M. Balda **LMFnlsq2– Nonlinear least squares** 2013-04-15

Let us have a measured values (column vector) $\boldsymbol{y}$ dependent on operational variables $\boldsymbol{x}$. We try to make a regression of $\boldsymbol{y}(\boldsymbol{x})$ by some function $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{c})$, where $\boldsymbol{c}$ is a vector of unknown parameters of the model. The difference

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{c}) - \boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{r}(\boldsymbol{x}, \boldsymbol{c}) \tag{1}$$

is a column vector of so called residuals $\boldsymbol{r}(\boldsymbol{x},\boldsymbol{c})$, which should be zero vector, provided our model $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{c})$ is physically perfect and $\boldsymbol{y}(\boldsymbol{x})$ is also without any measurement deviations.

The method of least squares of residuals searches such vector $\boldsymbol{c}$, for which sum of squares $S = \boldsymbol{r}^T \boldsymbol{r}$ becomes minimum. It follows from the condition of minimum that

$$\partial S(\boldsymbol{c})/\partial \boldsymbol{c} = 2\, \boldsymbol{J}^T \boldsymbol{r} = 2\boldsymbol{v}, \tag{2}$$

where ∂ is a symbol for partial differential, $\boldsymbol{J}$ is a matrix of partial derivatives of S due to $\boldsymbol{c}$, so called Jacobi's matrix (Jacobian matrix), and $\boldsymbol{r}$ is a vector of residuals.

It is clear that $\boldsymbol{v}$, $\boldsymbol{r}$ and $\boldsymbol{J}$ are functions of unknown parameters $\boldsymbol{c}$. In general case, $\boldsymbol{c}$ may not be obtained in closed form, it is necessary to solve it in iterations. Let $\boldsymbol{c}$ in (k+1)st step of iteration has the simplest form

$$\boldsymbol{c}^{(k+1)} = \boldsymbol{c}^{(k)} + \Delta \boldsymbol{c}^{(k)}. \tag{3}$$

It is possible to assume that the vector of residuals in (k+1)st iteration, provided $\boldsymbol{r}$ is continuous in $\boldsymbol{c}$, will have the form given by Taylor's expansion

$$\boldsymbol{r}^{(k+1)} = \boldsymbol{r}^{(k)} + \partial \boldsymbol{r}^{(k)}/\partial \boldsymbol{c}^{(k)}\, \Delta \boldsymbol{c}^{(k)} + ... \tag{4}$$

After multiplication of the equation by a matrix $\boldsymbol{J}^{(k)^T}$ from left, we get the following equation:

$$\boldsymbol{J}^{(k)^T} \boldsymbol{r}^{(k+1)} = \boldsymbol{J}^{(k)^T} \boldsymbol{r}^{(k)} + \boldsymbol{J}^{(k)^T} \boldsymbol{J}^{(k)}\, \Delta \boldsymbol{c}^{(k)}, \tag{5}$$

which after introducing

$$\boldsymbol{A}^{(k)} = \boldsymbol{J}^{(k)^T} \boldsymbol{J}^{(k)} \tag{6}$$

takes the form

$$\boldsymbol{A}^{(k)}\, \Delta \boldsymbol{c}^{(k)} - \boldsymbol{J}^{(k)^T} \boldsymbol{r}^{(k+1)} = -\boldsymbol{v}^{(k)}. \tag{7}$$

This equation (7) is the starting point for a series of methods:

Newton-Raphson:

It assumes that norms or residual vectors are dropping so fast, that the second term in eqn. (7) may be neglected. Hence,

$$\boldsymbol{A}^{(k)}\, \Delta \boldsymbol{c}^{(k)} = -\boldsymbol{v}^{(k)} \tag{8}$$

may be used to obtain $\Delta \boldsymbol{c}^{(k)}$ and then

$$\boldsymbol{c}^{(k+1)} = \boldsymbol{c}^{(k)} + \Delta \boldsymbol{c}^{(k)}. \tag{9}$$

There are some modifications of the method, say that only $a\, \Delta \boldsymbol{c}^{(k)}$ is added, where $a <= 1$.

Levenberg-Marquardt:

There is an artificial assumption, that the second term in the equation (7) could be approximated by

$$\lambda^{(k)}\, \boldsymbol{D}\, \Delta \boldsymbol{c}^{(k)}, \tag{10}$$

where $\boldsymbol{D}$ is a suitable diagonal matrix of scales. It is often chosen as a unity matrix $\boldsymbol{I}$ or a diagonal of the matrix $\boldsymbol{A}_o$. The equation (7) is then transformed into

$$\boldsymbol{A}^{(k)}\, \Delta \boldsymbol{c}^{(k)} + \lambda^{(k)}\, \boldsymbol{D}\, \Delta \boldsymbol{c}^{(k)} = -\boldsymbol{v}^{(k)}. \tag{11}$$

The idea belongs to Levenberg. The strategy of modification improved Marquardt and later Fletcher made the superfinish of it.

The equation (11) may be converted into the form

$$(\boldsymbol{A}^{(k)} + \lambda^{(k)}\, \boldsymbol{D})\, \Delta \boldsymbol{c}^{(k)} = -\boldsymbol{v}^{(k)}. \tag{12}$$

If $\boldsymbol{D} = \mathrm{diag}(\boldsymbol{A}^{(k)})$, the diagonal of the system matrix, is strongly influenced by a scale parameter λ. The higher it is, the closer the result is to the stable solution of steepest descent. For $\lambda = 0$, the method approaches the method of Newton- Raphson, which is less stable and may diverge.

The strategy is based on a comparison of a forecast of the solution for the next iteration. If the forecast is close to the reality, the lambda may be lowered. If it is bad, λ should become higher in order to stabilize the process. And it is the role of some heuristic choice of authors in multipliers 2 or 10. They could be other.

The Levenberg-Marquardt method in Fletcher's modification [1] for solution of non-linear least squares problems has been implemented in MATLAB in a simplified version under the name `LMFsolve` some time ago (see [2]), and is widely used by the MATLAB community. The convergence and stability of the function has been strongly influenced both by the simplification of the code and a bug in application of analytical form of jacobian matrix. This has been a reason why a new version of the function `LMFnlsq2` has been built. It is almost unchanged transcription of the original Fletcher's FORTRAN code into MATLAB structures, but the initial part containing option settings, finite difference evaluation of jacobian matrix and printout module. The new function is stable and efficient.

Unconstrained optimization

A script named `LMFnlsq2test` is provided for testing `LMFnlsq2`. It covers both unconstrained and constrained minimization problem of Rosenbrock's function

$$f(\boldsymbol{x}) = 100\,(x_2 - x_1^2)^2 + (1 - x_1)^2 \tag{13}$$

as a sum of squares of residuals, $f(\boldsymbol{x}) = f_1^2(\boldsymbol{x}) + f_2^2(\boldsymbol{x})$, where $f_1(\boldsymbol{x}) = 10\,(x_2 - x_1^2)$ and $f_2(\boldsymbol{x}) = 1 - x_1$. The results of this case of solution are shown in the graphical form in the left picture of figure 1.

Constrained optimization

An additional condition should be stated in case of a constrained problem. If the feasible domain were circular with its center at the origin of coordinates and a diameter r, the condition could be formulated as

$$x_1^2 + x_2^2 <= r^2\,. \tag{14}$$

This condition creates a new third equation $f_3(\boldsymbol{x}) = g(d)$, where $g(d)$ is a penalty function of d as an outer distance of $\boldsymbol{x}$ from the border of the circle with radius r. The function $f_3(\boldsymbol{x}) = 0$ inside the circle, and steep increasing outside. The trace of the solution for $r = 0.5$ is in the middle picture of the figure 1.

Nonlinear regression

The function `LMFnlsq2` may also be used for a fit of nonlinear functions. The third example in the script `LMFnlsq2test` shows how to solve a regression problem of measured

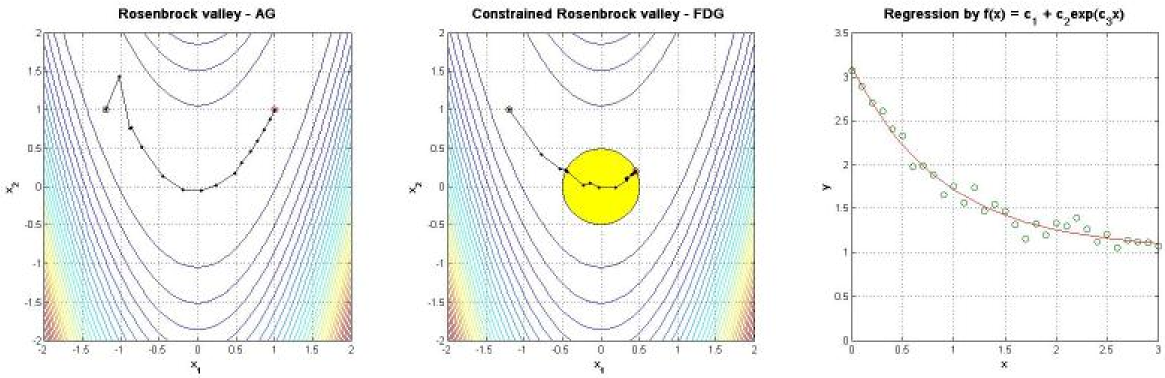

Fig. 1. results of application of the function `LMFnlsq2`

data suffering from a random measurement noise. The solution is presented on the right-hand side of the figure 1.

Script for all three tasks

```
%   LMFnlsq2test.m     Rosenbrock's valleys and a curve fitting
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   The script solves a testing problem of the Rosenbrock's function by
%   minimization of a sum of squares of residuals and a curve fitting.
%   It has been prepared in Matlab v. 2006b.
%   Requirements:                                               FEX ID:
%       inp         function for keyboard input with default value  9033
%       fig         function for coded figure window placement      9035
%       separator   for separating displayed results               11725
%       LMFnlsq2     function for nonlinear least squares           16063
%   Example:
%   A user may run the script multiply changing only few parameters:
%       iprint      as a step in displaying intermediate results,
%       ScaleD      diagonal scale matrix, and
%       Trace       a control variable for storing intermediate data.

% Miroslav Balda
% miroslav AT balda DOT cz
%   2008-08-18  v 1.1   Modified for analytical gradient
%   2009-01-06  v 1.2   updated for modified function LMFnlsq2

clear all
close all

Id = '';
if ~exist('inp.m','file'),       Id = [Id 'inp (Id=9033) ']; end
if ~exist('fig.m','file'),       Id = [Id 'fig (Id=9035) ']; end
if ~exist('separator.m','file'), Id = [Id 'separator (Id=11725 )']; end
if ~exist('LMFnlsq2.m','file'),  Id = [Id 'LMFnlsq2 (Id=16063 )'];  end
if ~isempty(Id)
    error(['Download function(s) ' Id 'from File Exchange'])
end
```

```
separator([mfilename,'   ',date],'#',38)
separator('Rosenbrock without constrains',' ');

ipr= eval(inp('iprint ','5'));  %   step in printing of iterations
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
%   Control variable (step in iterations) for display intermediate results

sd = eval(inp('ScaleD ','[]')); %   D = diag(J'*J)
xy = eval(inp('Trace  ','1'));  %   save intermediate results
disp(' ');

fig(8);
x0 = [-1.2, 1];          %   Usual starting point for Rosenbrock valey
for k = 1:2              %   Cycle for analytical | finite differences gradient
    t  = clock;
    if k==1              %   EXAMPLE 1:  Rosenbrock without constrains
        r  = 0;
        gr = 'AG_ ';    %   Analytical gradient
        ros =  @(x) [10*(x(2)-x(1)^2)
                     1-x(1)];
        jac =  @(x) [-20*x(1), 10
                     -1, 0];
        disp('Analytical gradient')
        [xf,ssq,cnt,loops,XY] = LMFnlsq2 ...% With analytical Jacobian matrix
            (ros,x0,'Display',ipr, 'ScaleD',sd, 'Trace',xy,'Jacobian',jac);
    else                 %   EXAMPLE 2:  Rosenbrock with constraint
        separator('Rosenbrock with constrains',' ')
        gr = 'FDG_ ';   %   Finite difference approx. of gradient
        r = 0.5;
        w = 1000;
        d = @(x) x'*x-r^2; %    delta of squares of position and radius
        ros =  @(x) [10*(x(2)-x(1)^2)
                     1-x(1)
                     (r>0)*(d(x)>0)*d(x)*w
                    ];
        disp('Gradient from finite differences')
        [xf,ssq,cnt,loops,XY] = LMFnlsq2 ...% With finite difference Jacobian mx
            (ros,x0,'Display',ipr, 'ScaleD',sd, 'Trace',xy);
    end
    R = sqrt(xf'*xf);
    fprintf('\n  Distance from the origin R =%9.6f,   R^2 = %9.6f\n', R, R^2);
    separator(['t = ',num2str(etime(clock,t)),' sec'],'*')

    if xy                               %   Saved sequence [x(1), x(2)]
        subplot(1,3,k)
        plot(-2,-2,2,2)
        axis square
        hold on
        fi=(0:pi/18:2*pi)';
        plot(cos(fi)*r,sin(fi)*r,'r')   %   circle
        grid
        fill(cos(fi)*r,sin(fi)*r,'y')   %   circle = fesible domain
        x=-2:.1:2;
        y=-2:.1:2;
        [X,Y]=meshgrid(x,y);
        Z=100*(Y-X.^2).^2 - (1-X).^2;   %   Rosenbrock's function
        contour(X,Y,Z,30)
        plot(x0(1),x0(2),'ok')          %   starting point
        plot(xf(1),xf(2),'or')          %   terminal point
        plot([x0(1),XY(1,:)],[x0(2),XY(2,:)],'-k.') %   iteration path
        if r>0
            tit = 'Constrained';
        else
            tit = '';
        end
```

```
        title([tit,' Rosenbrock valley - ' gr],...
            'FontSize',14,'FontWeight','demi')
        xlabel('x_1','FontSize',12,'FontWeight','demi')
        ylabel('x_2','FontSize',12,'FontWeight','demi')
    end
end

%                       EXAMPLE 3:  Curve fit of decaying exponential
iprint = -1;            %   without displaying lambda
separator('Exponential fit  y(x) = c1 + c2*exp(c3*x)',' ');
t = clock;
c = [1,2,-1];
x = (0:.1:3)';          %   column vector of independent variable values
y = c(1) + c(2)*exp(c(3)*x) + 0.1*randn(size(x)); % dependent variale
                        %   Initial estimates:
c1 = y(end);            %   c1 = y(x->inf)
c2 = y(1)-c1;           %   c2 for x=0
c3 = real(x(2:end-1)\log((y(2:end-1)-c1)/c2));  %   evaluated c3
res = @(c) real(c(1) + c(2)*exp(c(3)*x) - y);   %   anonym. funct. for residua

[C,ssq,cnt] = LMFnlsq2(res,[c1,c2,c3],'Display',iprint);

subplot(1,3,3)
plot(0,0, x,y,'o', x,res(C)+y,'-r', 'Linewidth',1), grid
axis 'square'
title('Regression by f(x) = c_1 + c_2exp(c_3x)',...
      'FontSize',14,'FontWeight','demi')
xlabel('x','FontSize',12,'FontWeight','demi')
ylabel('y','FontSize',12,'FontWeight','demi')
separator(['t = ',num2str(etime(clock,t)),' sec'],'*')
```

Record of one run of LMFnlsq2test

```
>> LMFnlsq2test

######################## LMFnlsq2test   22-Apr-2013 ########################


                       Rosenbrock without constrains

         iprint  = [5,0] =>
         ScaleD  = [] =>
         Trace   = 1 =>

Analytical gradient
***************************************************************************
  itr  nfJ   SUM(r^2)        x           dx           l           lc
***************************************************************************
    0     1  2.4200e+001 -1.2000e+000  0.0000e+000  0.0000e+000  1.0000e+000
                          1.0000e+000  0.0000e+000
    5     8  2.6137e+000 -4.5776e-001 -2.6926e-001  6.9532e-003  8.6655e-004
                          1.3964e-001  3.7337e-001
   10    13  1.9450e-001  5.6651e-001 -8.4431e-002  3.4766e-003  8.6655e-004
                          3.1282e-001 -1.4134e-001
   15    18  1.1799e-003  1.0000e+000 -5.8609e-002  0.0000e+000  8.6655e-004
                          9.9656e-001 -1.1612e-001

   17    19  0.0000e+000  1.0000e+000  0.0000e+000  0.0000e+000  8.6655e-004
                          1.0000e+000  0.0000e+000

  Distance from the origin R = 1.414214,   R^2 =  2.000000

***************************** t = 0.249 sec ******************************
```

```
                        Rosenbrock with constrains

Gradient from finite differences
*************************************************
  itr  nfJ   SUM(r^2)        x           dx
*************************************************
    0     1  4.7961e+006 -1.2000e+000  0.0000e+000
                          1.0000e+000  0.0000e+000
   10    14  1.0610e+000 -2.3397e-002 -1.2107e-001
                         -1.1131e-002  5.5415e-002
   20    28  3.0386e-001  4.4934e-001 -8.3355e-004
                          1.9938e-001 -8.8222e-004
   30    44  2.9665e-001  4.5580e-001 -1.1574e-006
                          2.0553e-001 -1.0814e-006

   35    50  2.9664e-001  4.5580e-001  2.2705e-007
                          2.0554e-001 -1.1629e-007

  Distance from the origin R = 0.500000,   R^2 =  0.250000

***************************** t = 0.156 sec ******************************


                 Exponential fit  y(x) = c1 + c2*exp(c3*x)

*************************************************
  itr  nfJ   SUM(r^2)        x           dx
*************************************************
    0     1  3.2856e-001  1.0678e+000  0.0000e+000
                          2.0012e+000  0.0000e+000
                         -1.2618e+000  0.0000e+000
    1     2  2.2294e-001  1.0355e+000  3.2277e-002
                          2.0807e+000 -7.9454e-002
                         -1.1000e+000 -1.6188e-001
    2     3  2.1980e-001  1.0294e+000  6.0639e-003
                          2.0895e+000 -8.8806e-003
                         -1.1128e+000  1.2882e-002

    3     4  2.1980e-001  1.0293e+000  1.1836e-004
                          2.0896e+000 -8.7186e-005
                         -1.1127e+000 -1.3811e-004

***************************** t = 0.156 sec ******************************
```

The last example shows how to suppress displaying of control parameters lambda. It is also obvious, that good initial estimate of unknown parameters diminishes a necessary number of iterations to minimum.

References

[1] Fletcher, R., (1971): A Modified Marquardt Subroutine for Nonlinear Least Squares. Rpt. AERE-R 6799, Harwell

[2] Balda, M.,(2007): LMFsolve: Levenberg-Marquardt-Fletcher's algoritm for nonlinear least squares problem. MathWorks, MATLAB Central, File Exchange, http://www.mathworks.com/matlabcentral/fileexchange/16063