HW VI Solutions

EE 588: Optimization for the Information and Data Sciences

University of Southern California

(a) First read the data from Hill-Valley-without-noise-Training.data.txt, remove the header, and separate the features from the outputs. For matlab users you may find the importdata function useful.

```
A=importdata('Hill_Valley_without_noise_Training.data.txt',',',1,0);
size(A.data);
X1=A.data;

y=X1(:,end);
X=X1(:,1:100);
```

(b) Before we proceed any further we must normalize our data. This is the first thing to do when dealing with real data. Often the right form of normalization is to make sure our data set is zero mean and our features have the same Euclidean norm. More specifically, calculate the mean vector of patient features across all of the data set.

```
X = X - repmat(mean(X.').',1,size(X,2));
X = normr(X)                            ;
```

(c) Now that we have normalized data, partition it into train/test sets at random 100 times as discussed earlier. In each trial learn the model by solving (1) with $\lambda = 0.01$. To due this run gradient descent for $T = 500$ iterations and then use the trained model to make predictions on the test data and calculate the average error (average number of miss-classified patients on the test data) for each trial. Report the average over the 100 trials. The value of the step size you use does not matter too much. However, make sure that the algorithm has converged.

The average error (averaged over the 100 trials) is 2.8986e-04. The code:

```
clear all
close all
clc

A=importdata('Hill_Valley_without_noise_Training.data.txt',',',1,0);
size(A.data);
X1=A.data;

y=X1(:,end);
X=X1(:,1:100);
```

```
  X = X - repmat(mean(X.').',1,size(X,2)) ;
  X = normr(X)                            ;

  Xones  = [ ones( size(X,1) , 1 ) , X ] ;

  numtrial = 100 ;

  meanerrs = zeros(1,numtrial) ;


for trial = 1 :numtrial

    indall            = randperm( size(X,1) ) ;
    indtest           = indall(1:69)          ;
    indtrain          = indall                ;
    indtrain(indtest) = [ ]                   ;

    Xtrain = Xones( indtrain , : ) ;
    Xtest  = Xones( indtest  , : ) ;

    ytrain = y( indtrain , 1 ) ;
    ytest  = y( indtest  , 1 ) ;

    XtrainT = Xtrain.' ;

    theta     = zeros( size(X,2) + 1 , 1 ) ;
    thetaprev = zeros( size(X,2) + 1 , 1 ) ;

    lambda = 0.01 ;
    T      = 500  ;
    mu     = 0.01 ;


    errs = zeros(T,1) ;

    for t = 1 : T


  yvals = ( 1 ) ./ (1 + exp( - Xtrain * theta ) ) ;
  gradf = - XtrainT*ytrain + XtrainT*yvals + ...
      lambda * [ 0 ; theta(2:end) ] ;

  f = sum( - ( Xtrain*theta ).* ytrain + ...
      log( 1 + exp( Xtrain*theta ) )+lambda/2*norm(theta(2:end))^2 ) ;

  thetaprev = theta             ;
  theta     = theta - mu*gradf ;


      ypredreg = ( 1 ) ./ (1 + exp( - Xtest * theta ) ) ;

     errs(t) = norm(gradf)^2/(1+abs(f)) ;
```

```
    end

    ypredreg = ( 1 ) ./ (1 + exp( - Xtest * theta ) ) ;

    ypred = ypredreg ;

    ypred( ypredreg >= 0.5 ) = 1 ;
    ypred( ypredreg <  0.5 ) = 0 ;

    meanerrs(trial)=sum( ypred ~= ytest ) / length(ypred ) ;

end

mean(meanerrs)
```

(d) Perform the experiment with the same step size you used before but now report the number of iterations it takes to get to an accuracy of 10^{-6} calculated via the first iteration t when the following inequality holds

$$\|\nabla f(\boldsymbol{w}_t, b_t)\|_{\ell_2}^2 \leq 10^{-6} \left(1 + |f(\boldsymbol{w}_t, b_t)|\right). \tag{4}$$

The number you should report is the average of this number over the 100 trials.

The average number of iterations to get to an accuracy of 10^{-6} (averaged over the 100 trials) is 4.1581e+03. The code:

```
clear all
close all
clc

A=importdata('Hill_Valley_without_noise_Training.data.txt',',',1,0);
size(A.data);
X1=A.data;

y=X1(:,end);
X=X1(:,1:100);


X = X - repmat(mean(X.').',1,size(X,2)) ;
X = normr(X)                            ;

Xones  = [ ones( size(X,1) , 1 ) , X ] ;

numtrial = 100 ;
```

```
iterstop = zeros(1,numtrial) ;

for trial = 1 :numtrial

    indall            = randperm( size(X,1) ) ;
    indtest           = indall(1:69)          ;
    indtrain          = indall                ;
    indtrain(indtest) = [ ]                   ;

    Xtrain = Xones( indtrain , : ) ;
    Xtest  = Xones( indtest  , : ) ;

    ytrain = y( indtrain , 1 ) ;
    ytest  = y( indtest  , 1 ) ;

    XtrainT = Xtrain.' ;

    theta     = zeros( size(X,2) + 1 , 1 ) ;
    thetaprev = zeros( size(X,2) + 1 , 1 ) ;

    lambda = 0.01  ;
    T      = 10000 ;
    mu     = 0.01  ;


    errs = zeros(T,1) ;

    for t = 1 : T




    yvals = ( 1 ) ./ (1 + exp( - Xtrain * theta ) ) ;
    gradf = - XtrainT*ytrain + XtrainT*yvals + ...
        lambda * [ 0 ; theta(2:end) ] ;

    f = sum( - ( Xtrain*theta ).* ytrain + ...
        log( 1 + exp( Xtrain*theta ) )+lambda/2*norm(theta(2:end))^2 ) ;

    thetaprev = theta            ;
    theta     = theta - mu*gradf ;


        ypredreg = ( 1 ) ./ (1 + exp( - Xtest * theta ) ) ;
```

```
        errs(t) = norm(gradf)^2/(1+abs(f)) ;

        if rem(t,1000)==0
        disp(['t = ',num2str(t), ', error = ',num2str(errs(t))]) ;
        end

    end

    myind = find(errs<1e-6) ;

    iterstop(trial) = myind(1) ;

end

mean(iterstop)
```

(e) Perform the experiment of part (d) but now add a momentum term (1) using the heavy ball method and (2) using Nesterov's accelerated scheme. In both cases keep the same step size as part (d) but fine tune the momentum parameter to get the smallest number of iterations for convergence based on the stopping criteria (4) (again averaged over the 100 trials). Draw the convergence of the three algorithms gradient descent, heavy ball, Nesterov's accelerated scheme for one trial. That is, draw the ratio

$$\frac{\|\nabla f(\boldsymbol{w}_t, b_t)\|_{\ell_2}^2}{(1+|f(\boldsymbol{w}_t, b_t)|)},$$

as a function of the iteration number $t = 1, 2, \ldots, 500$. Which algorithm would you use and why?

(1) using heavy ball method best tuning parameter is $\beta = 0.91$ and the corresponding average number of iterations to get to an accuracy of 10^{-6} (averaged over the 100 trials) is 371.2. The code

```
clear all
close all
clc

acceleratedopt = 1 ;

A=importdata('Hill_Valley_without_noise_Training.data.txt',',',1,0);
size(A.data);
X1=A.data;

y=X1(:,end);
X=X1(:,1:100);

X = X - repmat(mean(X.').',1,size(X,2)) ;
X = normr(X)                            ;
```

```
Xones  = [ ones( size(X,1) , 1 ) , X ] ;

numtrial = 100 ;

betavals     = 0.89:0.01:0.99 ;

iterstop = zeros(1,numtrial) ;

iterstopbeta = zeros(size(betavals)) ;

indbeta = 0 ;
for beta = betavals

    indbeta = indbeta + 1

for trial = 1 :numtrial

    indall            = randperm( size(X,1) ) ;
    indtest           = indall(1:69)          ;
    indtrain          = indall                ;
    indtrain(indtest) = [ ]                   ;

    Xtrain = Xones( indtrain , : ) ;
    Xtest  = Xones( indtest  , : ) ;

    ytrain = y( indtrain , 1 ) ;
    ytest  = y( indtest  , 1 ) ;

    XtrainT = Xtrain.' ;

    theta     = zeros( size(X,2) + 1 , 1 ) ;
    thetaprev = zeros( size(X,2) + 1 , 1 ) ;

    lambda = 0.01 ;
    T      = 1000 ;
    mu     = 0.01 ;


    errs = zeros(T,1) ;

    for t = 1 : T
```

```
        if acceleratedopt ~=1

            yvals = ( 1 ) ./ (1 + exp( - Xtrain * theta ) ) ;
            gradf = - XtrainT*ytrain + XtrainT*yvals + ...
                lambda * [ 0 ; theta(2:end) ] ;

            f     = sum( - ( Xtrain*theta ).* ytrain + ...
                log( 1 + exp( Xtrain*theta ) )+ ...
                lambda/2*norm(theta(2:end))^2 ) ;

            thetaprev = theta            ;
            theta     = theta - mu*gradf ;

        else

            z     = theta + beta*(theta-thetaprev) ;
            yvalsz = ( 1 ) ./ (1 + exp( - Xtrain * z ) ) ;
            gradfz = - XtrainT*ytrain + XtrainT*yvalsz + ...
                lambda * [ 0 ; z(2:end) ] ;

            yvals = ( 1 ) ./ (1 + exp( - Xtrain * theta ) ) ;
            gradf = - XtrainT*ytrain + XtrainT*yvals + ...
                lambda * [ 0 ; theta(2:end) ] ;

            thetaprev = theta ;
            theta     = z - mu*gradf ;

            f     = sum( - ( Xtrain*theta ).* ytrain + ...
                log( 1 + exp( Xtrain*theta ) )+...
                lambda/2*norm(theta(2:end))^2 ) ;

        end

        ypredreg = ( 1 ) ./ (1 + exp( - Xtest * theta ) ) ;

        errs(t) = norm(gradf)^2/(1+abs(f)) ;

    end

    myind = find(errs<1e-6) ;

    iterstop(trial) = myind(1) ;

end
```

```
    iterstopbeta(indbeta) = mean(iterstop) ;

    end

    [val,ind] = min(iterstopbeta)

    betavals(ind)
```

(2) using Nesterov method best tuning parameter is $\beta = 0.91$ and the corresponding average number of iterations to get to an accuracy of 10^{-6} (averaged over the 100 trials) is 345.7000. The code:

```
clear all
close all
clc

acceleratedopt = 1 ;

A=importdata('Hill_Valley_without_noise_Training.data.txt',',',1,0);
size(A.data);
X1=A.data;

y=X1(:,end);


X=X1(:,1:100);



X = X - repmat(mean(X.').',1,size(X,2)) ;
X = normr(X)                            ;

Xones = [ ones( size(X,1) , 1 ) , X ] ;

numtrial = 100 ;

betavals      = 0.89:0.01:0.99 ;

iterstop = zeros(1,numtrial) ;

iterstopbeta = zeros(size(betavals)) ;
```

```
indbeta = 0 ;
for beta = betavals

    indbeta = indbeta + 1

for trial = 1 :numtrial

    indall            = randperm( size(X,1) ) ;
    indtest           = indall(1:106)          ;
    indtrain          = indall                 ;
    indtrain(indtest) = [ ]                    ;

    Xtrain = Xones( indtrain , : ) ;
    Xtest  = Xones( indtest  , : ) ;

    ytrain = y( indtrain , 1 ) ;
    ytest  = y( indtest  , 1 ) ;

    XtrainT = Xtrain.' ;

    theta     = zeros( size(X,2) + 1 , 1 ) ;
    thetaprev = zeros( size(X,2) + 1 , 1 ) ;

    lambda = 0.01 ;
    T      = 1000 ;
    mu     = 0.01 ;


   errs = zeros(T,1) ;

   for t = 1 : T




        if acceleratedopt ~=1

            yvals = ( 1 ) ./ (1 + exp( - Xtrain * theta ) ) ;
            gradf = - XtrainT*ytrain + XtrainT*yvals + ...
                lambda * [ 0 ; theta(2:end) ] ;

            f     = sum( - ( Xtrain*theta ).* ytrain + ...
                log( 1 + exp( Xtrain*theta ) )+...
                lambda/2*norm(theta(2:end))^2 ) ;

            thetaprev = theta             ;
            theta     = theta - mu*gradf ;
```

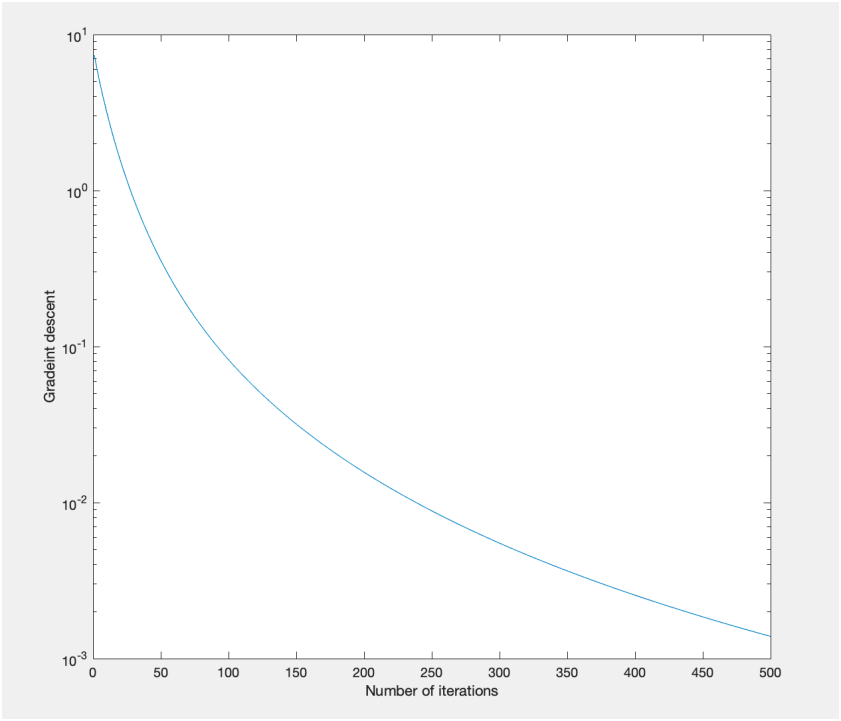
```
    else

        z      = theta + beta*(theta-thetaprev) ;
        yvalsz = ( 1 ) ./ (1 + exp( - Xtrain * z ) ) ;
        gradfz = - XtrainT*ytrain + XtrainT*yvalsz + ...
            lambda * [ 0 ; z(2:end) ] ;

        yvals = ( 1 ) ./ (1 + exp( - Xtrain * theta ) ) ;
        gradf = - XtrainT*ytrain + XtrainT*yvals + ...
            lambda * [ 0 ; theta(2:end) ] ;

        thetaprev = theta ;
        theta     = z - mu*gradfz ;

        f      = sum( - ( Xtrain*theta ).* ytrain + ...
            log( 1 + exp( Xtrain*theta ) )+...
        lambda/2*norm(theta(2:end))^2 ) ;

    end

    ypredreg = ( 1 ) ./ (1 + exp( - Xtest * theta ) ) ;

    errs(t) = norm(gradf)^2/(1+abs(f)) ;

end


    myind = find(errs<1e-6) ;

    iterstop(trial) = myind(1) ;

end

iterstopbeta(indbeta) = mean(iterstop) ;

end

[val,ind] = min(iterstopbeta)

betavals(ind)
```

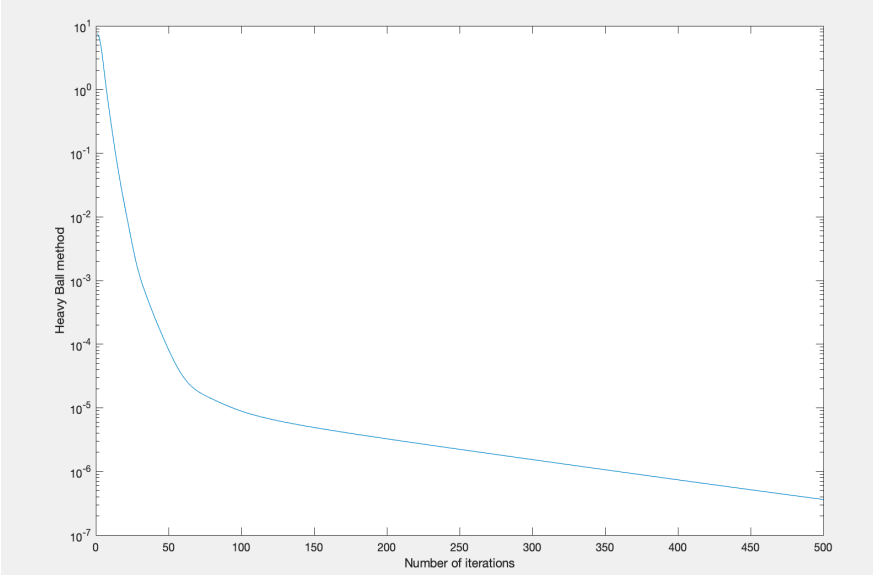
(3) The code:

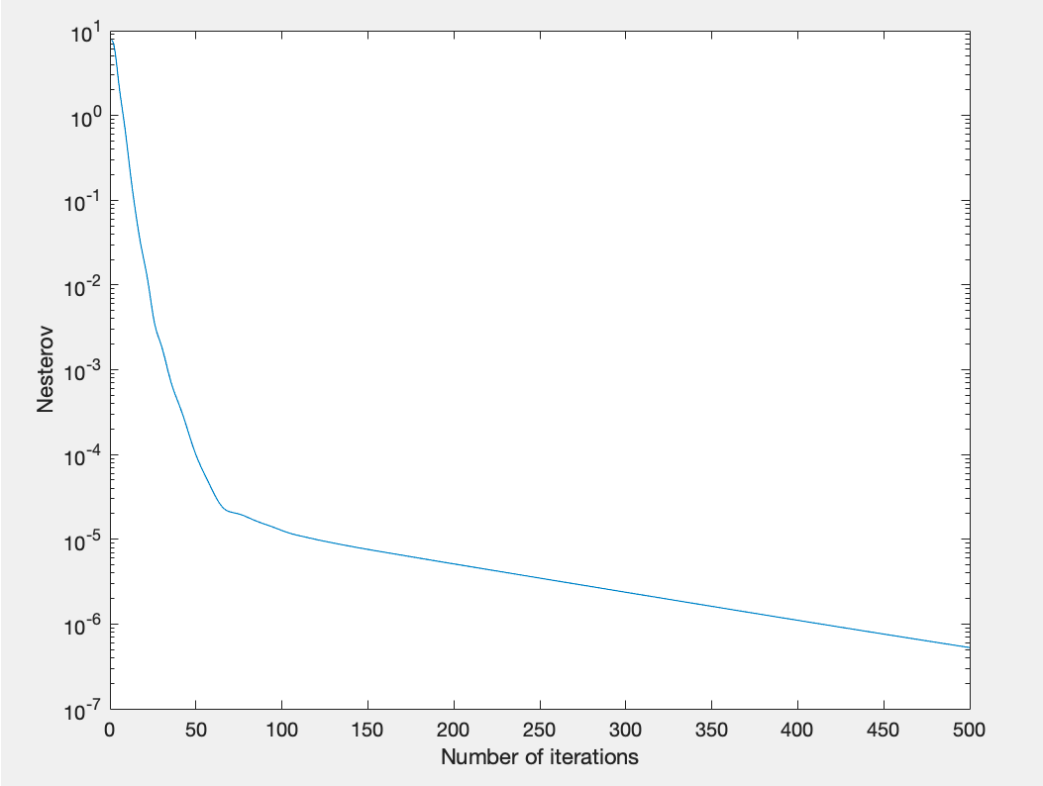
```
clear all
close all
clc



A=importdata('Hill_Valley_without_noise_Training.data.txt',',',1,0);
size(A.data);
X1=A.data;

y=X1(:,end);

X=X1(:,1:100);

X = X - repmat(mean(X.')',1,size(X,2)) ;
X = normr(X)                           ;

Xones  = [ ones( size(X,1) , 1 ) , X ] ;

numtrial = 100 ;

beta     = 0.91 ;

T      = 500 ;

errstot = zeros(T,3) ;

indopt = 0 ;

for acceleratedopt = [0,1,2]

    indopt = indopt + 1 ;
    indall           = randperm( size(X,1) ) ;
    indtest          = indall(1:106)          ;
    indtrain         = indall                 ;
    indtrain(indtest) = [ ]                   ;

    Xtrain = Xones( indtrain , : ) ;
    Xtest  = Xones( indtest  , : ) ;

    ytrain = y( indtrain , 1 ) ;
    ytest  = y( indtest  , 1 ) ;

    XtrainT = Xtrain.' ;

    theta     = zeros( size(X,2) + 1 , 1 ) ;
    thetaprev = zeros( size(X,2) + 1 , 1 ) ;

    lambda = 0.01 ;
    mu     = 0.01 ;


    errs = zeros(T,1) ;

    for t = 1 : T
```

```
if acceleratedopt == 0

    yvals = ( 1 ) ./ (1 + exp( - Xtrain * theta ) ) ;
    gradf = - XtrainT*ytrain +...
        XtrainT*yvals + ...
        lambda * [ 0 ; theta(2:end) ] ;

    f     = sum( - ( Xtrain*theta ).* ytrain + ...
        log( 1 + exp( Xtrain*theta ) )+ ...
        lambda/2*norm(theta(2:end))^2 ) ;

    thetaprev = theta             ;
    theta     = theta - mu*gradf ;

elseif acceleratedopt == 1

    z     = theta + beta*(theta-thetaprev) ;
    yvalsz = ( 1 ) ./ (1 + exp( - Xtrain * z ) ) ;
    gradfz = - XtrainT*ytrain + ...
        XtrainT*yvalsz + ...
        lambda * [ 0 ; z(2:end) ] ;

    yvals = ( 1 ) ./ (1 + exp( - Xtrain * theta ) ) ;
    gradf = - XtrainT*ytrain + ...
        XtrainT*yvals + ....
        lambda * [ 0 ; theta(2:end) ] ;

    thetaprev = theta ;
    theta     = z - mu*gradfz ;

    f     = sum( - ( Xtrain*theta ).* ytrain + ...
        log( 1 + exp( Xtrain*theta ) )+...
        lambda/2*norm(theta(2:end))^2 ) ;

else

    z     = theta + beta*(theta-thetaprev) ;
    yvalsz = ( 1 ) ./ (1 + exp( - Xtrain * z ) ) ;
    gradfz = - XtrainT*ytrain +...
        XtrainT*yvalsz + ...
        lambda * [ 0 ; z(2:end) ] ;

    yvals = ( 1 ) ./ (1 + exp( - Xtrain * theta ) ) ;
```

```
            gradf = - XtrainT*ytrain + ...
                XtrainT*yvals + ...
            lambda * [ 0 ; theta(2:end) ] ;

            thetaprev = theta ;
            theta     = z - mu*gradf ;

            f     = sum( - ( Xtrain*theta ).* ytrain +...
                log( 1 + exp( Xtrain*theta ) )+...
                lambda/2*norm(theta(2:end))^2 ) ;


        end

        ypredreg = ( 1 ) ./ (1 + exp( - Xtest * theta ) ) ;

        errs(t) = norm(gradf)^2/(1+abs(f)) ;

    end

    errstot(:,indopt) = errs ;

end
figure(1)
semilogy(1:T, errstot(:,1))
xlabel('Number of iterations')
ylabel('Gradeint descent')
figure(2)
semilogy(1:T, errstot(:,2))
xlabel('Number of iterations')
ylabel('Heavy Ball method')
figure(3)
semilogy(1:T, errstot(:,3))
xlabel('Number of iterations')
ylabel('Nesterov')



dlmwrite('HW5eCompare',[ (1:T)', errstot ] ,'delimiter',' ');
```


Gradeint descent
Number of iterations


Heavy Ball method
Number of iterations



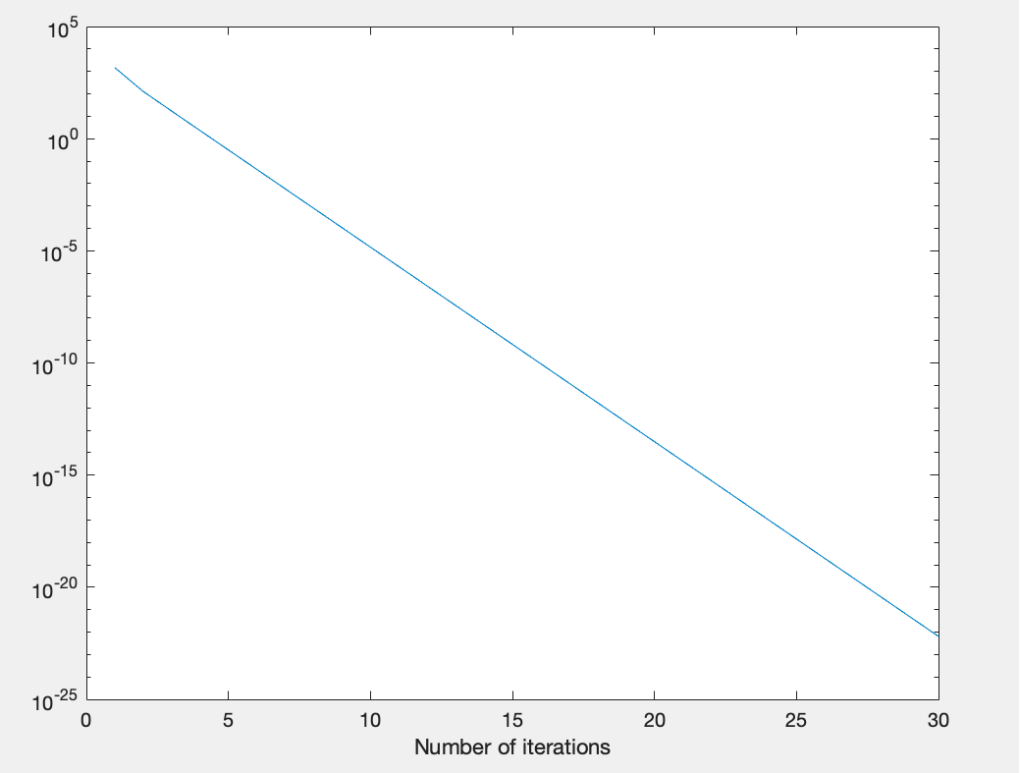
The convergence rate of the Heavy ball method is slightly faster. Given the minor difference, since we know that the Heavy ball method may not converge and we know that Nesterov's scheme does converge we should probably use Nesterov's scheme. ■

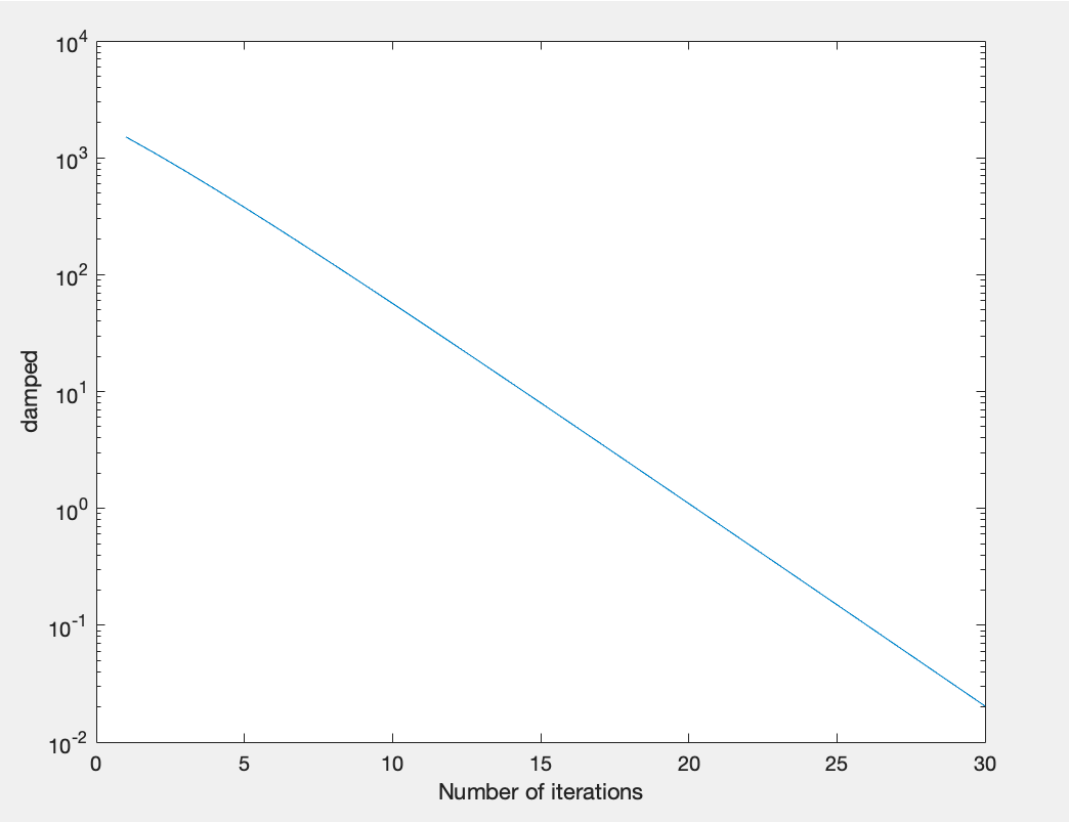
(f) Repeat the experiment of part (d) using Newton's method and damped Newton's method with appropriate step sizes. Then draw the ratio mentioned in part (e) as a function of the iteration number $t = 1, 2, ..., 500$.

```
clear all
close all
clc



A=importdata('Hill_Valley_without_noise_Training.data.txt',',',1,0);
size(A.data);
X1=A.data;

y=X1(:,end);
```

```
X=X1(:,1:100);



X = X - repmat(mean(X.').',1,size(X,2)) ;
X = normr(X)                            ;

Xones  = [ ones( size(X,1) , 1 ) , X ] ;

numtrial = 100 ;

beta     = 0.98 ;

T      = 500 ;

indall             = randperm( size(X,1) ) ;
indtest            = indall(1:106)          ;
indtrain           = indall                ;
indtrain(indtest) = [ ]                    ;

Xtrain = Xones( indtrain , : ) ;
Xtest  = Xones( indtest  , : ) ;

ytrain = y( indtrain , 1 ) ;
ytest  = y( indtest  , 1 ) ;



    lambda = 0.01 ;
    iter=30;
    w      = zeros( size(X,2), 1 ) ;
    b=.1;

    newton( Xtrain,ytrain );


function [theta, J ] = newton( x,y )

m = length(y);
ss=size(x,2);
theta = zeros(ss,1);
g = inline('1.0 ./ (1.0 + exp(-z))');
pos = find(y == 1);
neg = find(y == 0);
T=30;
J = zeros(T, 1);
%mu=.2;
```

```
 error=zeros(T,1);
 for num_iterations = 1:T

     h_theta_x = g(x * theta);

     pos_J_theta = -1 * log(h_theta_x(pos));
     neg_J_theta = -1 *log((1- h_theta_x(neg)));
     J(num_iterations) = sum([pos_J_theta;neg_J_theta])/m;

     delta_J = sum(repmat((h_theta_x - y),1,ss).*x);
     H = x'*(repmat((h_theta_x.*(1-h_theta_x)),1,ss).*x);

     theta = theta - pinv(H)*delta_J';
     error(num_iterations,1)=norm(delta_J)^2/(1+abs(J(num_iterations))) ;
 end
 delta_J
 size(delta_J)

figure;
semilogy(1:T, error(1:T))
xlabel('Number of iterations')

end
```




damped
Number of iterations
10^4
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
0
5
10
15
20
25
30