```
close all
clear all
clc

sig1 = 1e-12; sig2 = 0.15;
m1 = 0; m2 = 0; mo = [m1 m2]';

mtest1 = [1 0]'; mtest2 = [0 1]';
Y = @(t,m) (m(1).*exp(m(2).*t));
t = [1 2 4 5 8]';

L = eye(2);

max_iter = 1e3;
tol = 1e-6;

warning('off','all')

J = @(t,m) [(exp(m(2)*t)), (m(1)*t.*exp(m(2)*t))]; %exact jacobian

N0.1

%synthetic data sets
ds1 = Y(t,mtest1);
ds2 = Y(t,mtest2);

m1 = occam(Y, J, L, ds1, mo, 0,t);
disp(['Resulting parameter estimates are [',num2str(m1'),']']);

No.2
[m,n] = size(J(t,mo));
delta1 = (m-n)*sig1^2;
delta2 = (m-n)*sig2^2;
noise = sig1*randn(m,1); %noise

%introduce noise to the data
dn1 = ds1 + noise;
m1 = occam(Y, J, L, dn1, mo, delta1,t);
disp(['Resulting parameter estimates are [',num2str(m1'),']']);

%introduce noise to the data
dn2 = ds2 + noise;
m2 = occam(Y, J, L, dn2, mo, delta1,t);
disp(['Resulting parameter estimates are [',num2str(m2'),']']);

No.3
noise2 = sig2*randn(m,1); %noise
%introduce noise to the data
dn1 = ds1 + noise2;
m1 = occam(Y, J, L, dn1, mo, delta2,t);
disp(['Resulting parameter estimates are [',num2str(m1'),']']);
```

```
%introduce noise to the data
dn2 = ds2 + noise2;
m2 = occam(Y, J, L, dn2, mo, delta2,t);
disp(['Resulting parameter estimates are [',num2str(m2'),']']);

function m = occam(fun, jac, L, d, m0, delta,t)
m = m0;
oldm = zeros(size(m));
iter = 0;
mchi2 = 100;
nr = 1;

% while we have not converged sufficiently or the data misfit is higher than
% allowed keep iterating
while (nr > 1e-6 || (mchi2 > delta^2 * 1.01))
    % only allow 30 iterations
    iter = iter + 1;
    if (iter > 30)
        return;
    end

    % store the old mode to test for convergance
    oldm = m;

    % get the current data that would be generated and the jacobian
    G = feval(fun,t, m);
    J = feval(jac, t,m);

    % get the dhat that is in equation 10.14
    dhat = d - G + J * m;

    % This is a simple brute force way to do the line search.  Much more
    % sophisticated methods are available.  Note: we've restricted the line
    % search to the range from 1.0e-20 to 1.  This seems to work well in
    % practice, but might need to be adjusted for a particular problem.

    alphas = logspace(-20, 0, 100);
    for i = 1:length(alphas)
        M = J' * J + alphas(i)^2 * L' * L;

        % if M is not terribly conditioned
        if (cond(M) < 1.0e15)
            m = inv(J'*J+alphas(i)^2*L'*L) * J' * dhat;

            % store the associated data misfit
            chis(i) = norm(feval(fun,t, m)-d, 2)^2;
        else
            % M behaves poorly enough it should not be used
            chis(i) = + Inf;
        end
    end
```

```
    [Y, I] = min(chis);

    if (Y > delta^2)
        %disp('Improving Chi^2');
        alpha = alphas(I(1));
    else
        %disp('Smoothing m');
        I = find(chis <= delta^2);
        alpha = alphas(max(I));
    end

    % store the new model and misfit
    m = (J' * J + alpha^2 * L' * L) \ J' * dhat;
    mchi2 = norm(feval(fun, t,m)-d, 2)^2;
    nr = norm(oldm - m);
    %mchi2 = norm(fun(t,m)-d, 2)^2;
end
```