Solutions to Homework 3 (covering Statistics Lectures 3 and 4)

Contents

Problem 0

```
load('Homework3.mat');
```

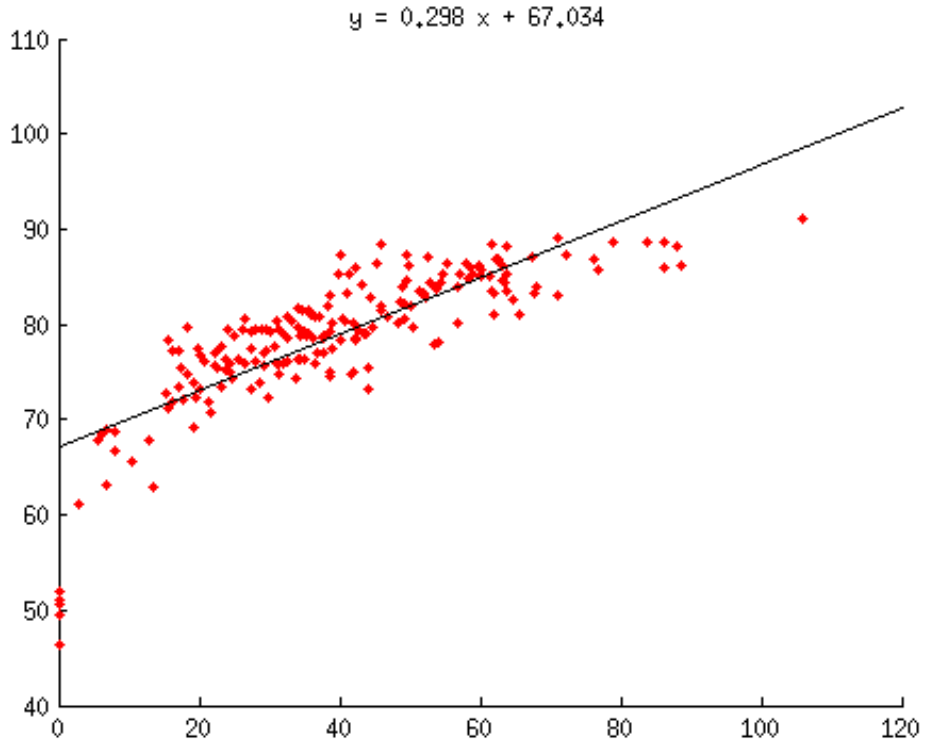
Problem 1

```
% construct regressor matrix
X = score1(:);
X(:,end+1) = 1;  % need a constant regressor

% construct data vector
y = score2(:);

% use OLS to estimate weights
w = inv(X'*X)*X'*y;

% visualize
figure; hold on;
scatter(score1,score2,'r.');
ax = axis;
plot(ax(1:2),w(1)*ax(1:2)+w(2),'k-');
title(sprintf('y = %.3f x + %.3f',w(1),w(2)));
```



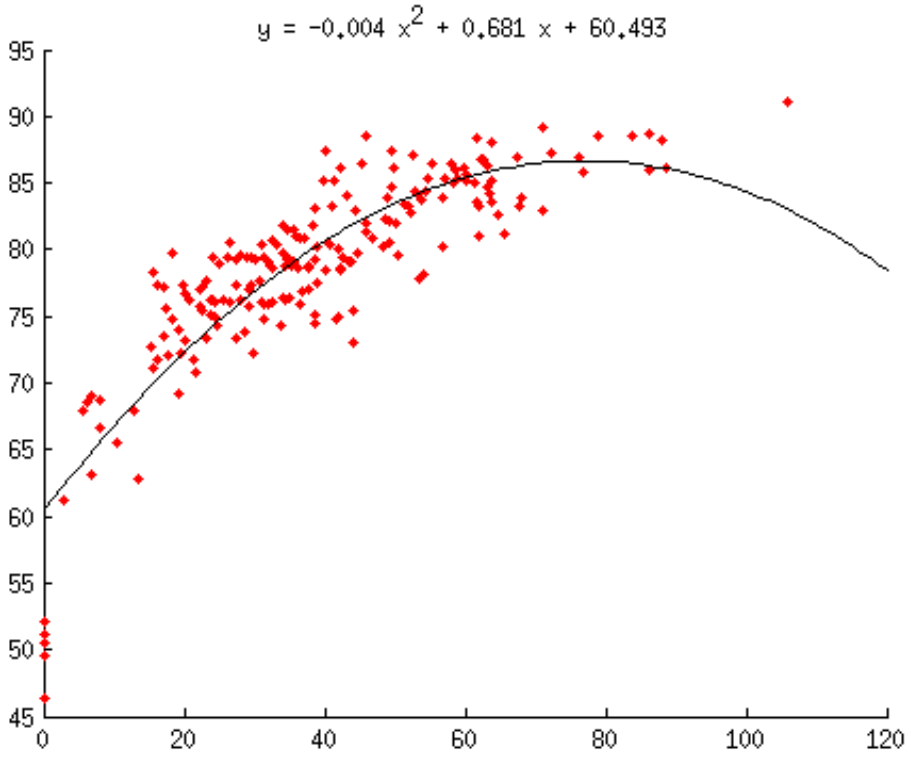
Problem 2

```
% construct regressor matrix
X = score1(:).^2;
X(:,end+1) = score1(:);
X(:,end+1) = 1;

% construct data vector
y = score2(:);

% use OLS to estimate weights
w = inv(X'*X)*X'*y;

% visualize
figure; hold on;
scatter(score1,score2,'r.');
ax = axis;
  % need more than two points since function may be nonlinear
xx = linspace(ax(1),ax(2),100);
plot(xx,polyval(w,xx),'k-');
title(sprintf('y = %.3f x^2 + %.3f x + %.3f',w(1),w(2),w(3)));
```



Problem 3

```
% define optimization options
options = optimset('Display','iter','FunValCheck','on', ...
                   'MaxFunEvals',Inf,'MaxIter',Inf, ...
                   'TolFun',1e-6,'TolX',1e-6);

% define bounds for parameters
%               a    n     b
paramslb = [-Inf    0  -Inf];  % lower bound
paramsub = [ Inf  Inf   Inf];  % upper bound

% define function that evaluates model given a vector
% of parameters (pp) and a matrix of x-values
modelfun = @(pp,xx) pp(1)*xx.^pp(2) + pp(3);

% perform optimization for first seed
params0 = [1 1 0];
[paramsA,resnormA] = lsqcurvefit(modelfun,params0,score1,score2,paramslb,paramsub,options);

% perform optimization for second seed
params0 = [10 7 100];
[paramsB,resnormB] = lsqcurvefit(modelfun,params0,score1,score2,paramslb,paramsub,options);

% visualize
figure; hold on;
scatter(score1,score2,'r.');
ax = axis;
xx = linspace(ax(1),ax(2),100);
hA = plot(xx,modelfun(paramsA,xx),'k-');
hB = plot(xx,modelfun(paramsB,xx),'k-');
```

```
if resnormA < resnormB
  set(hB,'LineStyle','--');
else
  set(hA,'LineStyle','--');
end
```

Iteration	Func-count	f(x)	Norm of step	First-order optimality	CG-iterations
0	4	350877		9.33e+05	
1	8	142784	10	2.06e+05	0
2	12	60521.2	20	5.36e+04	0
3	16	11504.9	34.9278	1.06e+05	0
4	20	3343.19	1.27111	6.45e+03	0
5	24	3343.19	1.87938	6.45e+03	0
6	28	3175.22	0.469846	1.96e+03	0
7	32	3093.01	0.939692	8.96e+03	0
8	36	2861.81	0.939692	7.94e+03	0
9	40	2660.09	0.939692	6.84e+03	0
10	44	2487.51	0.939692	5.98e+03	0
11	48	2487.51	1.87938	5.98e+03	0
12	52	2375.98	0.469846	1.27e+03	0
13	56	2268.77	0.939692	5.08e+03	0
14	60	2261.58	1.87938	1.63e+04	0
15	64	1973.86	0.469846	141	0
16	68	1892.51	0.939692	3.47e+03	0
17	72	1827.6	1.87938	1.17e+04	0
18	76	1672.24	1.87938	9.31e+03	0
19	80	1573.81	1.87938	7.98e+03	0
20	84	1511.9	1.87938	6.9e+03	0
21	88	1481.83	1.87938	6.06e+03	0
22	92	1464.5	0.733162	787	0
23	96	1464.25	0.00513487	0.14	0
24	100	1464.25	0.000323993	0.00168	0

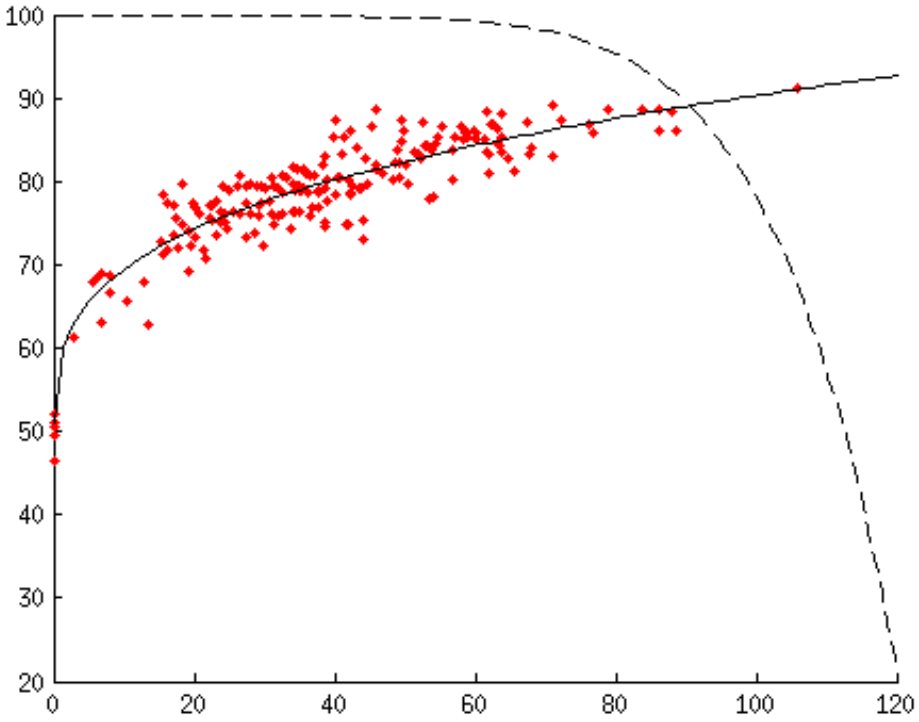
Local minimum possible.

lsqcurvefit stopped because the final change in the sum of squares relative to its initial value is less than the selected value of the function tolerance.

Iteration	Func-count	f(x)	Norm of step	First-order optimality	CG-iterations
0	4	3.10275e+30		9.99e+31	
1	8	3.28131e+13	10	1.01e+21	0
2	12	132342	0.178087	3.3e+16	0
3	16	97250.2	1.06352e-12	4.09e+03	0

Local minimum possible.

lsqcurvefit stopped because the size of the current step is less than the selected value of the step size tolerance.



Problem 4

```
% construct regressor matrix and data vector
X = score1(:);
X(:,end+1) = 1;
y = score2(:);

%%%%% beginning of the gradient descent routine

% define step size
stepsize = .01;

% initialize weights and other stuff
wcurrent = zeros(2,1);
errorold = Inf;  % this holds the squared error obtained previously

% loop until error starts to increase
while 1

  % compute the current squared error
  errorcurrent = sum((y-X*wcurrent).^2);

  % for debugging purposes:
  %fprintf('Current squared error = %.3f\n',errorcurrent);

  % if it is no better than the old squared error
  if errorcurrent >= errorold
    break;
```

```
  end

  % save the old results
  wold = wcurrent;
  errorold = errorcurrent;

  % compute gradient vector (dimensions are parameters x 1)
  thegrad = -2*X'*(y-X*wcurrent);

  % normalize it to be unit length by dividing by its length
  thegrad = thegrad / sqrt(sum(thegrad.^2));

  % update weights
  wcurrent = wcurrent - stepsize*thegrad;

end

% our final answer is the set of the weights before we took that final bad step
w = wold;

%%%%% end of the gradient descent routine

% inspect the weights
w
```

```
w =

         0.293208242564941
          67.0337894497326
```

```
% compare to the weights obtained by OLS
inv(X'*X)*X'*y
```

```
ans =

         0.298205522621571
          67.0339736111709
```

Published with MATLAB® R2012b