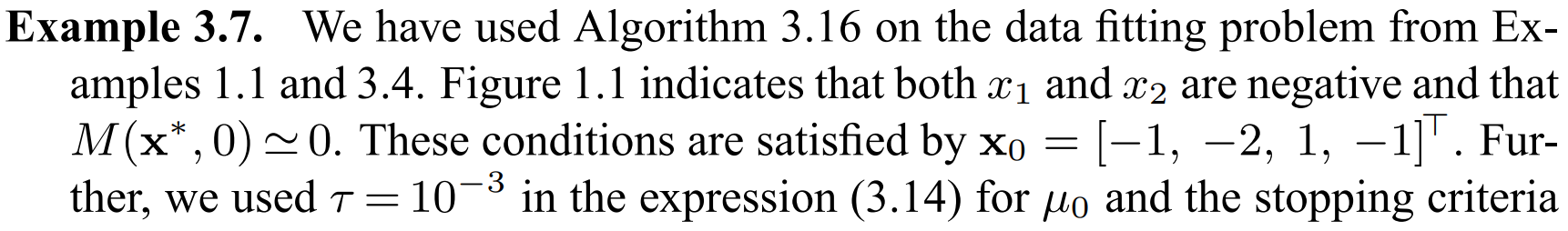
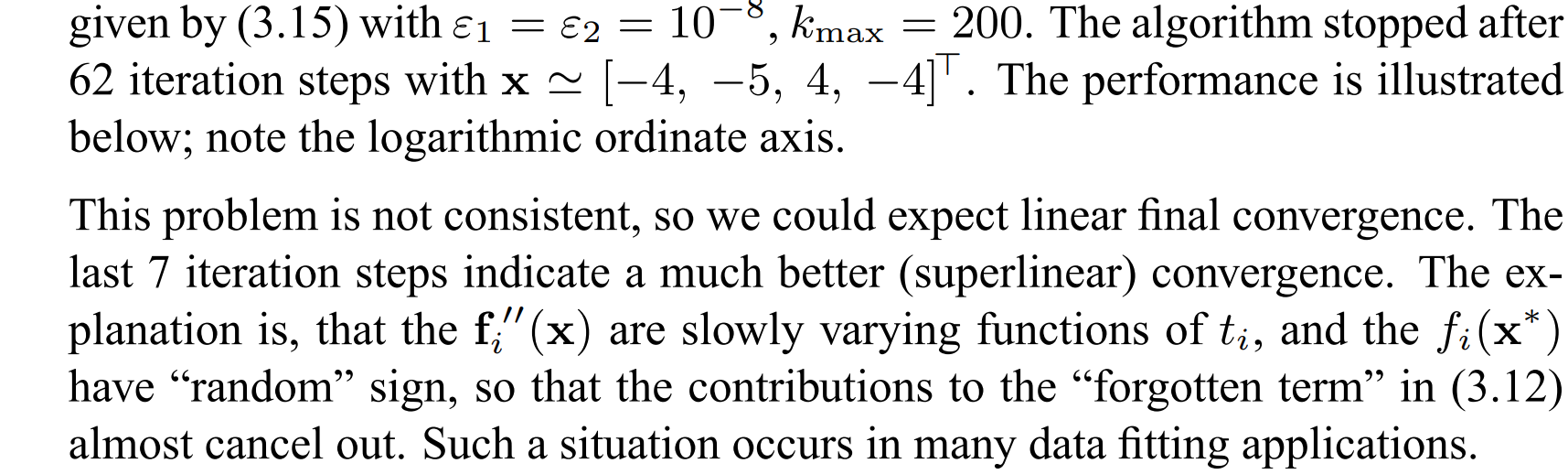
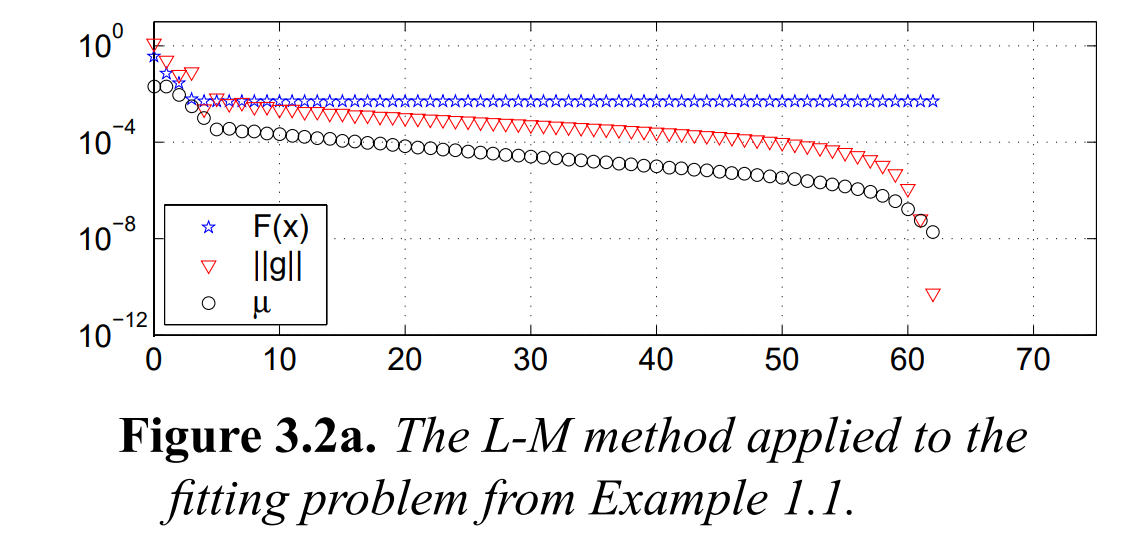
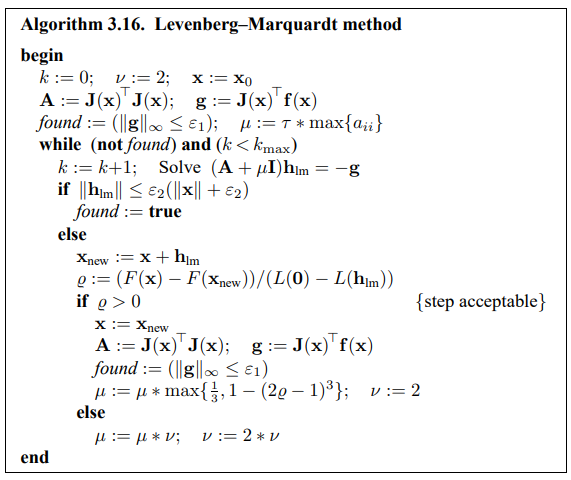
1. 基于附件中的数据efit1.dat，复现文献中的两道例题（50%）。

Example 3.7

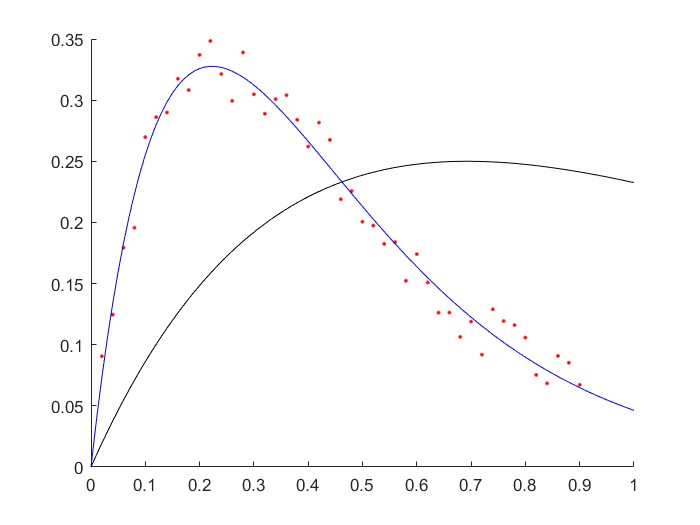
 



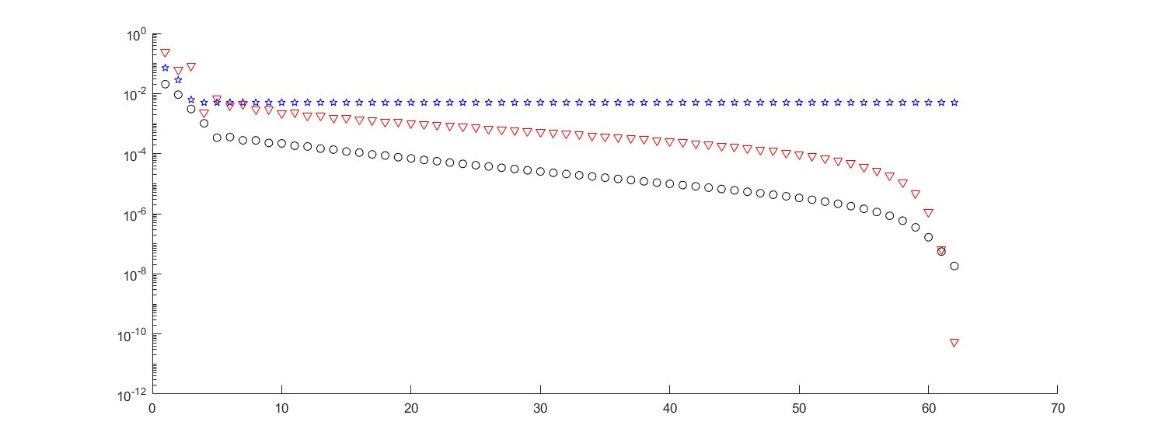
使用Levenberg-Marquardt算法拟合：



拟合结果与原论文中结果基本一致。



参数变化与3.2a基本一致。



代码：

syms c1 c2 c3 c4 t y

data=importdata('efit1.dat');

t1=data.data(:,1);

y1=data.data(:,2);

epsilon=1e-8;

n=200;

disp('Example 3.7')

c0=[-1;-2;1;-1];

lambda=0.001;

c\_root1=LevenbergMarquardt(t1,y1,c0,epsilon,epsilon,lambda,n)

function [ x ] = LevenbergMarquardt( t , y , x0 , epsilon\_1 , epsilon\_2 , tau , N )

%% Levenberg-Marquardt算法，进行高斯曲线拟合

% 输入变量为 t , y , x0 , epsilon\_1 , epsilon\_2 , tau , N

% t：自变量

% y：因变量

% x0：参数初始猜测值

% epsilon\_1：迭代终止误差

% epsilon\_2：迭代终止误差

% tau：倍率

% N：迭代最高次数

% 输出变量为 x

% x：迭代终止后参数的结果

Fx = [];

gg = [];

mul = [];

%% Objective function vector

fun = @(x) y-(x(3)\*exp(x(1)\*t)+x(4)\*exp(x(2)\*t)); % gauss type

% fun = @(x) log( abs( y - x(4) ) ) - log( abs( x(1) ) ) + ( ( t - x(2) ) / x(3) ).^2 / 2; % log type

%% Jacobi matrix for gauss function

Jacobi = @(x) [

-t\*x(3).\*exp(x(1)\*t),...

-t\*x(4).\*exp(x(2)\*t),...

-exp(x(1)\*t),...

-exp(x(2)\*t)

]; % gauss type

% Jacobi = @(x) -1 \* [

% ones( length(t) , 1 ) / x(1) ,...

% ( x(2) - t ) / x(3)^2,...

% -( t - x(2) ).^2 / x(3)^3,...

% 1 ./ ( x(4) - y )

% ]; % log type

%%

count = 0;

v = 2;

x = x0;

Jmatrix = Jacobi( x );

A = Jmatrix' \* Jmatrix;

g = Jmatrix' \* fun( x );

found = ( norm( g , Inf ) < epsilon\_1 );

mu = tau \* max( diag( A ) );

while ~found && ( count < N )

count = count + 1;

h\_lm = ( A + mu \* diag( ones( 4 , 1 ) ) ) \ ( -g );

if ( norm( h\_lm ) <= ( epsilon\_2 \* ( norm( x ) + epsilon\_2 ) ) )

found = 1;

else

x\_new = x + h\_lm;

gain\_radio = ( fun( x )' \* fun( x ) - fun( x\_new )' \* fun( x\_new ) ) / 2 /...

( 0.5 \* h\_lm' \* ( mu \* h\_lm - g ) );

if gain\_radio > 0

x = x\_new;

Jmatrix = Jacobi( x );

A = Jmatrix' \* Jmatrix;

g = Jmatrix' \* fun( x );

found = ( norm( g , Inf ) <= epsilon\_1 );

gg = [gg norm(g,Inf)];

mul = [mul mu];

tmp = 0.5\*fun(x)'\*fun(x);

Fx = [Fx tmp];

mu = mu \* max( [ 1/3 , 1 - ( 2 \* gain\_radio - 1 )^3 ] );

v = 2;

else

mu = mu \* v;

v = 2 \* v;

end

end

end

figure(1);

hold on;

plot(t,y,'r.');

ax = 0:0.01:1;

plot(ax,x(3)\*exp(x(1)\*ax)+x(4)\*exp(x(2)\*ax),'b');

plot(ax,x0(3)\*exp(x0(1)\*ax)+x0(4)\*exp(x0(2)\*ax),'k');

hold off;

figure(2);

hold on;

plot(1:1:count,Fx,'bp')

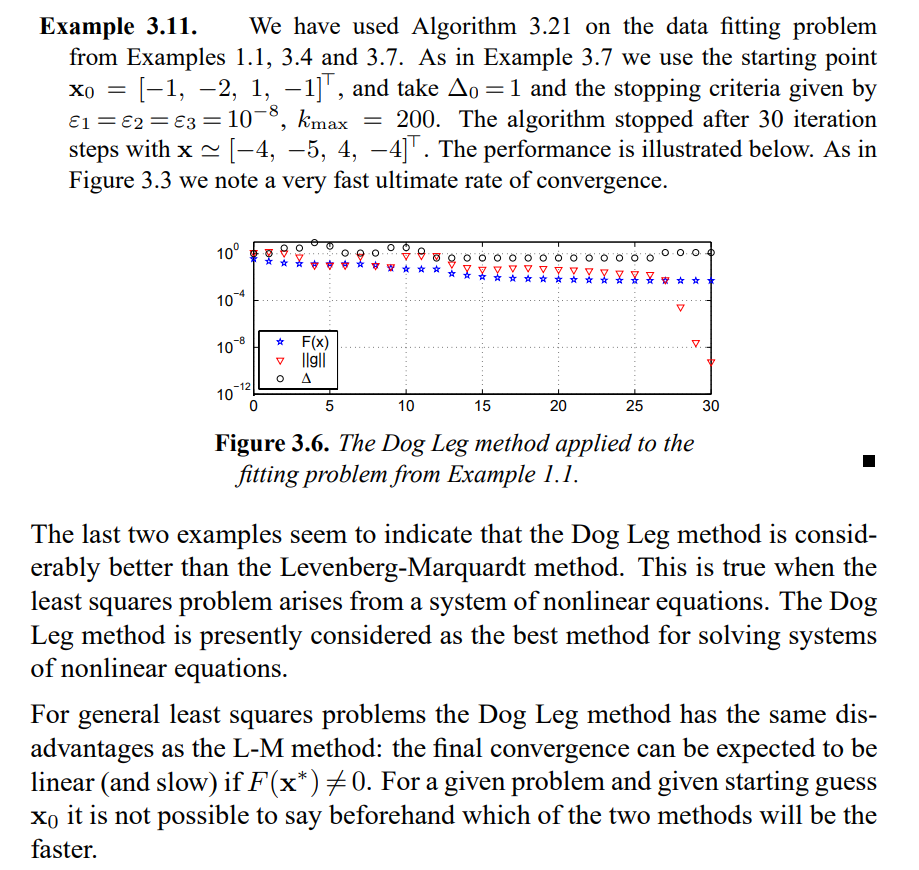
plot(1:1:count,gg,'rv')

plot(1:1:count,mul,'ko')

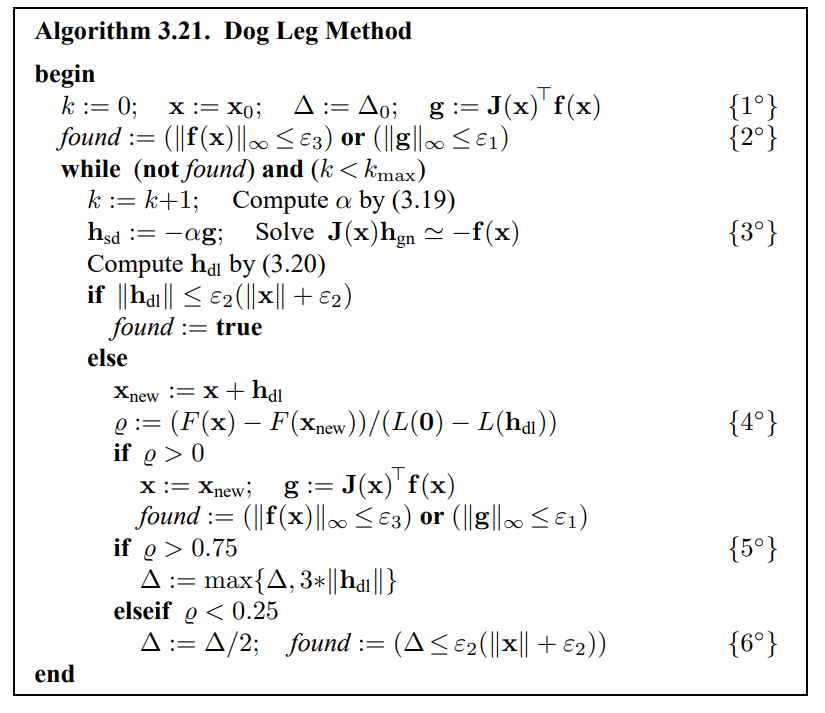
hold off;

end

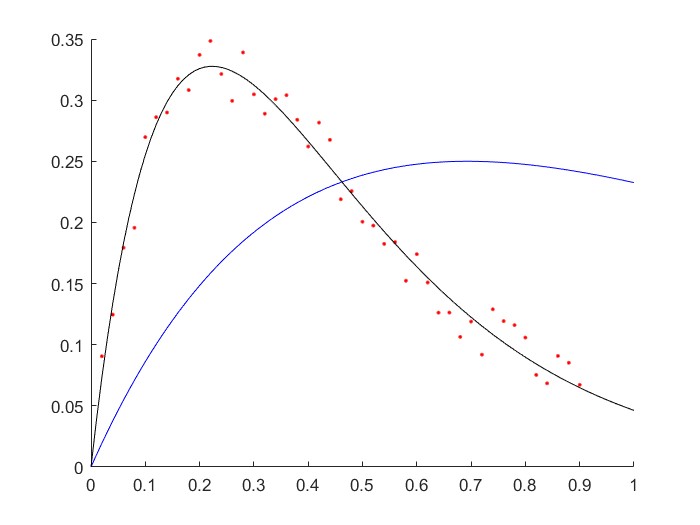
Example 3.11



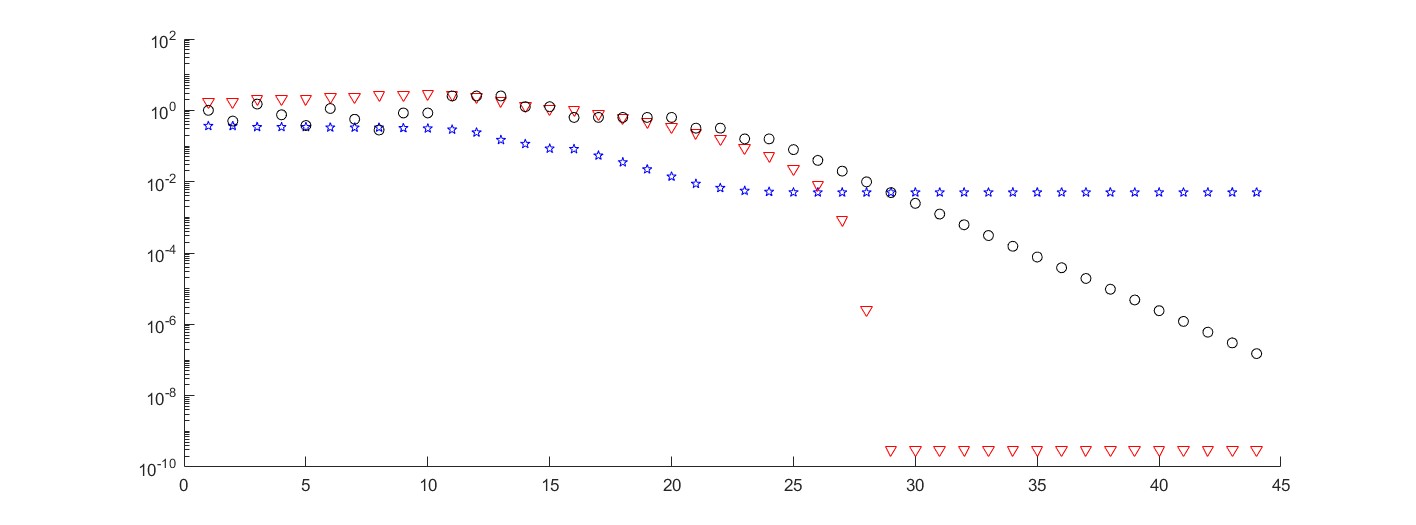
使用Dog Leg Method拟合：



拟合结果与原论文中结果基本一致。



参数变化与3.6基本一致。



代码：

M=100;

data=importdata('efit1.dat');

t\_data=data.data(:,1)';

y\_data=data.data(:,2)';

x0=[-1 -2 1 -1];

k\_max=200;

epsilon\_1=1.0e-8;

epsilon\_2=1.0e-8;

epsilon\_3=1.0e-8;

delta=1;

disp('Example 3.11');

fit\_result=powells\_dog\_leg\_sinus(t\_data, y\_data, x0, k\_max, epsilon\_1, epsilon\_2, epsilon\_3, delta)

function [x\_new] = powells\_dog\_leg\_sinus(t\_data, y\_data, x0, k\_max, epsilon\_1, epsilon\_2, epsilon\_3, delta)

M=length(t\_data(1,:));

k=0;

x=x0;

F\_x\_list = [];

g\_norm\_list = [];

delta\_list = [];

% Jacobian matrix

J=zeros(M,4);

J(:,1)= -t\_data\*x(3).\*exp(x(1)\*t\_data);

J(:,2)= -t\_data\*x(4).\*exp(x(2)\*t\_data);

J(:,3)= -exp(x(1)\*t\_data);

J(:,4)= -exp(x(2)\*t\_data);

% calculate f

f=zeros(1,M);

f=y\_data-(x(3)\*exp(x(1)\*t\_data)+x(4)\*exp(x(2)\*t\_data));

% calculate g

g=transpose(J(:,:))\*transpose(f(1,:));

% calculate norm f

f\_norm=sqrt(sum(f(1,:).\*f(1,:)));

% calculate norm g

g\_norm=sqrt(sum(g(:,1).\*g(:,1)));

% calculate boolean variable found

found\_bool=(f\_norm <= epsilon\_3) | (g\_norm <= epsilon\_1);

% main loop

while ((~found\_bool) && (k < k\_max))

% increase iteration variable k

k=k+1;

% Jacobian matrix

J=zeros(M,4);

% caclulate derivation according to amplitude

% fill first column of Jacobian matrix

J(:,1)= -t\_data\*x(3).\*exp(x(1)\*t\_data);

J(:,2)= -t\_data\*x(4).\*exp(x(2)\*t\_data);

J(:,3)= -exp(x(1)\*t\_data);

J(:,4)= -exp(x(2)\*t\_data);

% calculate f

f=zeros(1,M);

f=y\_data-(x(3)\*exp(x(1)\*t\_data)+x(4)\*exp(x(2)\*t\_data));

% calculate g

g=transpose(J(:,:))\*transpose(f(1,:));

% calculate norm g

g\_norm=sqrt(sum(g(:,1).\*g(:,1)));

% calculate J(x)\*g(x)

Jg=J(:,:)\*g(:,1);

% calculate norm Jg

Jg\_norm=sqrt(sum(Jg(:,1).\*Jg(:,1)));

% calculate alpha

alpha=g\_norm^2/Jg\_norm^2;

% calculate h\_sd

h\_sd=(-1.0)\*alpha\*g(:,1);

h\_sd=transpose(h\_sd);

% calculate h\_gn

A=transpose(J(:,:))\*J(:,:);

B=transpose(J(:,:))\*transpose(f(1,:));

h\_gn=(-1.0)\*transpose(B)\*inv(A);

% calculate h\_dl

% calculate norm h\_gn

h\_gn\_norm=sqrt(sum(h\_gn(1,:).\*h\_gn(1,:)));

% calculate norm alpha\*h\_sd

alpha\_h\_sd=alpha\*h\_sd(1,:);

alpha\_h\_sd\_norm=sqrt(sum(alpha\_h\_sd(1,:).\*alpha\_h\_sd(1,:)));

% calculate norm h\_sd

h\_sd\_norm=sqrt(sum(h\_sd(1,:).\*h\_sd(1,:)));

% caclulate h\_dl

if (h\_gn\_norm <= delta)

h\_dl=h\_gn(1,:);

elseif (alpha\_h\_sd\_norm >= delta)

h\_dl=(delta/h\_sd\_norm)\*h\_sd(1,:);

else

beta=sym('beta');

h\_dl=alpha\*h\_sd(1,:)+beta\*(h\_gn(1,:)-alpha\*h\_sd(1,:));

h\_dl\_norm=sqrt(sum(h\_dl(1,:).\*h\_dl(1,:)));

eqn=(h\_dl\_norm == delta);

beta\_result=solve(eqn, beta, 'Real', true);

beta\_result=double(beta\_result);

clear beta;

h\_dl=alpha\*h\_sd(1,:)+beta\_result(2,1)\*(h\_gn(1,:)-alpha\*h\_sd(1,:));

end

% calculate norm h\_dl

h\_dl\_norm=sqrt(sum(h\_dl(1,:).\*h\_dl(1,:)));

% calculate norm x

x\_norm=sqrt(sum(x(1,:).\*x(1,:)));

delta\_list = [delta\_list delta];

g\_norm\_list = [g\_norm\_list g\_norm];

if (h\_dl\_norm <= epsilon\_2\*(x\_norm+epsilon\_2))

found\_bool=1;

else

x\_new=x(1,:)+h\_dl(1,:);

% calculate F(x)

% caclulate function f

f=zeros(1,M);

f=y\_data-(x(3)\*exp(x(1)\*t\_data)+x(4)\*exp(x(2)\*t\_data));

F\_x=0.5\*sum(f(1,:).\*f(1,:));

F\_x\_list = [F\_x\_list F\_x];

% calculate F(x\_new)

f=zeros(1,M);

f=y\_data-(x\_new(3)\*exp(x\_new(1)\*t\_data)+x\_new(4)\*exp(x\_new(2)\*t\_data));

F\_x\_new=0.5\*sum(f(1,:).\*f(1,:));

% ro denominator

ro\_denominator=0.5\*h\_dl(1,:)\*(alpha\*transpose(h\_dl(1,:))-g(:,1));

% calculate ro - gain ratio

ro=(F\_x-F\_x\_new)/ro\_denominator;

if (ro > 0)

x=x\_new(1,:);

% Jacobian matrix

J=zeros(M,4);

% caclulate derivation according to amplitude

% fill first column of Jacobian matrix

J(:,1)= -t\_data\*x(3).\*exp(x(1)\*t\_data);

J(:,2)= -t\_data\*x(4).\*exp(x(2)\*t\_data);

J(:,3)= -exp(x(1)\*t\_data);

J(:,4)= -exp(x(2)\*t\_data);

% calculate f

f=zeros(1,M);

f=y\_data-(x(3)\*exp(x(1)\*t\_data)+x(4)\*exp(x(2)\*t\_data));

% calculate norm f

f\_norm=sqrt(sum(f(1,:).\*f(1,:)));

% calculate g

g=transpose(J(:,:))\*transpose(f(1,:));

% calculate norm g

g\_norm=sqrt(sum(g(:,1).\*g(:,1)));

% cacluate boolean variable found

found\_bool=(f\_norm <= epsilon\_3) | (g\_norm <= epsilon\_1);

end

if (ro > 0.75)

% calculate norm h\_dl

h\_dl\_norm=sqrt(sum(h\_dl(1,:).\*h\_dl(1,:)));

% calculate new delta

delta=max([delta 3\*h\_dl\_norm]);

elseif (ro < 0.25)

% calculate new delta

delta=delta/2;

% calculate norm x

x\_norm=sqrt(sum(x(1,:).\*x(1,:)));

% calculate boolean variable found

found\_bool=(delta <= epsilon\_2\*(x\_norm+epsilon\_2));

end

end

end

figure(1);

hold on;

plot(t\_data,y\_data,'r.');

ax = 0:0.01:1;

plot(ax,x0(3)\*exp(x0(1)\*ax)+x0(4)\*exp(x0(2)\*ax),'b');

plot(ax,x\_new(3)\*exp(x\_new(1)\*ax)+x\_new(4)\*exp(x\_new(2)\*ax),'k');

hold off;

figure(2);

hold on;

plot(1:1:k,F\_x\_list,'bp')

plot(1:1:k,g\_norm\_list,'rv')

plot(1:1:k,delta\_list,'ko')

hold off;

end