Homework # 3

Carlos Rangel

Question 1

Estimate the parameter vector Beta using the maximum likelihood estimator computed via the Nelder-Mead simplex method

```
% Load Data
load('hw3.mat');

% Define the Objective function
min_lnL =@(b) (-log_like(b, X, y));

% Initial Guess
b0 = ones(6,1);

% Set Options for Nelder-Mead
options_mle_nm = optimset('MaxFunEvals', 1000);

% Perform Nelder-Mead Optimization
[bsol_nm, fval, exitflag, ...
    output] = fminsearch(min_lnL, b0, options_mle_nm);
% Display Optimizer's report
output
```

```
output =
    iterations: 450
     funcCount: 732
     algorithm: 'Nelder-Mead simplex direct search'
       message: 'Optimization terminated: the current x satisfies the
       termination criteria using OPTIONS.TolX of 1.000000e-04 and
       F(X) satisfies the convergence criteria using OPTIONS.TolFun
       of 1.000000e-04
```

```
% Estimate
bsol_nm
```

```
bsol_nm = 6x1
    2.5924
   -0.0333
    0.1162
```

```
-0.3572
 0.0783
-0.4131
```

Question 2

Estimate the Parameter vector beta using the maximum likelihood estimator computed via a Quasi-Newton Optimization Method, report which method you choose.

```
% Define Objective Function
min_lnL = @(b) (-log_like(b, X, y));

% The following fminunc performs a maximization of ...
    the log-likelihood
% function using BFGS. (Use same initial guess as ...
    before)
[bsol_qn,fval,exitflag,output] = fminunc(min_lnL, b0);
```

```
Local minimum found.

Optimization completed because the size of the gradient is less than
the default value of the optimality tolerance.
```

```
% Display Results
output
```

```
output =
       iterations: 1
        funcCount: 14
         stepsize: 1.0459
     lssteplength: 1.0640e-41
    firstorderopt: 3.1287e+14
        algorithm: 'quasi-newton'
          message: 'Local minimum found.

          Optimization completed because the size of the gradient is less than
          the default value of the optimality tolerance.

          Stopping criteria details:

          Optimization completed: The first-order optimality measure,
          3.328952e-27, is less than options.OptimalityTolerance = 1.00000e-06.
```

```
bsol_qn
```

```
bsol_qn = 6x1
    0.9825
         0
    0.7368
    0.9196
    0.8973
    0.9142
```

Question 3

Estimate the parameter vector beta using nonlinear least squares estimator computed using the command lsqnonlin. What computation method are you using?

```
% Define Objective Function
residuals= @(b) resids(b, X, y);

% Specify Options. We are Using Levenberg-Marquadt
options_nonls=optimoptions('lsqnonlin', ...
    'Algorithm', 'levenberg-marquardt', ...
    'MaxFunctionEvaluations', 10000, ...
    'FunctionTolerance', 1e-6, 'MaxIterations', 400, ...
    'Display', 'off');

% Optimize
[bsol_lsqnonlin, ...
    resnorm,rep_resid,exitflag,output]=lsqnonlin(residuals, ...
    b0, [], [], options_nonls);

% Report Results
output
```

```
output =
        iterations: 12
         funcCount: 91
          stepsize: 1.0000
      cgiterations: []
     firstorderopt: 1.0699e+69
         algorithm: 'levenberg-marquardt'
           message: 'Local minimum found.

          Optimization completed because the size of the gradient is
```

```
        less than 1e-4 times the selected value of the function tolerance.
```

```
bsol_lsqnonlin
```

```
bsol_lsqnonlin = 6x1
  -10.9840
    1.0000
    0.9989
    1.0000
    1.0000
    1.0000
```

Question 4

Estimate the parameter vector beta using the nonlinear least squares method estimator computed using the Nelder-Mead Simplex Method

```
% Define The Objective Function
rss=@(b) RSS(b, X,y);

% Set Options for Nelder-Mead
options_ls_nm=optimset('MaxFunEvals', 10000);

% Optimize and Report Results
[bsol_rss_nm,fval,exitflag,output]=fminsearch(rss,b0, ...
    options_ls_nm);
output
```

```
output =
     iterations: 638
      funcCount: 1034
      algorithm: 'Nelder-Mead simplex direct search'
        message: 'Optimization terminated: the current x satisfies the
        termination criteria using OPTIONS.TolX of 1.000000e-04 and
        F(X) satisfies the convergence criteria using
        OPTIONS.TolFun of 1.000000e-04
```

```
bsol_rss_nm
```

```
bsol_rss_nm = 6x1
    3.0081
   -0.5959
    0.7868
    3.0735
    0.1022
```

```
-3.4007
```

Question 5

Test all four approaches with regard to the choice of initial values. Roughly rank them in order of robustness and time to convergence. Submit a short writeup summarizing your results.

```
% Create 80 Initial Guesses Drawn from a Uniform(0,1)
% Obtain the Estimates
% Calculate the Standard Deviation of each beta ...
    coefficient

n0 = 80;

% Matrix For Storing vector estimates using MLE and ...
    Nelder-Mead

B_mle_nm = zeros(6,n0);

% Matrix for Storing vector estimates using MLE and ...
    Quasi-Newton
B_mle_qn = zeros(6,n0);

% Matrix for Storing vector estimates using ...
    nonlinear least swuares
B_lsqnon = zeros(6,n0);

% Matrix for Storing vector estimates using ...
    nonlinear least squares and
% Nelder-Mead
B_lsqnon_nm = zeros(6,n0);

% Matrix for Storing the computation times of MLE ...
    using Nelder-Mead
mle_nm_time = zeros(n0,1);

% Matrix for storing the computation times of MLE ...
    using Quasi Newton
mle_qn_time = zeros(n0,1);

% Matrix for Storing the computation times of ...
    Nonlinear Least Squares
% Command
lsqnonlin_time = zeros(n0,1);
```

```
% Matrix for Storing the computation times of ...
    Nonlinear least squares using
% Nelder-Mead
nonls_nm_time = zeros(n0,1);

% Options for Nelder-Mead Optimizations
options_nm = optimset('MaxFunEvals', 10000, ...
    'Display', 'off');

% Options for 'Quasi-Newton' Options
options_qn = optimoptions(@fminunc, ...
    'MaxFunctionEvaluations', 10000, 'Display', 'off');

% Options for Nonlinear Least Squares Command
options_nonls = optimoptions('lsqnonlin', ...
    'Algorithm', 'levenberg-marquardt', ...
    'MaxFunctionEvaluations', 10000, ...
    'FunctionTolerance', 1e-6, 'MaxIterations', 400, ...
    'Display', 'off');

for i = 1:n0
     % Choose Initial Guess at Randome
     b0s = rand(6,1);

     % Solve using MLE and Nelder-Mead
     tic;
     B_mle_nm(:,i) = fminsearch(min_lnL, b0s, ...
         options_nm);
     mle_nm_time(i) = toc;

     % Solve using MLE and Quasi-Newton
     tic;
     B_mle_qn(:,i) = fminunc(min_lnL, b0s, ...
         options_qn);
     mle_qn_time(i) = toc;

     % Solve using Nonlsq
     tic;
     B_lsqnon(:,i) = lsqnonlin(residuals, b0s, [], ...
         [], options_nonls);
     lsqnonlin_time(i) = toc;

     % Solve using Nonlsq and Nealder-Mead
     tic;
     B_lsqnon_nm(:,i) = fminsearch(rss, b0s, ...
```

```
        options_nm);
    nonls_nm_time(i) = toc;

end
```

Compute Standard Deviations and Timings

```
% Standard Deviation of Maximum Likelihood using ...
    Nelder-Mead
std_mle_nm=std(B_mle_nm, 0, 2);

% Standard Deviation of Maximum Likelihood using ...
    Quasi-Newton
std_mle_qn = std(B_mle_qn, 0, 2);

% Standard Deviation of Nonlsq command
std_lsqnon = std(B_lsqnon, 0, 2);

% Standard Deviation of Nonlinear least squares ...
    using Nelder-Mead
std_lsqnon_nm=std(B_lsqnon_nm, 0, 2);

% Average Time Taken by Maximum Likelihood using ...
    Nelder-Mead
avg_time_mle_nm=mean(mle_nm_time);

% Average Time taken by MLE using Quasi-Newton
avg_time_mle_qn = mean(mle_qn_time);

% Average Time Taken by Nonlsq command
avg_time_nonlsq = mean(lsqnonlin_time);

% Average Time Taken by nonlinear least squares ...
    using Nelder-Mead
avg_time_nonls_nm=mean(nonls_nm_time);
```

Display Results

Table for Displaying Standard Deviations:

MLENM: Maximum Likelihood using Nelder Mead

MLEQN: Maximum Likelihood using BFGS

LSQNONL: using Matlab's nonlinear least squares command

NLSNM: Minimizing nonlinear least squares using Nelder-Mead

```
stds = table(std_mle_nm, std_mle_qn, std_lsqnon, ...
    std_lsqnon_nm, 'VariableNames', {'MLENM', ...
    'MLEQN', 'LSQNONLIN', 'NLSNM'}, 'rowNames', {'b0' ...
    ,'b1', 'b2', 'b3', 'b4', 'b5'})
```

	MLENM	MLEQN	LSQNONLIN	NLSNM
1 b0	1.1314	0.3883	3.1028	24.8713
2 b1	0.0227	0.2951	0.3045	14.7222
3 b2	0.0139	0.2646	0.2843	11.3743
4 b3	0.0843	0.3262	0.3398	8.8462
5 b4	0.0376	0.2894	0.2990	20.4501
6 b5	0.0802	0.3045	0.3250	59.8913

```
avg_times = table(avg_time_mle_nm, ...
    avg_time_mle_qn, avg_time_nonlsq, ...
    avg_time_nonls_nm, 'VariableNames', {'MLENM', ...
    'MLEQN', 'LSQNONLIN', 'NLSNM'})
```

	MLENM	MLEQN	LSQNONLIN	NLSNM
1	0.0478	0.0042	0.0052	0.0178

Roughly speaking, we can see that the ranking, taking into account robustness and time to convergence, is:
1.- MLENM 2.- MLEQN 3.- LSQNONLIN 4.- NLSNM