jsym = \begin{bmatrix} \frac{\partial d_i}{\partial a} & \frac{\partial d_i}{\partial b} \end{bmatrix}$$

2. Generate the synthetic data from the curve function with some additional noise

```
a=100;
b=102;
x=[0:0.1:2*pi]';
y = a * cos(b*x) + b * sin(a*x);
% add random noise
y_input = y + 5*rand(length(x),1);
```

3. The main code for the LM algorithm for estimating a and b from the above data

```
% initial guess for the parameters
a0=100.5; b0=102.5;
```

```
y_init = a0 * cos(b0*x) + b0 * sin(a0*x);
Ndata=length(y_input);
Nparams=2;                    % a and b are the parameters to be estimated
n_iters=100;                  % set # of iterations for the LM
lamda=0.01;                   % set an initial value of the damping factor for the LM
updateJ=1;
a_est=a0;
b_est=b0;
for it=1:n_iters
    if updateJ==1
        % Evaluate the Jacobian matrix at the current parameters (a_est, b_est)
        J=zeros(Ndata,Nparams);
        for i=1:length(x)
          J(i,:)=[-cos(b_est*x(i))-(b_est*cos(a_est*x(i))*x(i))   (a_est*sin(b_est*x(i))*x(i))-sin(a_est*x(i))];
        End
        % Evaluate the distance error at the current parameters
        y_est = a_est * cos(b_est*x) + b_est * sin(a_est*x);
        d=y_input-y_est;
        % compute the approximated Hessian matrix, J' is the transpose of J
        H=J'*J;
        if it==1              % the first iteration : compute the total error
            e=dot(d,d);
        end
    end

    % Apply the damping factor to the Hessian matrix
    H_lm=H+(lamda*eye(Nparams,Nparams));

    % Compute the updated parameters
    dp=-inv(H_lm)*(J'*d(:));
    a_lm=a_est+dp(1);
    b_lm=b_est+dp(2);

    % Evaluate the total distance error at the updated parameters
    y_est_lm = a_lm * cos(b_lm*x) + b_lm * sin(a_lm*x);
    d_lm=y_input-y_est_lm;
    e_lm=dot(d_lm,d_lm);

   % If the total distance error of the updated parameters is less than the previous one
   % then makes the updated parameters to be the current parameters
   % and decreases the value of the damping factor
    if e_lm<e
        lamda=lamda/10;
        a_est=a_lm;
        b_est=b_lm;
        e=e_lm;
        disp(e);
        updateJ=1;
    else % otherwise increases the value of the damping factor
        updateJ=0;
        lamda=lamda*10;
    end
end
```

4. Results

The LM algorithm requires an initial guess for the parameters to be estimated. We chose the value a=100.5 and b=102.5 for the initial guess. (For the camera calibration parameter refinement problem, the initial guess is supplied by the linear least-squares solution.) The plot of the generated curve with the initial parameters vis-à-vis the input data is shown in Figure 1. After 100 iterations of the LM algorithm, the refined parameters a and b returned by the LM algorithm are 99.999 and 102.0013, respectively. The plot of the generated curve with the refined parameters vis-à-vis the input data is shown in Figure 2.

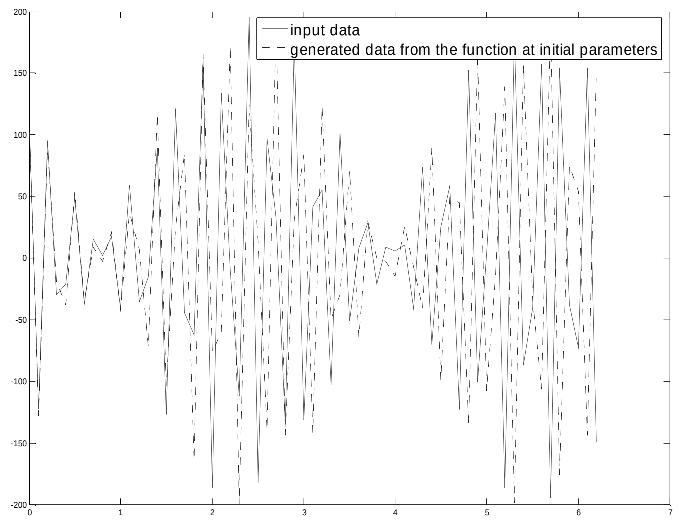

Figure 1. Plot of the input data and the generated data from the function at the initial parameters

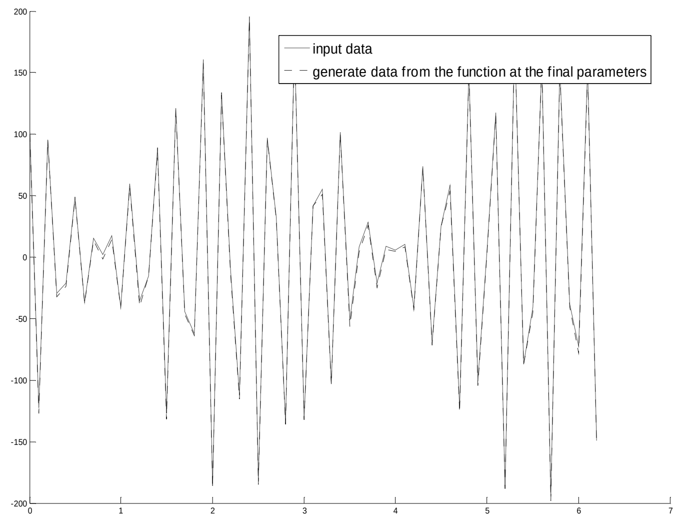

Figure 2. Plot of the input data and the generated data from the function at the final parameters after running 100 iterations of the LM algorithm.

To apply the LM method for the minimization required in Equation 10 of Zhang's report, I suggest the following:

To simplify the explanation, let's reduce Equation 10 to

$$\sum_{j=1}^{m} \left\| \mathbf{x}_{\mathrm{j}} - \hat{\mathbf{x}}_j(\mathbf{K},\mathbf{R},\mathbf{t},\mathbf{x}_{\mathrm{M},j}) \right\|^2$$

where

- $\mathbf{x}_{\mathrm{j}}$ is the jth point extracted in the image captured from the camera at the position that is used to define the extrinsic parameters.
- $\mathbf{x}_{\mathrm{M},j}$ is the jth point on the model plane.
- $\hat{\mathbf{x}}_j$ is the projection of the jth model point into the camera image.
- $\mathbf{K}$ is the intrinsic camera parameter.
- {$\mathbf{R}$, $\mathbf{t}$} are the rotation and translation of the camera position that is used to define the extrinsic parameters.

Note that in the above equation we will refine the intrinsic parameters and the extrinsic parameters for just one of the images you recorded (as opposed to all the images in Equation 10 of Zhang's report).

In a manner similar to the previous example, the minimization of the geometric distance given by the above equation can be stated as follows

$$\underset{\mathbf{K},\mathbf{R},\mathbf{t}}{\arg\min} \quad \mathbf{d}^T\mathbf{d}$$

$$\begin{aligned} \textit{where} \quad \mathbf{d} &= [d_{1x}\, d_{1y}\, d_{2x}\, d_{2y} \ldots\ldots d_{mx}\, d_{my}]^T \\ d_{jx} &= [x_{ju} - \hat{x}_{ju}(\mathbf{K},\mathbf{R},\mathbf{t},\mathbf{x}_{\mathrm{M},\mathbf{j}})] \\ d_{jy} &= [x_{jv} - \hat{x}_{jv}(\mathbf{K},\mathbf{R},\mathbf{t},\mathbf{x}_{\mathrm{M},\mathbf{j}})] \end{aligned}$$

x_{ju} , x_{jv} are the x and y positions of the jth point in the recorded image, and
$\hat{x}_{ju}$, $\hat{x}_{jv}$ are the x and y positions of the projection of the corresponding model point in the image

The minimization shown above is directly in terms of the rotation matrix **R** that has 9 elements. But we do not want to optimize with respect to all nine elements because they are not independent. Recall that **R** has only 3 DOF. So we need a way to express **R** in terms of only 3 parameters. For that purpose we will use the Rodrigues formula that allows us to represent **R** using a 3-vector **w**=$[w_x\ w_y\ w_z]^T$(see Wikipedia and Appendix A4.3.2 of the text for the Rodrigues representation).

The Rodrigues form represents a rotation by a vector whose direction stands for the axis of rotation and whose magnitude is the angle through which the object is rotated clockwise around the axis. That is, if the 3-vector $\mathbf{w}=[w_x\ w_y\ w_z]^T$ represents an object rotation, the axis of rotation is given by

$$\frac{\mathbf{w}}{\|\mathbf{w}\|}$$

and the angle

$$\phi = \|\mathbf{w}\|$$

How to get the rotation matrix **R** from the 3-vector $\mathbf{w}=[w_x\ w_y\ w_z]^T$? The question is answered on page 584 of the text. If we use the following notation to convert a 3-vector $\mathbf{w}=[w_x\ w_y\ w_z]^T$ into the special 3x3 matrix form shown below:

$$[\mathbf{w}]_x = \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix}$$

then one can show that

$$\mathbf{R} = e^{[\mathbf{w}]_x} = I + \frac{\sin(\|\mathbf{w}\|)}{\|\mathbf{w}\|}[\mathbf{w}]_x + \frac{(1-\cos(\|\mathbf{w}\|))}{\|\mathbf{w}\|^2}([\mathbf{w}]_x)^2$$

To go in the other direction, that is given **R**, how do we get the 3-vector representation $\mathbf{w}=[w_x\ w_y\ w_z]^T$, see Equation (A4.10) on Page 584 of the text. To display a more convenient form here, if the rotation matrix $\mathbf{R} \neq \mathbf{I}$ is given as

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix},$$

the corresponding $\mathbf{w}=[w_x\ w_y\ w_z]^T$ is given by

$$\|\mathbf{w}\| = \cos^{-1}\left(\frac{trace(\mathbf{R})-1}{2}\right), \qquad \frac{\mathbf{w}}{\|\mathbf{w}\|} = \frac{1}{2\sin(\|\mathbf{w}\|)}\begin{bmatrix} r_{32}-r_{23} \\ r_{13}-r_{31} \\ r_{21}-r_{12} \end{bmatrix}$$

Now that we have shown how to express the rotation matrix as a 3-vector, we are ready to discuss the MATLAB implementation of the needed minimization. Do the following:

1. Using the MATLAB symbolic toolbox, find the analytic form of the Jacobians of d_{ix} and d_{iy} with respect to the parameters that we want to refine i.e. [α, β, γ, u_0, v_0, w_x, w_y, w_z, t_x, t_y and t_z]. Note the difference in the notation here and that used in class: α here corresponds to α_x in class; β to α_y; u_0 to x_0; v_0 to y_0. The MATLAB code for computing the analytic Jacobians can be shown as follows

```
% Note that: au = α, av=β, sk= γ in Zhang's report
 syms u0 v0 au av sk real
 syms tx ty tz wx wy wz real
 syms X Y um vm real

% the intrinsic parameter matrix
 K=[au sk u0;
     0  av v0;
     0  0   1];

% Expression for the rotation matrix based on the Rodrigues formula
 theta=sqrt(wx^2+wy^2+wz^2);
 omega=[0 -wz wy;
             wz 0  -wx;
            -wy wx 0;];
 R = eye(3) + (sin(theta)/theta)*omega + ((1-cos(theta))/theta^2)*(omega*omega);

% Expression for the translation vector
 t=[tx;ty;tz];

% perspective projection of the model point (X,Y)
 uvs=K*[R(:,1) R(:,2) t]*[X; Y; 1];
 u=uvs(1)/uvs(3);
 v=uvs(2)/uvs(3);

 % calculate the geometric distance in x and y direction
 % um,vm =the x and y positions of an extracted corner
 % u,v = the x and y positions of the projection of the corresponding model point
 dx=um-u;
 dy=vm-v;

 % Evaluate the symbolic expression of the Jacobian w.r.t. the estimated parameters
 Jx=jacobian(dx,[au,av,u0,v0,sk,wx wy wz tx ty tz]);
 Jy=jacobian(dy,[au,av,u0,v0,sk,wx wy wz tx ty tz]);
```

Don't be surprised if the symbolic expressions for the Jacobians Jx and Jy are long. (For people who want to use the derived expressions in a C program, you can use the MATLAB function *ccode* which will return a fragment of C code that evaluates the corresponding symbolic expressions.)

2. The main code for the LM algorithm for refining the parameters can be shown as follows

```
% R0, t0 = initial extrinsic parameters obtained from the solution in Section 3
tx=t0(1);
ty=t0(2);
tz=t0(3);

% convert the 3x3 rotation matrix into 3-vector w=[wx wy wz] of the Rodigrues representation
R=R0;
theta=acos((trace(R)-1)/2);
w=(theta/(2*sin(theta)))*[R(3,2)-R(2,3); R(1,3)-R(3,1); R(2,1)-R(1,2)];
wx=w(1);
wy=w(2);
wz=w(3);


iters=200;                              % set the number of iterations for the LM algorithm
lamda=0.01;                             % initial the value of damping factor
updateJ=1;
Ndata=2*"number of corner points"
Nparams=11;
for it=1:iters
   if updateJ==1

     % create the intrinsic parameter matrix
      K=[au sk u0;
          0  av v0;
          0  0   1];

      % convert the 3-vector [wx wy wz] of the Rodigrues representation
      % into the 3x3 rotation matrix
      theta2=wx^2+wy^2+wz^2;
      theta=sqrt(theta2);
      omega=[0 -wz wy;
                 wz 0  -wx;
                 -wy wx 0;];
      R = eye(3) + (sin(theta)/theta)*omega + ((1-cos(theta))/theta^2)*(omega*omega);
      t=[tx;ty;tz];

      % Evaluate the Jacobian at the current parameter values
      % and the values of geometric distance
      J=zeros(Ndata,Nparams);
      d=zeros(Ndata,1);
      for i=1:size(xe,2)
         X=Xw(1,i);              % (X,Y) are the coordinates of the ith model point
         Y=Xw(2,i);
         J(2*(i-1)+1,:)= "the value of Jx evaluated at the ith model point X,Y and the current parameters"
         J(2*(i-1)+2,:)= "the value of Jy evaluated at the ith model point X,Y and the current parameters"

         % perspective projection of the model point
        uvs=K*[R(:,1) R(:,2) t]*[X; Y; 1];
         up=uvs(1)/uvs(3);
         vp=uvs(2)/uvs(3);
         % compute the geometric distance in x & y directions
         % xe(1,i), xe(2,i) = the the x and y positions of the ith extracted corner.
         %  up,vp = the x and y positions of the projection of the corresponding model point
         d(2*(i-1)+1,1)=xe(1,i)-up;
         d(2*(i-1)+2,1)=xe(2,i)-vp;
      end

     % compute the approximated Hessian matrix
      H=J'*J;
      if it==1 % the first iteration : compute the initial total error
         e=dot(d,d);
         disp(e);
      end
   end

   % Apply the damping factor to the Hessian matrix
   H_lm=H+(lamda*eye(Nparams,Nparams));

   % Compute the updated parameters
   dp=-inv(H_lm)*(J'*d(:));
   au_lm=au+dp(1);
   av_lm=av+dp(2);
   u0_lm=u0+dp(3)
   v0_lm=v0+dp(4)
   sk_lm=sk+dp(5);
   wx_lm=wx+dp(6);
   wy_lm=wy+dp(7);
   wz_lm=wz+dp(8);
   tx_lm=tx+dp(9);
   ty_lm=ty+dp(10);
   tz_lm=tz+dp(11);

   % Evaluate the total geometric distance at the updated parameters
   K=[au_lm sk_lm u0_lm;
          0    av_lm v0_lm;
          0    0       1];
   omega=[0 -wz_lm wy_lm;
               wz_lm 0  -wx_lm;
               -wy_lm wx_lm 0;];
   theta2=wx_lm^2+wy_lm^2+wz_lm^2;
   theta=sqrt(theta2);
   R = eye(3) + (sin(theta)/theta)*omega + ((1-cos(theta))/theta^2)*(omega*omega);
   t=[tx_lm;ty_lm;tz_lm];
   d_lm=zeros(Ndata,1);
   for i=1:size(xe,2)
      X=Xw(1,i);
      Y=Xw(2,i);
      uvs=K*[R(:,1) R(:,2) t]*[X; Y; 1];
      up=uvs(1)/uvs(3);
      vp=uvs(2)/uvs(3);
      d_lm(2*(i-1)+1,1)=xe(1,i)-up;
```

```
        d_lm(2*(i-1)+2,1)=xe(2,i)-vp;
    end

    % e_lm is the error between the image coordinates and projective coordinates using
    % the updated parameters
    e_lm=dot(d_lm,d_lm);

   %If the total geometric distance of the updated parameters is less than the previous one
   % then makes the updated parameters to be the current parameters
   % and decreases the value of the damping factor
    if e_lm<e
        lamda=lamda/10;
        au=au_lm;
        av=av_lm;
        u0=u0_lm;
        v0=v0_lm;
        sk=sk_lm;
        wx=wx_lm;
        wy=wy_lm;
         wz=wz_lm;
        tx=tx_lm;
        ty=ty_lm;
        tz=tz_lm;
        e=e_lm;
        disp(e);
        updateJ=1;
    else
    % otherwise increase the value of the damping factor and try again
        updateJ=0;
        lamda=lamda*10;
    end
end
```

After the number of iterations expressed by the variable *iters*, the values of the refined parameters are stored in the variables *au, av, u0, v0, sk, wx, wy, wx, tx, ty,* and *tz.*