Uppsala University

Optimization

Assignment 1 - Basic Option
Nonlinear Least Squares Problem
Levenberg-Marquardt Algorithm

Efthymia Chantzi
Email: Efthymia.Chantzi.0787@student.uu.se

MSc programme in Bioinformatics

November 25, 2015

1 Short Description

In this assignment, a Matlab implementation of the ***Levenberg-Marquardt*** algorithm is developed for the solution of an arbitrary nonlinear least squares problem.

1.1 Nonlinear Least Squares

A nonlinear least squares problem is an unconstrained optimization problem, where a function $f(x)$, which is a sum of squares, is minimized:

$$\min_x f(x) = \|F(x)\|_2^2 = \sum_i F_i(x)^2 \tag{1}$$

1.2 Parameter Estimation

In the context of parameter estimation, which occurs in a great number of applications, the auxiliary functions $\{F_i(x)\}$ are not arbitrary nonlinear functions, but they correspond to the residuals in a data fitting problem. Considering a set of data points $t = (t_1, t_2, t_3, \cdots, t_m)^T$, $y = (y_1, y_2, y_3, \cdots, y_m)^T$ and a model $y = \phi(t, \mathbf{x})$, the residuals can be expressed as:

$$\mathbf{x} \in \mathbb{R}^n, \qquad set \quad of \quad unknown \quad parameters$$

$$F_i(x) = \phi(t_i, \mathbf{x}) - y_i, \qquad i = 1, \cdots, m \tag{2}$$

$$\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^m, \qquad residual \quad vector$$

Then, the optimization problem can be restated as:

$$\min_x f(\mathbf{x}) = \frac{1}{2}\mathbf{F}(\mathbf{x})^T\mathbf{F}(\mathbf{x}) \tag{3}$$

where $\mathbf{F}(\mathbf{x})$ is the vector-valued function provided by the user.

1.3 Levenberg-Marquardt Algorithm

The Levenberg-Marquardt method is a popular and effective technique for solving nonlinear least squares problems. Actually, it is a combination of two methods; Steepest Descent and Gauss-Newton. The former is slow but reliable even far from x_*, while the latter is faster but unreliable away from the optimum solution x_*. Consequently, Leveberg-Marquardt acts more like the Steepest Descent method when the unknown parameters are far from their optimal values, but more like the Gauss-Newton method in the opposite case. It uses a search direction that is produced by solving the linear system:

$$\left(\mathbf{J}(\mathbf{x}_k)^T\mathbf{J}(\mathbf{x}_k) + \mu_k\mathbf{I}\right)\mathbf{p}_k = -\mathbf{J}(\mathbf{x}_k)^T\mathbf{F}(\mathbf{x}_k) \tag{4}$$

where μ_k is a positive scalar and $\mathbf{I}$ the identity matrix.

It is true that an algorithm incorporating this technique is likely to fail, when $\mathbf{J}(\mathbf{x}_k)$ is not of full rank and thus, the square matrix $\mathbf{J}(\mathbf{x}_k)^T\mathbf{J}(\mathbf{x}_k)$ is either singular or nearly singular. This fact indicates the production of a random/wrong solution if not any at all. As a result, it would be more effective to work directly with the corresponding linear least squares problem, which is well defined even in cases where $\mathbf{J}(\mathbf{x}_k)$ in not of full rank. However, the solution may not be unique. As a result, the direction $\mathbf{p}_k$ can be calculated with greater reliability when solving the following least squares equation[1] :

$$\begin{bmatrix} \mathbf{J}(\mathbf{x}_k) \\ \sqrt{\mu}\,\mathbf{I} \end{bmatrix} \mathbf{p}_k \approx \begin{bmatrix} -\mathbf{F}(\mathbf{x}_k) \\ \mathbf{0} \end{bmatrix} \tag{5}$$

The algorithm developed for this assignment implements both formulae, (4) and (5). To be more precise, direction $\mathbf{p}_k$ is estimated by default using equation (5). However, the user has the opportunity to change this option with equation (4). More details are provided in section 2, where the Matlab implementation is described thoroughly.

2 Implementation

The implementation[2] of the Levenberg-Marquardt algorithm is divided into four separate parts, each one of them corresponding to a different *.m* file. Specifically, these four files are associated with:

1. objective function
2. numerical approximation of Jacobian through finite differences
3. Levenberg-Marquardt algorithm
4. testing of the algorithm and comparison with the respective built-in function *lsqnonlin*

2.1 Objective Function

The objective function defines the nonlinear least squares problem and is provided by the user. It should be mentioned that this function should be a vector-valued function:

$$\mathbf{F} = \mathbf{r} = \begin{bmatrix} F_1(\mathbf{x}) \\ F_2(\mathbf{x}) \\ \vdots \\ F_m(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \phi(t_1, \mathbf{x}) - y_1 \\ \phi(t_2, \mathbf{x}) - y_2 \\ \vdots \\ \phi(t_m, \mathbf{x}) - y_m \end{bmatrix} \tag{6}$$

In the case of the given test model of question 2, the vector-valued function is:

$$\mathbf{F} = \mathbf{r} = \begin{bmatrix} x_1 e^{x_2 t_1} - y_1 \\ x_1 e^{x_2 t_2} - y_2 \\ \vdots \\ x_1 e^{x_2 t_m} - y_m \end{bmatrix} \tag{7}$$

[1] Appendix D. p.752, Exercise 2.9

[2] MATLAB R2014b

It is important to state that t and y data are not defined inside the residual function but they are passed as input arguments (column vectors) along with the initial solution $\mathbf{x_0}$. In this way, the implementation is parameterized, and offers flexibility, since the user is able to experiment with different data without being required to make changes in the file of the objective function. The output arguments are either the residual vector or the residual vector along with the Jacobian matrix. This choice is made by users, who provide the respective objective function.

2.1.1 Matlab Code

The Matlab code providing the vector-valued function for the given test model, which is described in formula (7), is cited:

```
 %% Chantzi Efthymia - Optimization - Assignment 1 %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function provides the vector-valued objective function of the    %
% given test model: y(t)=x1*e^(x2*t).                                   %
%                                                                       %
%                                                                       %
% %%%% Inputs %%%%                                                      %
% x: vector of unknown parameters                                       %
% t: vector of t data                                                   %
% y: vector of y data                                                   %
%                                                                       %
% %%%% Outputs %%%%                                                     %
% F: vector-valued function-> F(x)= x1*e^(x2*t) - y(t)                  %
% J: matrix of Jacobian. It is an optional output, in order to test     %
% the function of the algorithm in both requested cases, user-supplied  %
% and not user-supplied.                                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [F, J] = residualfunc(x, t, y)

F = x(1)*exp(x(2)*t) - y;

if(nargout > 1)

    J = [exp(x(2)*t)  x(1)*t.*exp(x(2)*t)];

end

end
```

For the needs of the validation phase, *lsqnonlin* is required. According to its official R2014b documentation[3], this function does not accept t and y data as separate input arguments. For this reason, I have exclusively created an other function that accepts only the initial solution as an input argument, while t and y data are incorporated inside its implementation. In this way, *lsqnonlin* and *levmarq* can be tested simultaneously on the test model.

[3] http://se.mathworks.com/help/optim/ug/lsqnonlin.html

```matlab
%% Chantzi Efthymia - Optimization - Assignment 1 %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function provides the vector-valued objective function of the    %
% given test model: y(t)=x1*e^(x2*t). It is written only for the built  %
% in function 'lsqnonlin', which does not accept y and t data as inputs,%
% So, they have been defined inside.                                    %
%                                                                       %
% %%% Inputs %%%%                                                       %
% x: vector of unknown parameters                                       %
%                                                                       %
% %%%% Outputs %%%%                                                     %
% F: vector-valued function-> F(x)= x1*e^(x2*t) - y(t)                  %
% J: matrix of Jacobian. It is an optional output, in order to test     %
% the function of the algorithm in both requested cases, user-supplied  %
% and not user-supplied.                                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [F, J] = residualfuncMatlab(x)

 y = [6.8 3.0 1.5 0.75 0.48 0.25 0.2 0.15]';
 t = [0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0]';

F = x(1)*exp(x(2)*t) - y;

if(nargout > 1)

    J = [exp(x(2)*t) ; x(1)*t.*exp(x(2)*t)];

end

end
```

2.2 Jacobian Approximation

With respect to the requirements of the exercise, the Levenberg-Marquardt function should be called either with or without the user-supplied gradient/jacobian. In case the user provides only the vector-valued function, the implementation finds the numerical approximation of the jacobian, using finite differences. The concept of this approximation with respect to each one of the unknown parameters is:

$$\frac{\mathbf{F}(x+h)-\mathbf{F}(x)}{h} \tag{8}$$

where x is one of the unknown parameters and h is a very small change. For the whole set of unknown parameters, formula (8) is repeated for each one of them yielding a matrix of m rows (rows of residual vector) and k columns (number of unknown parameters). It is true that the numerical approximation of the Jacobian is quite simple but computationally expensive. In addition, an approximation may contain small deviations.

2.2.1 Matlab Code

```matlab
%% Chantzi Efthymia - Optimization - Assignment 1 %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function calculates an approximation of the Jacobian using       %
% finite differences. It is called when the user has not supplied       %
% the gradient/Jacobian inside the objective function                   %
%                                                                       %
% %%%% Inputs %%%%                                                      %
% func: name of user's objective function                               %
% F: residual vector at x                                               %
% x: vector of unknown parameters                                       %
% t: vector of t data                                                   %
% y: vector of y data                                                   %
%                                                                       %
% %%%% Outputs %%%%                                                     %
% J: matrix of Jacobian (dimensions: rows_F x length_x)                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function J = findJacobian(func, F, x, t, y)

numOfParam = length(x);                     %number of unknown parameters
residualRows = length(F);                   %number of rows of residual-vector function
J = zeros(residualRows, numOfParam);        %dimensions of the Jacobian

for i = 1 : numOfParam

    dx = 0.25*(10^-8);                  %very small change
    x_dif = x;
    x_dif(i) = x_dif(i) + dx;           %change in x vector with respect to the i parameter x
    F_dif = feval(func, x_dif, t, y);   %vector-valued function in the changed x vector
    J(:, i) = (F_dif - F)/dx;           %approximation of Jacobian(finite differences)

end


end
```

2.3 Levenberg-Marquardt Function

As requested, the **levmarq** function takes four necessary input arguments:

1. string variable with the name of the objective function
2. vector of initial guess **x**
3. vector of **t** data
4. vector of **y** data

There is also a fifth optional argument, which allows the user to create various optimization parameters and pass them through a cell structure. In case this optional argument is not given, the algorithm is being executed with the default values. Precisely, the syntax of this optional cell structure provided by users should be:

options = { Name, Value }

The detailed documentation for the names and values of possible options is summarized in table 1.

OPTIONS	
NAMES	**VALUES**
'jacobian'	Choose between 'on' and 'off'. Default is 'off' and the jacobian is numerically approximated via finite differences. When set to 'on' the gradient/jacobian provided by the user is used. The algorithm works with the jacobian matrix and when the gradient is supplied, is recognised and converted to the respective jacobian.
'linear'	Choose between 'on' and 'off'. Default is 'on' and responds to the direct calculation of direction p by solving the corresponding linear least squares equation. When set to 'off' the direction is calculated by the Levenberg-Marquadt formula.
'tolerance'	Termination tolerance; a positive scalar. Default is $1e-7$.
'dampingFactor'	Initial value of the Levenberg-Marquardt parameter μ; a positive scalar. Default is 1e2.
'iterations'	Maximum number of iterations till convergence; a positive scalar. Default is 350.
'display'	Choose between 'off' and 'iter'. Default is 'off'. When set to 'iter', a further output argument must be set in order to trace all the intermediate solutions and norm of gradient towards convergence. It iteratively prints the whole route from initial guess to optimum.

***Table 1**: Names and Values for passing optimization options*

2.3.1 Example

A possible intervention would be to provide the Jacobian through the objective function and display iteratively the output at each step. In this case, depending on table 1, the syntax would be:

options = { 'jacobian', 'on', 'display', 'iter' }

2.3.2 Parameter μ

The choice of the regularization parameter μ requires attention, in order to provide the algorithm with reliability for a wide range of problems. Consequently, its value should be updated in each step by some heuristic strategy. As it has already been stated, *Levenberg-Marquardt* constitutes a combination of the Gauss-Newton and Steepest Descent method. This combination serves for the reliability of the latter one far from the optimum, where the size of the residual is large and Gauss-Newton fails. More precisely, this is achieved by the regularization parameter μ. When μ is zero the direction estimated by *Levenberg-Marquardt* is identical to that one of Gauss-Newton. On the other hand, when it tends to infinity the Steepest Descent direction is adopted, which gives reliable results far from the optimum.

As far as my algorithm is concerned, the execution starts with μ being set to 100 as an initial value and its updating is strictly associated with the size of the residual. Being more specific, if the new found step is successful, meaning that it results in a lower function value $f(x_{k+1}) < f(x_k)$[4], then $\mu_{k+1} = {}^{\mu_k}/_{10}$. However, when the step is unsuccessful, the parameter μ is being decreased, so as to invoke a smaller but more reliable step. In this case, $\mu_{k+1} = \mu_k \times 10$.

[4] $f(\mathbf{x}) = \frac{1}{2}\mathbf{r}(\mathbf{x})^T\mathbf{r}(\mathbf{x})$, $\mathbf{r} \rightarrow$ residual vector at $\mathbf{x}$

2.3.3 Updating x

Apart from parameter μ, an other important aspect of the algorithm consists the updating of vector **x** in each iteration. The underlying idea is to update $x_{k+1} = x_k + p_k$, only when $f(x_{k+1}) < f(x_k)$. Otherwise, it remains unchanged, $x_{k+1} = x_k$ and $\mu_{k+1} = \mu_k \times 10$, as mentioned before.

2.3.4 Matlab Code

```
 %% Chantzi Efthymia - Optimization - Assignment 1 %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This is the function that implements the Levenberg-Marquardt algorithm     %
% for an arbitrary nonlinear least squares problem.                           %
%                                                                             %
%                                                                             %
% %%%% Inputs %%%%                                                            %
% func: defines the least-squares problem (vector-valued function)            %
% x0: vector of initial guess/solution                                        %
% t: vector of t data                                                         %
% y: vector of y data                                                         %
% varargin: optional set of optimization parameters defined by user           %
% Otherwise, the following default values are set:                            %
%                                                                             %
% %%%% Default Values %%%%                                                    %
% jacobianMode = 'off'->finite differences, no user-supplied gradient         %
% linear = 'on'->direction p by solving the linear least squares equation     %
% tol = 1e-7->tolerance parameter for error stopping criterion                %
% mu = 1e2->damping factor of Levenberg-Marquardt                             %
% maxIter = 350->maximum number of iterations performed                       %
% The user can change any of these by passing a cell structure 'options'      %
% as the last input argument(varargin)                                        %
%                                                                             %
% %%%% Outputs %%%%                                                           %
% x: solution vector                                                          %
% resnorm: residual norm at the solution x                                    %
% residual: residual vector at the solution x                                 %
% iterationHistory: optional cell structure output argument if the user       %
% wants to be informed of all the intermediate solutions and norm of          %
% gradient towards convergence                                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [x, resnorm, residual, iterationHistory] = levmarq(func, x0, t, y, varargin)


if(nargin < 4) %function aborted if the number of input is less than
               %the four necessary: objective function, initial guess, t data, y data

    error('Invalid number of inputs');

else

    %default values of optimization parameters
```

```
jacobianMode = 'off';
linear = 'on';
tol = 1e-7;
mu = 1e2;
maxIter = 350;
displayIter = 'off';

if(nargin == 5) %optimization option parameters are set/changed by the user

    options = varargin{:};
    if(length(options) > 12)

        error('Maximum number of optimization parameters exceeded.');

    else

        for i = 1 : 2 : (length(options) - 1) %check for valid/invalid options passed by user

            if(strcmpi(options{i}, 'jacobian') == 1)

                if((strcmpi(options{i + 1}, 'on') == 1) || (strcmpi(options{i + 1}, 'off') == 1))
                    jacobianMode = options{i + 1};
                else
                    error('Invalid Jacobian option');
                end

            elseif(strcmpi(options{i}, 'linear') == 1)

                if((strcmpi(options{i + 1}, 'off') == 1) || (strcmpi(options{i + 1}, 'on') == 1))
                    linear = options{i + 1};
                else
                    error('Invalid least squares problem');
                end

            elseif(strcmpi(options{i}, 'tolerance') == 1)

                if(~isnumeric(options{i + 1}))
                    error('Invalid non-numeric tolerance option');
                else
                    tol = options{i + 1};
                end


            elseif(strcmpi(options{i}, 'dampingFactor') == 1)

                if(~isnumeric(options{i + 1}))
                    error('Invalid non-numeric damping factor option');
                else
                    if(options{i + 1} >= 0)
                        mu = options{i + 1};
                    else
                        error('Invalid negative damping factor')
                    end
                end
```

```
            elseif(strcmpi(options{i}, 'iterations') == 1)

                if(~isnumeric(options{i + 1}))
                    error('Invalid non-numeric maximum iterations option');
                else
                    if(options{i + 1} >= 1)
                        maxIter = options{i + 1};
                    else
                        error('Invalid negative or zero iterations option');
                    end
                end

            elseif(strcmpi(options{i}, 'display') == 1)

                if(strcmpi(options{i + 1}, 'iter') == 1) || (strcmpi(options{i + 1}, 'off') == 1)
                    displayIter = options{i + 1};
                else
                    error('Invalid iterative display option');
                end

            else

                error('Unidentified optimization parameter option');

            end

        end

    end

end

end


x = x0;
if(size(x, 2) > size(x, 1))  %the user may give the
                             %initial guess either as a row or a column vector,

    x = x';

end

numOfParam = length(x);
I = eye(numOfParam, numOfParam);
iter = 0;
counter = 1;
xIterations(:, counter) = x;


if(strcmpi(jacobianMode, 'on') == 1)    %user supplies the gradient/Jacobian

    [r, J] = feval(func, x, t, y);     %user-supplied gradient/Jacobian
```

```
    if(size(J, 1) == numOfParam)   %gradient transformed to Jacobian

        J = J';

    end

else             %numerical approximation of Jacobian

    r = feval(func, x, t, y);

    J = findJacobian(func, r, x, t, y);

end
f(counter) = 1/2*(r')*r; %f=1/2*r'*r
gradient_f = J'*r;

if(strcmpi(linear, 'on') == 1)

    linearForm = 1;

else

    linearForm = 0;

end

condition(counter) = norm(gradient_f);
zeroMatrix = zeros(numOfParam, size(r, 2));
while((condition(counter) > tol) && (iter < maxIter))

    iter = iter + 1;
    counter = counter + 1;

    x_k = x;

    if(linearForm == 0)

        p_k = -((J'*J) + (mu*I))\gradient_f; %Levenberg-Marquardt formula
                                             %However, J'*J may be singular or close to singular.

    else

        A = [J ; (sqrt(mu).*I)];
        b = [-r ; zeroMatrix];
        p_k = A\b;              %direction p as a solution to the linear
                                %least squares equation. Default option.

    end
    x = x_k + p_k;


    if(strcmpi(jacobianMode, 'on') == 1)

        [r, J] = feval(func, x, t, y);
        f(counter) = 1/2*(r')*r;
```

```
    else

        r = feval(func, x, t, y);
        J = findJacobian(func, r, x, t, y);

    end


    f(counter) = 1/2*(r')*r;

    if(f(counter) < f(counter - 1))

        mu = mu/10;

    else

        x = x_k;

        if(strcmpi(jacobianMode, 'on') == 1)

            [r, J] = feval(func, x, t, y);

        else

            r = feval(func, x, t, y);
            J = findJacobian(func, r, x, t, y);

        end
        f(counter) = 1/2*(r')*r;
        mu = 10*mu;

    end
    gradient_f = J'*r;
    condition(counter) = norm(gradient_f);
    xIterations(:, counter) = x;

end

x = xIterations(:, end);
resnorm = 2*f(end);
residual = r;

if ((nargout > 3) && (strcmpi(displayIter, 'iter') == 1))

    iterationHistory{:, 1} = xIterations;
    iterationHistory{:, 2} = condition;

end

end
```

3 Testing

The user can experiment with the *levmarq* function either by calling it directly in the command window, as described in 2.1.1, or by using the test file *levmarqSolveTest.m*, which is attached below. Briefly, the user is prompted to type the name of the objective function and then the optimization problem is being solved by *levmarq* and *lsqnonlin*. Results from both routines, as well as the deviations between them, are displayed and saved in a *.txt* file named *output.txt*. Matlab uses a 5-digit output format for the displayed numeric values, which is inconvenient when very small changes between values need to be traced. For this reason, the format has been altered, so as to print 15 digits.

3.1 Initial Guess

It is known that an appropriate initial guess is a prerequisite for achieving convergence. It is essential to examine carefully the data of the respective problem and make a good initial guess that would contribute to the proper behavior of the optimization algorithm.

As far as the test model of question 2 is concerned, a reliable way to come up with a good initial solution is to plot the values of y in proportion with t data.

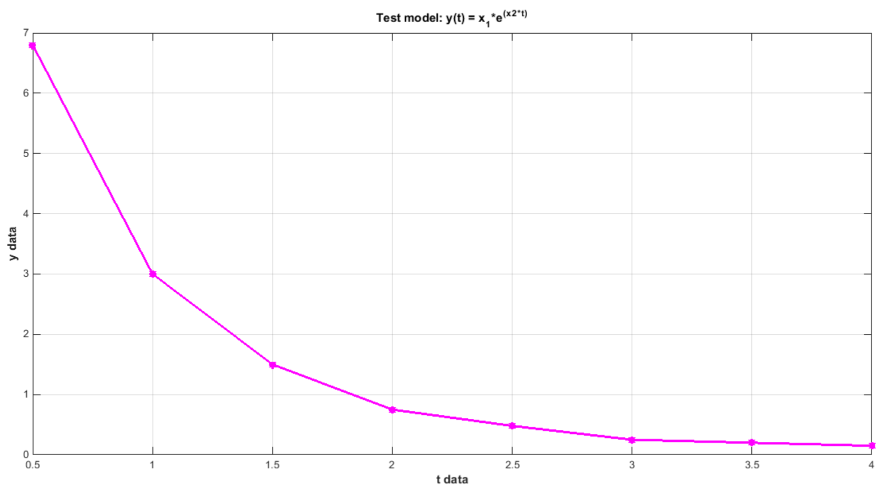

***Figure 1**: y vs. t data of the test model* $y(t) = x_1 e^{x_2 t_1}$

Figure 1 demonstrates that a good initial guess for x_1 is a positive number maybe close to 10 and for x_2 a negative number, possibly -3. The function *levmarq* is going to be tested under different sets of initial solutions for x_1 and x_2, even for not so good ones. Detailed results are presented in section 4, Results.

3.2 Matlab Code

```
%% Chantzi Efthymia - Optimization - Assignment 1 %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Main test file for function 'levmarq', as an alternative to call      %
% it directly in the command window. It requests the name of the        %
% objective function, provided that its .m file is placed in the        %
% folder. Results and deviations from the built in matlab function      %
% 'lsqnonlin' are displayed in the command window and saved in a .txt   %
```

```
% file('output.txt').                                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;
clear all;
close all;

diary on
diary('output.txt'); %save output displayed results in output.txt

format long g %format changed for displaying numbers with greater accuracy
fprintf('%s\n', date);
fprintf('---- Levenberg-Marquardt Algorithm ----\n');
str = input('Enter the name of your objective function: \n', 's');
func = eval(['@' str]); %function handle

%y data (must be a column vector)
y = [6.8 3.0 1.5 0.75 0.48 0.25 0.2 0.15]';

%t data (must be a column vector)
t = [0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0]';

fprintf('\n\n');
fprintf('-----------------------------------------------------------------------------------------------');


x = [10 -3]; %initial guess (either a column vector or row vector)
initial_guess = x';
fprintf('\n running...\n');
fprintf('--- levmarq Results, Initial guess: [%.2f %.2f] ----\n', initial_guess(1), initial_guess(2));
options = {'display', 'iter'};
[x, resnorm, residual, hist] = levmarq(func, x, t, y, options)
history_x = hist{:, 1}
history_norm = hist{:, 2}

fprintf('-----------------------------------------------------------------------------------------------');
fprintf('\n\n\n');

fprintf('--- lsqnonlin Results, Initial guess: [%.2f %.2f] ----\n', initial_guess(1), initial_guess(2))
options = optimset('Display', 'iter-detailed', 'Algorithm', 'levenberg-marquardt');
[solMatlab, resnormMatlab, residualMatlab] = lsqnonlin(@residualfuncMatlab, x, [], [], options)

fprintf('-----------------------------------------------------------------------------------------------\n');
fprintf('\n\n');
fprintf('Comparison: ---levmarq--- & ---lsqnonlin---, Initial guess: [%.2f %.2f]\n', initial_guess(1), initial_guess(2));
fprintf('\n');

difference_x = abs(x - solMatlab)
difference_resnorm = abs(resnorm - resnormMatlab)
difference_residual = abs(residual - residualMatlab)
fprintf('%s\n', date);

diary off
```

4 Results

In this section, detailed results of *levmarq* and *lsqnonlin* are presented for different sets of initial guesses. The cited results are extracted from the *output.txt* that is produced each time the test file is being executed.

4.1 Initial Guess: $[10 - 3]$

```
24-Nov-2014
---- Levenberg-Marquardt Algorithm ----
Enter the name of your objective function:
residualfunc


-----------------------------------------------------------------------------------------------

 running...
--- levmarq Results, Initial guess: [10.00 -3.00] ----

x =

          14.3766288360705
         -1.51391571629679


resnorm =

        0.0959098958362515


residual =

       -0.0560484773725074
         0.163528992655058
       -0.0160160695416782
       -0.0538760476128025
         -0.15345430961809
       -0.0968202573100264
        -0.128144715236865
        -0.116293310997126


hist =

    [2x14 double]    [1x14 double]


history_x =

                        10                        -3
          10.0114295314519         -2.93484468069003
          10.1122197350378         -2.35382625001741
          10.1122197350378         -2.35382625001741
          10.2123602293748         -1.74036469181168
          10.4438448835844        -0.934195675709846
          11.6746304779311         -1.23615425382863
          13.5837903662356         -1.44824030835613
          14.2954041497436         -1.50703598740872
```

```
          14.3713799471442          -1.5133671694807
          14.3762399321593         -1.51387360562689
          14.3765989159905         -1.51391247618812
          14.3766264876381         -1.51391546627128
          14.3766288360705         -1.51391571629679


history_norm =

          6.71906975680455
          6.90249107511453
          8.50580517396821
          8.50580517396821
          8.94536129923662
          13.0774973911306
          1.14842159451086
         0.192347788165629
        0.0139635051277216
       0.00216712492693409
      0.000202817967386832
      1.57130596910334e-05
      1.20231760019666e-06
      9.68500313304843e-08
```

```
--- lsqnonlin Results, Initial guess: [10.00 -3.00] ----

                                        First-Order                    Norm of
 Iteration  Func-count     Residual       optimality      Lambda           step
     0           3       0.0959099        7.83e-08         0.01
     1           6       0.0959099        6.79e-09        0.001    9.72419e-08

Optimization stopped because the relative norm of the current step, 6.726695e-09, is less than
options.TolX = 1.000000e-06.

Optimization Metric                                    Options
relative norm(step) =   6.73e-09                TolX =   1e-06 (default)


solMatlab =

          14.3766289326576
         -1.51391572756251


resnormMatlab =

        0.0959098958362507


residualMatlab =

       -0.0560484700520467
         0.163528978269313
       -0.0160160846489767
       -0.0538760586206722
        -0.153454316621171
       -0.0968202614579506
```

```
        -0.128144717587371
        -0.116293312289593

-------------------------------------------------------------------------------------------------


Comparison: ---levmarq--- & ---lsqnonlin---, Initial guess: [10.00 -3.00]


difference_x =

       9.6587131181991e-08
      1.12657161377427e-08


difference_resnorm =

      7.91033905045424e-16


difference_residual =

      7.32046068208092e-09
      1.43857450396467e-08
      1.51072985232759e-08
      1.10078697268534e-08
      7.00308150358708e-09
      4.14792428182764e-09
      2.35050579000529e-09
      1.29246681412898e-09

24-Nov-2014
```

4.2 Initial Guess: $[11 - 4]$

```
24-Nov-2014
---- Levenberg-Marquardt Algorithm ----
Enter the name of your objective function:
residualfunc


-------------------------------------------------------------------------------------------------

 running...
--- levmarq Results, Initial guess: [11.00 -4.00] ----

x =

          14.3766288068658
         -1.51391571645278


resnorm =

        0.0959098958362516


residual =
```

```
       -0.0560484915981387
         0.163528985735205
       -0.0160160729034613
       -0.0538760492440786
        -0.153454310408776
       -0.0968202576928771
        -0.128144715422062
        -0.116293311086629


hist =

    [2x13 double]    [1x13 double]


history_x =

                    11                    -4
      11.0076908362572     -3.95444306175776
       11.081539127832     -3.51481227742226
      11.5145087493294    -0.747742206148668
       11.249590785911     -1.07594267130341
      12.9993332398411     -1.37370072743565
      14.1591490453094     -1.49539134468699
      14.3613653543334     -1.51239673742693
      14.3755431477445     -1.51379859524519
      14.3765461199634     -1.51390674887099
      14.3766226685287        -1.51391504739
      14.3766284033311     -1.51391567031339
      14.3766288068658     -1.51391571645278


history_norm =

      4.64855431950707
      4.74603974474651
      5.76809238688243
      52.2315045980013
      7.26350848891982
     0.493021591606113
    0.0351669268042278
   0.00411976719656168
  0.000553568587841555
  4.35372341587626e-05
  3.29739838821243e-06
   2.9951622834093e-07
  2.45649082433398e-09

------------------------------------------------------------------------------------------------


--- lsqnonlin Results, Initial guess: [11.00 -4.00] ----

                                        First-Order                    Norm of
 Iteration  Func-count    Residual       optimality      Lambda           step
     0           3       0.0959099        7.56e-09         0.01
     1           6       0.0959099        4.65e-09        0.001     1.20931e-07

Optimization stopped because the relative norm of the current step, 8.365364e-09, is less than
options.TolX = 1.000000e-06.
```

```
Optimization Metric                                     Options
relative norm(step) =   8.37e-09                TolX =   1e-06 (default)


solMatlab =

          14.3766289273288
         -1.51391572707828


resnormMatlab =

        0.0959098958362505


residualMatlab =

       -0.0560484709189133
         0.163528978628614
       -0.0160160841211361
        -0.053876058204524
        -0.153454316346898
       -0.0968202612922042
        -0.128144717492224
        -0.116293312236799

-----------------------------------------------------------------------------------------------


Comparison: ---levmarq--- & ---lsqnonlin---, Initial guess: [11.00 -4.00]


difference_x =

      1.20463004904536e-07
      1.06254982590315e-08


difference_resnorm =

      1.09634523681734e-15


difference_residual =

      2.06792254431321e-08
      7.10659131542002e-09
      1.12176747890658e-08
      8.96044538567509e-09
      5.93812216065359e-09
      3.59932708637878e-09
      2.07016206954513e-09
      1.15017020907437e-09

24-Nov-2014
```

4.3 Initial Guess: $[9 - 2]$

```
24-Nov-2014
---- Levenberg-Marquardt Algorithm ----
Enter the name of your objective function:
residualfunc


-----------------------------------------------------------------------------------------------

 running...
--- levmarq Results, Initial guess: [9.00 -2.00] ----

x =

          14.3766288161961
         -1.51391571574954


resnorm =

        0.0959098958362516


residual =

         -0.05604848485007
          0.163528990013018
       -0.0160160703749825
       -0.0538760478132215
        -0.153454309622754
       -0.0968202572703005
        -0.128144715198569
        -0.116293310969939


hist =

    [2x12 double]    [1x12 double]


history_x =

                         9                        -2
          9.01519728189121         -1.9148943859475
          9.12063516803566        -1.33698976591903
          9.50510613976943       -0.982605879382997
          11.4341287041793        -1.25511010928213
          13.6080712764179        -1.45653581066247
          14.3066016859275        -1.50813289490454
          14.3722467692424        -1.51345617966139
          14.3763031070957        -1.51388043955606
          14.3766042885245         -1.5139130489693
           14.376627758214        -1.51391558592044
          14.3766288161961        -1.51391571574954


history_norm =

          9.07321034889598
```

```
          9.21806878240519
           7.4072973016914
          4.07876116206065
         0.418440455930788
         0.369367747898888
        0.0162042603288551
       0.00185460219539272
      0.000170092602451998
      1.31332506163806e-05
      1.04486043129165e-06
      4.21871356007772e-08
```

```
--- lsqnonlin Results, Initial guess: [9.00 -2.00] ----

                                       First-Order                    Norm of
 Iteration  Func-count    Residual       optimality      Lambda           step
     0           3       0.0959099        4.42e-08         0.01
     1           6       0.0959099        3.24e-09        0.001    1.17535e-07

Optimization stopped because the relative norm of the current step, 8.130454e-09, is less than
options.TolX = 1.000000e-06.

Optimization Metric                                    Options
relative norm(step) =   8.13e-09                 TolX =   1e-06 (default)


solMatlab =

          14.3766289331261
         -1.51391572765869


resnormMatlab =

        0.0959098958362505


residualMatlab =

       -0.0560484701566208
         0.163528978068119
       -0.0160160848147228
       -0.0538760587318998
        -0.153454316689051
       -0.0968202614971593
        -0.128144717609219
        -0.116293312301463
```

```
Comparison: ---levmarq--- & ---lsqnonlin---, Initial guess: [9.00 -2.00]


difference_x =

      1.16929919968811e-07
```

```
      1.19091483341549e-08


difference_resnorm =

      1.12410081243297e-15


difference_residual =

      1.46934491240813e-08
      1.19448992919047e-08
      1.44397402923602e-08
      1.09186782948356e-08
      7.06629754709809e-09
      4.22685880741014e-09
      2.41065031825194e-09
      1.33152407155723e-09

24-Nov-2014
```

4.4 Initial Guess: $[6-5]$

A more challenging option has been set as initial guess.

```
24-Nov-2014
---- Levenberg-Marquardt Algorithm ----
Enter the name of your objective function:
residualfunc


-----------------------------------------------------------------------------------------------

 running...
--- levmarq Results, Initial guess: [6.00 -5.00] ----

x =

          14.3766293358643
         -1.51391576121315


resnorm =

        0.0959098958362576


residual =

        -0.056048394380424
         0.163528960538827
       -0.0160161179346969
       -0.0538760859472208
        -0.153454334934052
       -0.0968202726256619
        -0.128144724035035
        -0.116293315881262


hist =
```

```
    [2x22 double]    [1x22 double]


history_x =

                    6                     -5
     6.00538175885953      -4.98320368198424
     6.05928274419857      -4.81473923815144
     6.60028696399206      -3.09832244713608
     6.60028696399206      -3.09832244713608
     6.60028696399206      -3.09832244713608
     6.72083332702088      -2.65173151264821
     6.72083332702088      -2.65173151264821
     6.85902259599274      -2.11336990827076
     6.85902259599274      -2.11336990827076
     7.00961499026083      -1.49057172114798
     7.00961499026083      -1.49057172114798
     7.14387809539417      -0.96001912859931
     7.92628755311651     -0.906559633870217
     10.7231799924627      -1.22331241540014
     13.4572332700713      -1.45363642954069
     14.3003406226277      -1.50810993268685
     14.3722173012023      -1.51345555398776
     14.3763027742214      -1.51388040353996
     14.3766036489959      -1.51391299231095
     14.3766271558009      -1.51391553118001
     14.3766293358643      -1.51391576121315


history_norm =

     1.76496263945691
     1.78121677167125
     1.95263355772318
     4.91960254747702
     4.91960254747702
     4.91960254747702
     6.18787676067522
     6.18787676067522
     7.98824741964616
     7.98824741964616
     9.57704072892368
     9.57704072892368
     4.61091949783895
    0.947643034381285
     1.55792849370798
    0.687147338794436
   0.0318259038613357
  0.00179621321900209
 0.000170286343206175
  1.30928047903096e-05
  1.05577595027166e-06
  9.68876561215705e-08

--------------------------------------------------------------------------------


--- lsqnonlin Results, Initial guess: [6.00 -5.00] ----
```

```
                                        First-Order                      Norm of
 Iteration  Func-count     Residual       optimality      Lambda           step
     0           3       0.0959099        8.57e-08          0.01
     1           6       0.0959099        9.72e-09         0.001    2.92576e-07

Optimization stopped because the relative norm of the current step, 2.023888e-08, is less than
options.TolX = 1.000000e-06.

Optimization Metric                                     Options
relative norm(step) =   2.02e-08                TolX =   1e-06 (default)


solMatlab =

          14.3766290441944
         -1.51391573820886


resnormMatlab =

        0.0959098958362509


residualMatlab =

       -0.0560484536303596
         0.163528969132564
       -0.0160160968344534
        -0.053876068042363
        -0.153454322779064
       -0.0968202651619674
        -0.128144719707392
        -0.116293313463503

-----------------------------------------------------------------------------------------------


Comparison: ---levmarq--- & ---lsqnonlin---, Initial guess: [6.00 -5.00]


difference_x =

      2.91669957519503e-07
      2.30042878079928e-08


difference_resnorm =

      6.68909372336657e-15


difference_residual =

      5.92499356244502e-08
      8.59373638917305e-09
      2.11002435523966e-08
       1.7904857818607e-08
      1.21549881271221e-08
      7.46369452353512e-09
      4.32764385438489e-09
```

```
      2.41775918730092e-09

24-Nov-2014
```

4.5 Initial Guess: $[3 - 10]$

An even more challenging option has been set as initial guess.

```
24-Nov-2014
---- Levenberg-Marquardt Algorithm ----
Enter the name of your objective function:
residualfunc


-----------------------------------------------------------------------------------------------

 running...
--- levmarq Results, Initial guess: [3.00 -10.00] ----

x =

          14.3766288748286
         -1.51391572038767


resnorm =

        0.0959098958362509


residual =

       -0.0560484729857711
         0.163528988242031
       -0.0160160746471898
        -0.053876051431633
        -0.153454312077399
       -0.0968202587769869
         -0.12814471607198
        -0.116293311457816


hist =

    [2x28 double]    [1x28 double]


history_x =

                         3                       -10
           3.0004581706086         -9.99931067711373
          3.00504145495897         -9.99241416124373
          3.05102746408175         -9.92311172741062
          3.52636942543649         -9.19569112835855
          3.52636942543649         -9.19569112835855
          4.20798489362531          -7.9884395648213
          4.20798489362531          -7.9884395648213
           5.4339137088401         -5.38731633797664
           5.4339137088401         -5.38731633797664
           5.4339137088401         -5.38731633797664
```

```
       5.86585718824049         -4.17452389414476
       5.86585718824049         -4.17452389414476
       6.55539664567715         -2.01219270817026
       6.55539664567715         -2.01219270817026
       6.55539664567715         -2.01219270817026
       6.71307441720489         -1.38647940576814
       6.71307441720489         -1.38647940576814
        6.8510249196659        -0.888674158168717
       7.71748215799451         -0.90169410283715
       10.6424726061297         -1.22235185873956
       13.4469667784054         -1.45429476034629
       14.3010066884671         -1.50821851318212
       14.3723009310676         -1.51346431449341
       14.3763089768015         -1.51388107324893
       14.3766039948045         -1.51391303230316
       14.3766271905585         -1.51391553577015
       14.3766288748286         -1.51391572038767


history_norm =

     0.0827700652400778
     0.0828072774939014
     0.0831841484410705
      0.087037730914033
      0.139216452720937
      0.139216452720937
      0.292855872719297
      0.292855872719297
         1.337580644343
         1.337580644343
         1.337580644343
       2.62400028014203
       2.62400028014203
       8.15693928272937
       8.15693928272937
       8.15693928272937
       9.58240563901059
       9.58240563901059
       3.37724495014609
       0.88183862875729
       1.77093051014739
      0.732936424189548
     0.0332138737025061
    0.00176104080850742
   0.000167107830710262
   1.28667523354007e-05
   1.01723313723061e-06
   7.79909915769929e-08

-----------------------------------------------------------------------------------------------


--- lsqnonlin Results, Initial guess: [3.00 -10.00] ----

                                        First-Order                     Norm of
 Iteration  Func-count     Residual      optimality      Lambda           step
     0           3        0.0959099        6.43e-08        0.01
     1           6        0.0959099        8.77e-10       0.001    6.70616e-08
```

```
Optimization stopped because the relative norm of the current step, 4.638975e-09, is less than
options.TolX = 1.000000e-06.

Optimization Metric                                     Options
relative norm(step) =   4.64e-09                TolX =   1e-06 (default)


solMatlab =

          14.3766289413968
         -1.51391572850664


resnormMatlab =

        0.0959098958362506


residualMatlab =

       -0.0560484691361323
         0.163528977205565
        -0.016016085848511
       -0.0538760595119797
        -0.153454317193427
        -0.096820261798702
        -0.128144717781135
        -0.116293312396398

-----------------------------------------------------------------------------------------------


Comparison: ---levmarq--- & ---lsqnonlin---, Initial guess: [3.00 -10.00]


difference_x =

      6.65682957645686e-08
      8.11896994079575e-09


difference_resnorm =

      3.33066907387547e-16


difference_residual =

      3.84963882993361e-09
      1.10364659633433e-08
       1.1201321203913e-08
      8.08034672505897e-09
      5.11602760155938e-09
      3.02171501709303e-09
       1.7091555948312e-09
      9.38581823373141e-10

24-Nov-2014
```

4.6 Initial Guess: $[20 - 10]$

This initial guess is inappropriate with reference to the given t and y data of the test model. However, I find it useful to trace the progress of the algorithm, even in unexpected cases.

```
24-Nov-2014
---- Levenberg-Marquardt Algorithm ----
Enter the name of your objective function:
residualfunc


-----------------------------------------------------------------------------------------------

 running...
--- levmarq Results, Initial guess: [20.00 -10.00] ----

x =

          14.3766292896182
         -1.51391575993345


resnorm =

        0.0959098958362558


residual =

       -0.0560484117589475
         0.163528954410896
       -0.0160161198597291
       -0.0538760864048221
        -0.153454334939769
       -0.0968202725303314
         -0.12814472394434
         -0.11629331581715


hist =

    [2x16 double]    [1x16 double]


history_x =

                        20                       -10
          20.0004504521705         -9.99548181980017
           20.004962926815         -9.95021730352481
          20.0509012831815         -9.48926723700225
          20.5880476529133         -4.08116363187602
          20.5880476529133         -4.08116363187602
          20.5880476529133         -4.08116363187602
          20.7671510593234         -1.84601711917744
          19.3780022075224         -1.93225966638892
          15.4599442102723         -1.64736049893727
           14.400137526212         -1.51878762207987
          14.3799763083118         -1.51427663054743
          14.3768833312349         -1.51394334608383
          14.3766487725034         -1.51391787187924
```

```
         14.3766302423754         -1.51391587327659
         14.3766292896182         -1.51391575993345


history_norm =

        0.454078744280509
        0.455098447069771
        0.465467553489283
        0.585223365993328
         6.57405666734267
         6.57405666734267
         6.57405666734267
         6.09470633548569
        0.452366526060893
        0.932398513973038
       0.0791135399032268
      0.00170017717333406
     0.000134751944760708
     1.02403558891855e-05
     8.36231201197993e-07
     2.54503672190056e-08

-------------------------------------------------------------------------------------------------


--- lsqnonlin Results, Initial guess: [20.00 -10.00] ----

                                        First-Order                    Norm of
 Iteration  Func-count    Residual       optimality      Lambda           step
     0           3       0.0959099        1.56e-08         0.01
     1           6       0.0959099        9.72e-09        0.001     2.5641e-07

Optimization stopped because the relative norm of the current step, 1.773713e-08,is less than
options.TolX = 1.000000e-06.

Optimization Metric                                 Options
relative norm(step) =   1.77e-08                TolX =   1e-06 (default)


solMatlab =

         14.3766290342134
        -1.51391573725084


resnormMatlab =

       0.0959098958362508


residualMatlab =

      -0.0560484550818954
        0.163528969967027
      -0.0160160957321687
      -0.0538760671918395
       -0.153454322223672
      -0.0968202648280629
        -0.12814471951634
```

```
        -0.116293313357736

-----------------------------------------------------------------------------------------------


Comparison: ---levmarq--- & ---lsqnonlin---, Initial guess: [20.00 -10.00]


difference_x =

      2.55404788873648e-07
      2.26826111227041e-08


difference_resnorm =

      4.94049245958195e-15


difference_residual =

      4.33229478957742e-08
      1.55561310499763e-08
      2.41275603940494e-08
      1.92129826492859e-08
      1.27160971197249e-08
      7.70226854518441e-09
      4.42799985567177e-09
      2.45941397802874e-09

24-Nov-2014
```

5 Conclusions

Looking carefully at the results produced by the aforementioned different sets of initial solutions, it is deduced that the performance of *levmarq* function is very effective, compared to that one of *lsqnonlin*. Actually, the differences between the results from these two functions, are unnoticeable. Being more specific, the error concerning the solution vector $\mathbf{x}$, residual norm $resnorm$ and residual vector $residual$, span the range of $[10^{-9}\ 10^{-7}]$, $[10^{-16}\ 10^{-15}]$ and $[10^{-9}\ 10^{-8}]$ respectively, even for the most challenging and bad initial guesses. Apparently, these differences, which may be caused by some kind of random error, are unimportant and practically zero.

It is true that the user can experiment a lot and perform a great number of executions with different initial solutions. In this way, the performance can be monitored and tested under various conditions. After all, optimization problems are multilateral and sensitive. A very small change to one of the parameters may lead to uncontrolled and undesirable behavior. Consequently, the user must be well aware of the respective problem and make wise and careful use of software resources.

The following link contains the folder of Matlab implementation for this assignment. I have attached my source code for further testing and validation, if required.

https://www.dropbox.com/sh/nr2bik24j6a5wyz/AAA7ZB5QeHlUqlp6Zs305dEma?dl=0