LUND UNIVERSITY

Optimization - FMA051

Project Report

Nonlinear Least-square Fitting Using Gauss-Newton Method

Alexander Gerdt: al2244ge-s@student.lu.se
Hicham Mohamad: hsmo@kth.se

January 31, 2017

1 Introduction

The main task of this project is to perform *nonlinear least-square fitting using Gauss-Newton method*, where we need to solve an optimization problem numerically by an approximation of the Newton method. Specifically, we need to fit the given set of data (t, y) by the following function:

$$\phi(x,t) = x_1 e^{-x_2 t} + x_3 e^{-x_4 t} \tag{1}$$

by solving the **least-squares problem**

$$\min_x f(x) \quad \text{where} \quad f(x) = \sum_m^{i=1} (\phi(x,t_i) - y_i)^2 \tag{2}$$

To achieve this objective we implement the function `gaussnewton()` in Matlab. Then we need to experiment and analyze how the algorithm, i.e. **Gauss-Newton** method, behaves in different examples. At the end, eventual final modifications will be done to improve the performance.

1.1 Gauss-Newton Method

Assuming the general problem of minimizing the sum of squares

$$f(x) = \sum_{i=1}^{m} r_i^2(x) \tag{3}$$

Differentiating we obtain

$$\nabla f(x) = \sum_{i=1}^{m} 2 r_i(x) \nabla r_i(x) \tag{4}$$

Introducing the Jacobian of $r = (r_1, \ldots, r_m)^T$, i.e., the mxn-matrix

$$J(x) = \left(\frac{\partial r_i}{\partial x_k} \right) = \begin{pmatrix} \frac{\partial r_1}{\partial x_1} & \cdots & \frac{\partial r_1}{\partial x_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial r_i}{\partial x_1} & \cdots & \frac{\partial r_i}{\partial x_k} \end{pmatrix} \tag{5}$$

whose rows are the transposed gradients

$$\nabla r_1(x)^T, \ldots, \nabla r_m(x)^T \tag{6}$$

The gradient in 4 can be rewritten

$$\nabla f(x) = 2 J(x)^T r(x) \tag{7}$$

For the Hessian $H(x)$ of f we have the formula

$$H(x) = 2 J(x)^T J(x) + 2 \sum_{i=1}^{m} r_i(x) \nabla^2 r_i(x) \tag{8}$$

Assuming small residuals near the minimum, we neglect the second term in 8, which leads to

$$H(x) \approx 2J(x)^T J(x) \tag{9}$$

In this way, the computation now requires only first derivatives of r_i.

Application of Newton's method to the least-squares problem with this approximation is called the Gauss-Newton method. The iterative scheme becomes

$$x_{k+1} = x_k - \left(J(x_k)^T J(x_k)\right)^{-1} J(x_k)^T r(x_k) \tag{10}$$

where the factor 2 cancels out.

For the Gauss-Newton method to converge it is required not only that the initial point is sufficiently near the minimum, but also that the neglected terms in the Hessian 8 are small enough. It can be shown that the convergence is linear.

To improve the convergence it is desirable to include line search (exact or inexact) in the algorithm. Then, we have the following iteration scheme:

- Solve the system $J(x_k)^T J(x_k) d_k = -J(x_k)^T r(x_k)$ for the direction d_k
- Find some λ_k such that $f(x_k + \lambda_k d_k) \leq f(x_k)$. This can be done, since it can be shown that d_k is a descent direction.
- let $x_{k+1} := x_k + \lambda_k d_k$

This is sometimes called the "damped" Gauss-Newton method.

1.2 Termination Criteria

In most cases the sequence generated by an optimization algorithm needs an infinite number of steps to reach its limit (assuming convergence). Hence some criterion is required on how to decide when to terminate the procedure. In order to be able to interpret the computed results correctly a user must of course be aware of and understand the criterion (criteria) used.

The design of a termination criterion is a very delicate matter. There are sometimes several conflicting goals that it should fulfil. Of course there is the issue of ensuring a certain accuracy of the computed minimum point (or value). But a termination criterion should also have the ability to recognize, preferably as early as possible, that a certain generated sequence is in fact not converging, and the stop the computation. Furthermore, efficiency should be promoted; waste of computer time should be avoided by not carrying out unnecessary iterations.

In this section, we list some suggestions for termination criteria. Others may be designed using the concept of *descent function*. The number ϵ, N and b below are parameters to be decided in advance.

1. We are close to a stationary point, which might of course not be a minimum:

$$\|\nabla f(x_k)\| \leq \epsilon \tag{11}$$

2. this requires some a priori estimate of the minimum value to make a good choice for b. When you expect the minimum to be close to 0, this criterion might be a good alternative:

$$f(x_k) \leq b \tag{12}$$

3. We have not moved very far in the last N iterations. It is time to stop:

$$\|x_{k+N} - x_k\| \leq \epsilon \tag{13}$$

4. The relative distance moved in the last step is small:

$$\frac{\|x_{k+1} - x_k\|}{\|x_k\|} \leq \epsilon \tag{14}$$

5. Not much change in the function value lately:

$$|f(x_{k+N}) - f(x_k)| \leq \epsilon \tag{15}$$

6. The relative change in the function value is small:

$$\frac{|f(x_{k+N}) - f(x_k)|}{|f(x_k)|} \leq \epsilon \tag{16}$$

Sometimes *duality theory* can be used to suggest good termination criteria.

Finally, it should be pointed out that although you feel confident that the generated sequence is convergent and that you have a good estimate of its limit, this may in fact not be the result you are looking for. You may have found a *local minimum*, and the required *global minimum* is somewhere else. A global minimum might not exist. These kinds of problems are the most difficult ones to handle. You will certainly require some help from theory.

2 Implementation of Gauss-Newton method

Following the Gauss-Newton method on page 94 in the text book we could implement this algorithm by writing the Matlab function as shown in Listing 1:
`function gaussnewton(phi,t,y,start,tol,use_linesearch,printout,plotout)`

Where `phi` denotes the function handle for the fitting function, `t,y` are the given data to be fitted, `start` is the initial point chosen by the user and `tol` is a user defined tolerance for termination. Different **termination criteria** are discussed on the page 107. The parameter `use_linesearch` takes the values 1 or 0 depending on whether you want to use a **linesearch** or not. Similarly `printout` and `plotout` can be turned on and off.

Listing 1: The Gauss-Newton method implemented in MATLAB

```
function gaussnewton(phi,t,y,start,tol,use_linesearch,printout,plotout)
% XXXX  %Don't forget:
    %[t,y]=data1; or [t,y]=data2;
```

```
    %gaussnewton(@phi1,t,y,[1;2],0.1,1,1,1);
    %or gaussnewton(@phi2,t,y,[1;2;3;4],0.1,1,1,1);

% XXXX Preparation
%if printout==1
%    time_total=cputime;
%end
close(gcf);                                 %Close all Plots
f_tol=1.0e60;                               %Only for Starting
tolJ=0.0001;                                %For Jacobian
iter=0;                                     %Iteration Step
iterMAX = 200;
x=start;                                    %For printing in the end
auxf=@(x)sum((phi(x,t)-y).^2);              %Auxiliary Function

% XXXX Computing
while f_tol>tol && iter<iterMAX             %Termination Criteria: (Book page 108)
                                            %Change in the Function < tol.

    %Computing Jacobian
    J=zeros(length(t),length(x));
    for i=1:length(x)
        dx=x;
        dx(i)=x(i)+tolJ;
        J(:,i)=(phi(x,t)-phi(dx,t))./tolJ;
        %dx(i)=x(i)-tolJ;                   %Not necessery becaause of Line 21.
    end

    %Determining Direction
    %H=J'*J;                                %Simplyfied Hessian (Book page 94)
    R=(phi(x,t)-y);                         %Vector of r(x)
    maxR=max(abs(R));                       %Determining Residuum, only for print.
    d=-J\R;                                 %Book page 94&95: J'*J*d=-J'*R;

    %Determining Lambda
    if use_linesearch==1
        [lambda,No_of_iterations] = linesearch(auxf,x,d);
    else
        lambda=-0.1;                        %own Choise, good convergence.
        No_of_iterations=0;                 %No Line Search -> 0.
    end

    %New x and Functionvalue
    ns=lambda*d;                            %Step can be also printed
    x=x+ns;                                 %New x
    ss=norm(ns);                            %Step Size, only for print.
    if iter>0                               %Calculate Functiontolerance for Stoping
        f_old=f;
        f=auxf(x);
        f_tol=abs(f_old-f);
    else
        f=auxf(x);                          %Function value only first iteration.
        f_tol=f;
    end
```

```
    iter=iter+1;                            %Iterarion

    %Print
    if printout==1
      if iter==1                            %Only first Line
        fprintf('\n\n%s\t%s\t\t\t%s\t\t\t%s\t\t\t%s\t%s\t%s\t\t%s\t%s\n\n', ...
                '|iter','|x','|d','|ss','|f(x)','|max(|r|)', ...
                '|f_tol','|ls iter','|lambda');
      end
      fprintf('(%0.0f)\t\t%0.4f\t\t%0.4f\t\t%0.3f\t\t%0.2f\t%0.4f\t\t%0.4f\t\t%0.0f\t
            iter,x(1),d(1),ss,f,maxR,f_tol,No_of_iterations,lambda);
        if length(x)>1
            for j=2:length(x)
                fprintf('\t\t%0.4f\t\t%0.4f\n',x(j),d(j));
            end
        end
        fprintf('\n');
    end
end

% XXXX SolutionPlot
if plotout==1
    hold on;
    plot(t, y, 'rx');
    maxt=max(t);
    mint=min(t);
    xt=[mint:(maxt-mint)/500:maxt];   %It is also possible to use t for xt!
    plot(xt, phi(start,xt), 'b-');
    plot(xt, phi(x,xt), 'g-');
    xlabel('t');
    ylabel('y');
    %Plottitle with Iterationsteps
    title(sprintf('Least Square Problem, iteration %0.0f',iter))
    hold off
end
%if printout==1
 %   time_total=cputime-time_total;
  %  fprintf('Total time: %0.1f ms.\n',1000*time_total);
%end
end
```

3 Least squares Fitting

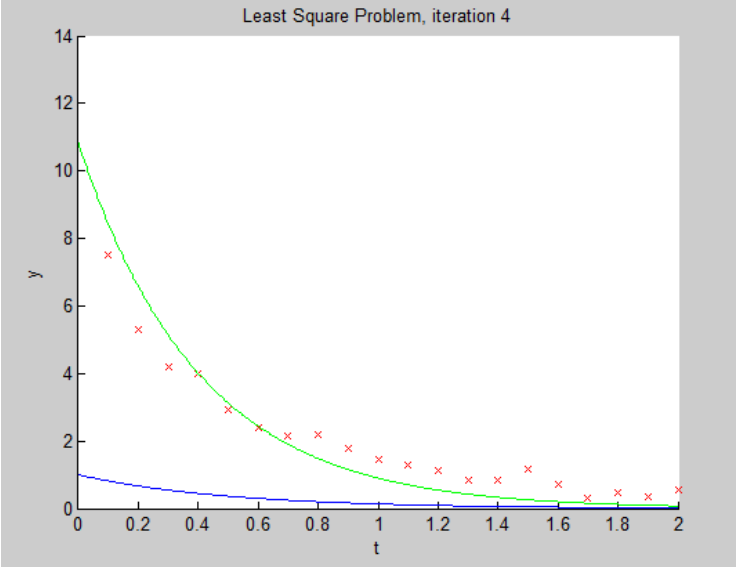

Figure 1: The results of Gauss-Newton program using data1 and phi1

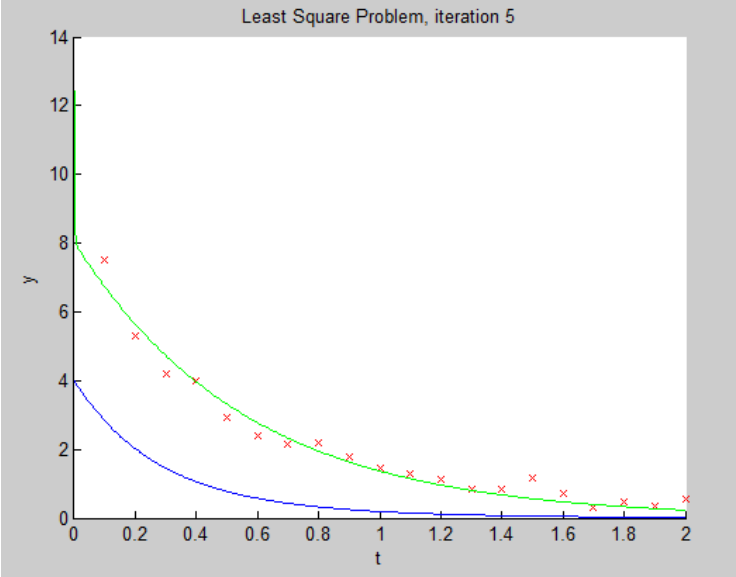

Figure 2: The results of Gauss-Newton program using data1 and phi2

4 Choice of line search and stopping criterion

To improve the convergence of the Gauss-Newton method we choose to use the line search **Armijo** which we see that it is more appropriate. the implemented line search is shown in Listing 2.
As mentioned in the book on page 107, the decision to use a certain termination criteria must be base on the actual application and the optimization method used. In our problem, we choose to stop when we see that there is not much change in the function value lately, i.e.

$$|f(x_{k+N}) - f(x_k)| < \epsilon \tag{17}$$

Listing 2: The line search "quasi" Armijo implemented in MATLAB

```
function [lambda,No_of_iterations] = linesearch(func,x,d)

% time_ls=cputime;          %Only for Computing Time, Starttime.
xstart=x;
No_of_iterations=0;

iterMAX=1000;          %Maximum of Iterations
tolMIN=1.0e-60;        %Toleranz between Funktion values
atol=1.0e-60;          %Minimale Step size !!!!!!!
tol=1;                 %Toleranz value only for starting must be > tolMIN!

a=1;                   %first Stepsize (Lambda) only for starting must be > atol!
ax=0.5;                %a=ax*a by using smaller step ax<1! own decision

bigStep=1.0e70;        %For bad startingponts.

h=0;                   %auxiliary svariable min(fl,fx,fr) -> h=(1,0,2)

fx=func(x);

while tol>tolMIN && No_of_iterations<iterMAX && a>atol

    if h==1
        fl=func(x-a*d);
    end
    if h==2
        fr=func(x+a*d);
    end
    if h==0
        fl=func(x-a*d);
        fr=func(x+a*d);
    end

%compare: min(fl,fx,fr) -> h=(1,0,2)
if fx>fr
    if fr>fl
        h=1;
        tol=fx-fl;
```

```
        fx=fl;
        x=x-a*d;
    else
        h=2;
        tol=fx-fr;
        fx=fr;
        x=x+a*d;
    end
else
    h=0;
    if fx>fl
        h=1;
        tol=fx-fl;
        fx=fl;
        x=x-a*d;
    end
end

if h==0
    a=ax*a;          %Funktion for much smaller value near the search point.
end

if h==1
    a=a+ax*a;
end

if h==2
    a=a+ax*a;
end

if a<atol
    lambda=d\(x-xstart);
    if lambda==0
        a=bigStep;
    end
end

No_of_iterations=No_of_iterations+1;

end

lambda=d\(x-xstart);

% didn't used... from assignment.
% if isnan(func(xstart+lambda*d)) || func(xstart+lambda*d)>func(xstart)
%     error('Bad job of the line search!')
% end

% time_ls=cputime-time_ls;       %Only for Computing Time, Endtime.

end

% lambda_1=linesearch(@test_func, [0;0], [1;0])
% lambda_1=linesearch(@test_func, [0;0], [0;1])
```

5 Consistency

We notice that the consistency of the program behaviour depends on the choice of the initial points.