CS/ECE/ME 532
Homework 7: Convexity and the SVM

1. **Verifying convexity**. Prove that the following functions are convex.

a) The sum of two convex functions: $f(\boldsymbol{x}) = g(\boldsymbol{x}) + h(\boldsymbol{x})$ where g and h are convex.

SOLUTION:

$$\begin{aligned} f(t\boldsymbol{x} + (1-t)\boldsymbol{y}) &= g(t\boldsymbol{x} + (1-t)\boldsymbol{y}) + h(t\boldsymbol{x} + (1-t)\boldsymbol{y}) \\ &\le (tg(\boldsymbol{x}) + (1-t)g(\boldsymbol{y})) + (th(\boldsymbol{x}) + (1-t)h(\boldsymbol{y})) \\ &= t(g(\boldsymbol{x}) + h(\boldsymbol{x})) + (1-t)(g(\boldsymbol{y}) + h(\boldsymbol{y})) \\ &= tf(\boldsymbol{x}) + (1-t)f(\boldsymbol{y}) \end{aligned}$$

b) A positive quadratic form: $f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{P}\boldsymbol{x}$, where $\boldsymbol{P} \succ 0$.

SOLUTION: First, a little lemma. By expanding the following expression, we obtain: $(\boldsymbol{u}-\boldsymbol{v})^\top \boldsymbol{P}(\boldsymbol{u}-\boldsymbol{v}) = \boldsymbol{u}^\top\boldsymbol{P}\boldsymbol{u} + \boldsymbol{v}^\top\boldsymbol{P}\boldsymbol{v} - 2\boldsymbol{u}^\top\boldsymbol{P}\boldsymbol{v}$. Now rearrange and obtain:

$$\begin{aligned} 2\boldsymbol{u}^\top\boldsymbol{P}\boldsymbol{v} &= \boldsymbol{u}^\top\boldsymbol{P}\boldsymbol{u} + \boldsymbol{v}^\top\boldsymbol{P}\boldsymbol{v} - (\boldsymbol{u}-\boldsymbol{v})^\top\boldsymbol{P}(\boldsymbol{u}-\boldsymbol{v}) \\ &\le \boldsymbol{u}^\top\boldsymbol{P}\boldsymbol{u} + \boldsymbol{v}^\top\boldsymbol{P}\boldsymbol{v} \end{aligned}$$

where we used in the last step that $\boldsymbol{P} \succ 0$. Back to the original problem:

$$\begin{aligned} f(t\boldsymbol{x} + (1-t)\boldsymbol{y}) &= (t\boldsymbol{x} + (1-t)\boldsymbol{y})^\top \boldsymbol{P}(t\boldsymbol{x} + (1-t)\boldsymbol{y}) \\ &= t^2\boldsymbol{x}^\top\boldsymbol{P}\boldsymbol{x} + (1-t)^2\boldsymbol{y}^\top\boldsymbol{P}\boldsymbol{y} + 2t(1-t)\boldsymbol{x}^\top\boldsymbol{P}\boldsymbol{y} \\ &\le t^2\boldsymbol{x}^\top\boldsymbol{P}\boldsymbol{x} + (1-t)^2\boldsymbol{y}^\top\boldsymbol{P}\boldsymbol{y} + t(1-t)\left(\boldsymbol{x}^\top\boldsymbol{P}\boldsymbol{x} + \boldsymbol{y}^\top\boldsymbol{P}\boldsymbol{y}\right) \\ &= t\boldsymbol{x}^\top\boldsymbol{P}\boldsymbol{x} + (1-t)\boldsymbol{y}^\top\boldsymbol{P}\boldsymbol{y} \\ &= tf(\boldsymbol{x}) + (1-t)f(\boldsymbol{y}) \end{aligned}$$

Where the inequality follows from the lemma.

c) The pointwise maximum of several affine functions: $f(\boldsymbol{x}) = \max\limits_{i\in\{1,\dots,m\}} (\boldsymbol{a}_i^\top \boldsymbol{x} + b_i)$

SOLUTION:

$$\begin{aligned} f(t\boldsymbol{x} + (1-t)\boldsymbol{y}) &= \max_{i\in\{1,\dots,m\}} (\boldsymbol{a}_i^\top(t\boldsymbol{x} + (1-t)\boldsymbol{y}) + b_i) \\ &= \max_{i\in\{1,\dots,m\}} (t(\boldsymbol{a}_i^\top\boldsymbol{x} + b_i) + (1-t)(\boldsymbol{a}_i^\top\boldsymbol{y} + b_i)) \\ &\le t\max_{i\in\{1,\dots,m\}} (\boldsymbol{a}_i^\top\boldsymbol{x} + b_i) + (1-t)\max_{j\in\{1,\dots,m\}} (\boldsymbol{a}_j^\top\boldsymbol{y} + b_j) \\ &= tf(\boldsymbol{x}) + (1-t)f(\boldsymbol{y}) \end{aligned}$$

d) The definition of convexity we saw in class may also be extended to functions that take a matrix as an argument, e.g. $f : \mathbb{R}^{m\times n} \to \mathbb{R}$. Prove that $f(\boldsymbol{X}) = \|\boldsymbol{X}\|_2$ (the induced 2-norm) is convex.

SOLUTION:

$$\begin{aligned} f(t\boldsymbol{X} + (1-t)\boldsymbol{Y}) &= \|t\boldsymbol{X} + (1-t)\boldsymbol{Y}\|_2 \\ &\le t\|\boldsymbol{X}\|_2 + (1-t)\|\boldsymbol{Y}\|_2 \\ &= tf(\boldsymbol{X}) + (1-t)f(\boldsymbol{Y}) \end{aligned}$$

where the inequality follows from the triangle inequality.

2. Gradient Descent and Stochastic Gradient Descent. Suppose we have training data $\{\boldsymbol{x}_i, y_i\}_{i=1}^m$, with $\boldsymbol{x}_i \in \mathbb{R}^n$ and y_i is a scalar label. Derive gradient descent and SGD algorithms to solve the following ℓ_1-loss optimization:

$$\min_{\boldsymbol{w}\in\mathbb{R}^n} \sum_{i=1}^m \left| y_i - \boldsymbol{x}_i^\mathsf{T}\boldsymbol{w} \right| .$$

a) Simulate this problem as follows. Generate each x_i as random points in the interval $[0, 1]$ and generate $y_i = w_1 x_i \; + \; w_2 + \epsilon_i$, where w_1 and w_2 are the slope and intercept of a line (of your choice) and $\epsilon_i =$ `randn`, a Gaussian random error generated in Matlab. With $m = 10$. Repeat this experiment with several different datasets (with different random errors in each case).

SOLUTION: This is quite straightforward. Here is sample code.

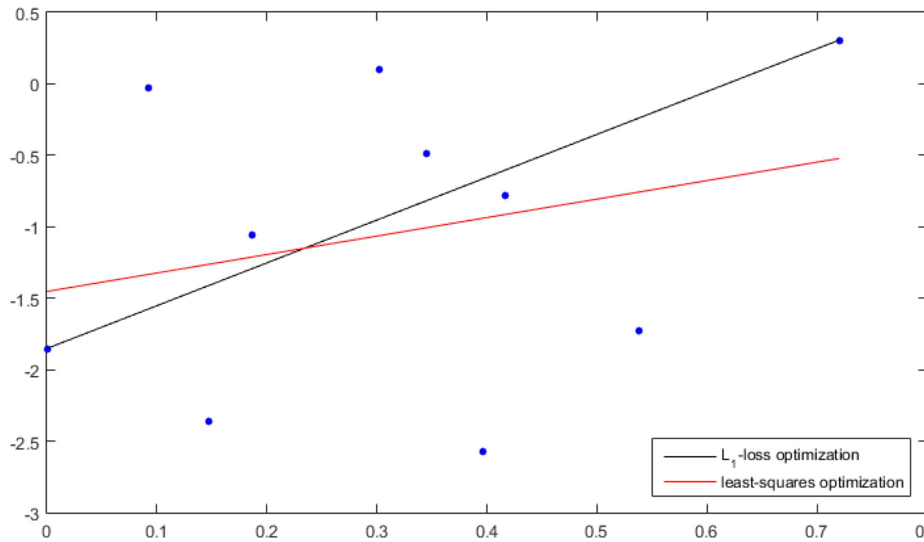
```
m = 10;
x = rand(m,1);
e = randn(m,1);
w = [1;-1];      % choice of slope and intercept
y = w(1)*x + w(2) + e;
plot( x, y ,'b.', 'MarkerSize', 15)
```

b) Implement the GD or SGD algorithm for the ℓ_1-loss optimization. Compare the solution to this optimization with the LS line fit.

SOLUTION: Here is an implementation of gradient descent for solving the ℓ_1-norm minimization problem. The code also finds the least-squares solution and plots both lines.

```
% use GD to compute the solution
w = rand(2,1);  % random initialization
MAX_ITER = 100000;
for j = 1:MAX_ITER
if mod(j,10000) == 0; disp(j); end  % progress indicator
eta = 1/sqrt(j+1);                  % diminishing stepsize
delta = 0;
for i = 1:m
delta = delta + sign(y(i)-w(1)*x(i)-w(2))*[x(i); 1];
end
w = w + eta*delta;
end

% compute least-squares solution
A = [x ones(m,1)];
wLS = A\y;
```

```
% plot solutions
figure(1); clf;
plot( x, y, 'b.', ...
      x, w(1)*x + w(2), 'k-', ...
      x, wLS(1)*x + wLS(2), 'r-', 'MarkerSize', 15 )

legend( 'L_1-loss optimization', 'least-squares optimization',...
'Location','southeast' )
```



The ℓ_1 solution touches two points perfectly because of the nature of the ℓ_1 cost function.

c) Now change the simulation as follows. Instead of generating ϵ_i as Gaussian, now generate the errors according to a Laplacian (two-sided exponential distribution) using `laprnd(1,1)`. Compare the LS and ℓ_1-loss solution compare in this case. Repeat this experiment with several different datasets (with different random errors in each case).

SOLUTION: The code is very similar in this case; simply change the noise `e=randn(m,1)` to `e=laprnd(m,1)` instead.

3. Error Bounds using Hinge Loss. State the SGD algorithm for solving the hinge-loss optimization

$$\min_{\boldsymbol{w}} \sum_{i=1}^{m} f_i(\boldsymbol{w}) \qquad \text{where:} \qquad f_i(\boldsymbol{w}) = \left(1 - y_i \boldsymbol{x}_i^{\mathsf{T}} \boldsymbol{w}\right)_+ .$$

a) Derive a bound on the average error $\frac{1}{T}\sum_{t=1}^{T}\left(f_{i_t}(\boldsymbol{w}_t) - f_{i_t}(\boldsymbol{w}^*)\right)$ using Theorem 1 from the lecture notes (on moodle). Assume that $\boldsymbol{w}_1 = \boldsymbol{0}$ and $\|\boldsymbol{w}^*\| \le 1$, and that the features are normalized so that $\|\boldsymbol{x}_i\| \le 1$ for all i. Assume a constant stepsize of $\gamma = 1/\sqrt{T}$ as in Corollary 1.

SOLUTION: Corollary 1 from the notes states that:

$$\frac{1}{T}\sum_{t=1}^{T}\left(f_t(\boldsymbol{w}_t) - f_t(\boldsymbol{w}^*)\right) \;\le\; \frac{\|\boldsymbol{w}_1 - \boldsymbol{w}^*\|_2^2 + G^2}{2\sqrt{T}} \qquad \text{for all } T$$

Since $\boldsymbol{w}_1 = \boldsymbol{0}$ and $\|\boldsymbol{w}^*\| \le 1$, we have $\|\boldsymbol{w}_1 - \boldsymbol{w}^*\|_2^2 \le 1$. As for the gradient,

$$\|\nabla f_i(\boldsymbol{w})\| = \left\| \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}} (1 - y_i \boldsymbol{x}_i^{\mathsf{T}} \boldsymbol{w})_+ \right\| \le \|y_i \boldsymbol{x}_i\| = \|\boldsymbol{x}_i\| \le 1$$

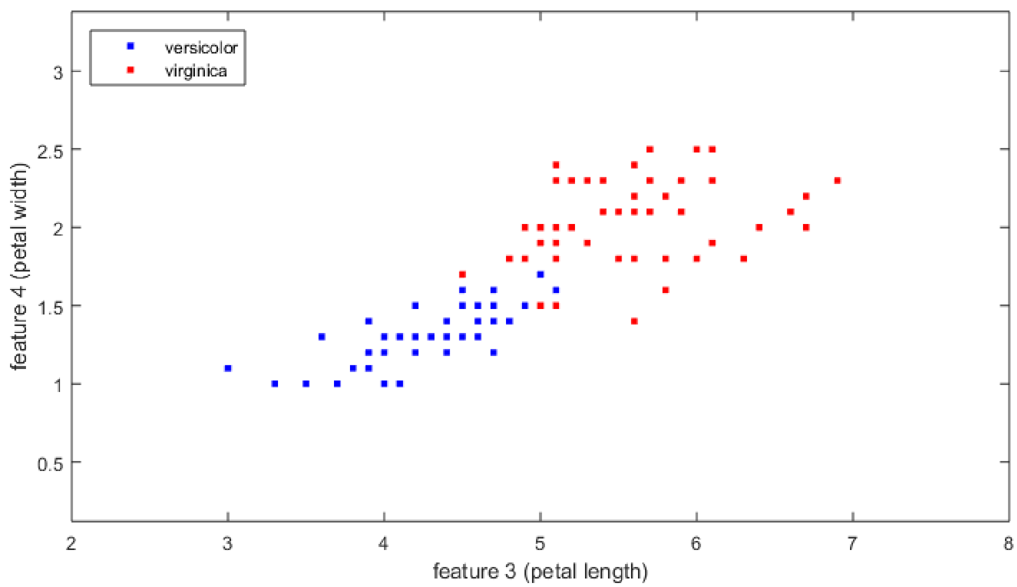
Therefore $G = 1$ and the bound we seek is:

$$\frac{1}{T}\sum_{t=1}^{T}\left(f_t(\boldsymbol{w}_t) - f_t(\boldsymbol{w}^*)\right) \;\le\; \frac{1}{\sqrt{T}} \qquad \text{for all } T$$

b) How many iterations are required to guarantee that the average error is less than 0.01?

SOLUTION: For the average error to be less than 0.01, we must have $1/\sqrt{T} \le 0.01$. This implies that $T \ge 100^2 = 10{,}000$. Every time we want to reduce the error by a factor of 10, we have to do 100 times more iterations... So for an error of 0.001, we could require up to 1,000,000 iterations. That's a lot of iterations!

4. Classification and the SVM. Revisit the `iris` data set from Homework 3. For this problem, we will use the 3rd and 4th features to classify whether an iris is *versicolor* or *virginica*. Here is a plot of the data set for this restricted set of features.



We will look for a linear classifier of the form: $x_{i3}w_1 + x_{i4}w_2 + w_3 \approx y_i$. Here, x_{ij} is the measurement of the j^{th} feature of the i^{th} iris, and w_1, w_2, w_3 are the weights we would like to find. The y_i are the labels; e.g. $+1$ for *versicolor* and -1 for *virginica*.

a) Reproduce the plot above, and also plot the decision boundary for the least squares classifier.

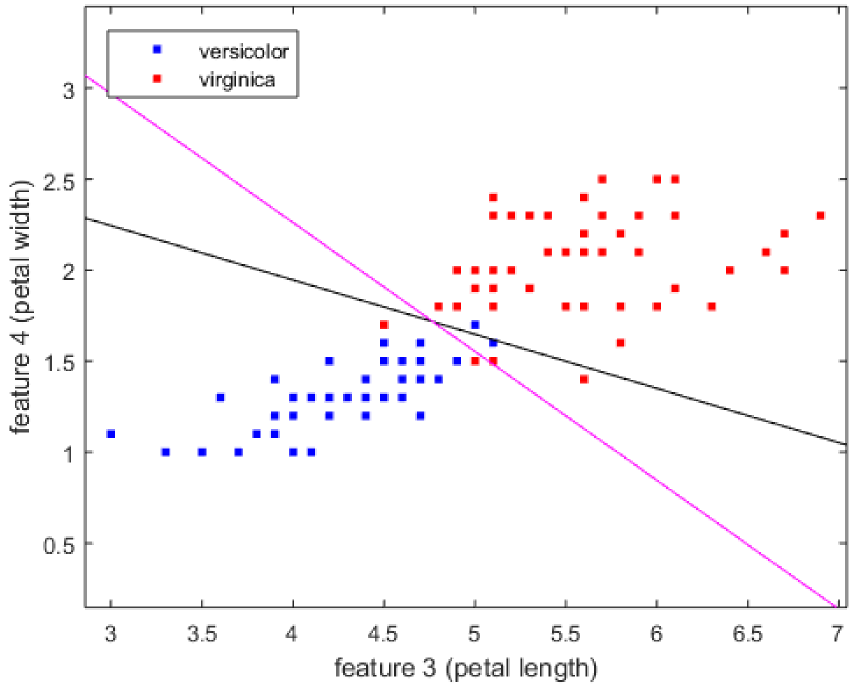
SOLUTION: See solution of part (b).

b) This time, we will use a regularized SVM classifier with the following loss function:

$$\underset{\boldsymbol{w}}{\text{minimize}} \quad \sum_{i=1}^{m} (1 - y_i \boldsymbol{x}_i^\mathsf{T} \boldsymbol{w})_+ + \lambda(w_1^2 + w_2^2)$$

Here, we are using the standard hinge loss, but with an ℓ_2 regularization that penalizes only w_1 and w_2 (we do not penalize the offset term w_3). Solve the problem by implementing gradient descent of the form $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \gamma \nabla f(\boldsymbol{w}_t)$. For your numerical simulation, use parameters $\lambda = 0.1$, $\gamma = 0.003$, $\boldsymbol{w}_0 = \mathbf{0}$ and $T = 20{,}000$ iterations. Plot the decision boundary for this SVM classifier. How does it compare to the least squares classifier?

SOLUTION: Here is the plot of the data, the least-squares classifier (black) and the SVM classifiers (magenta).



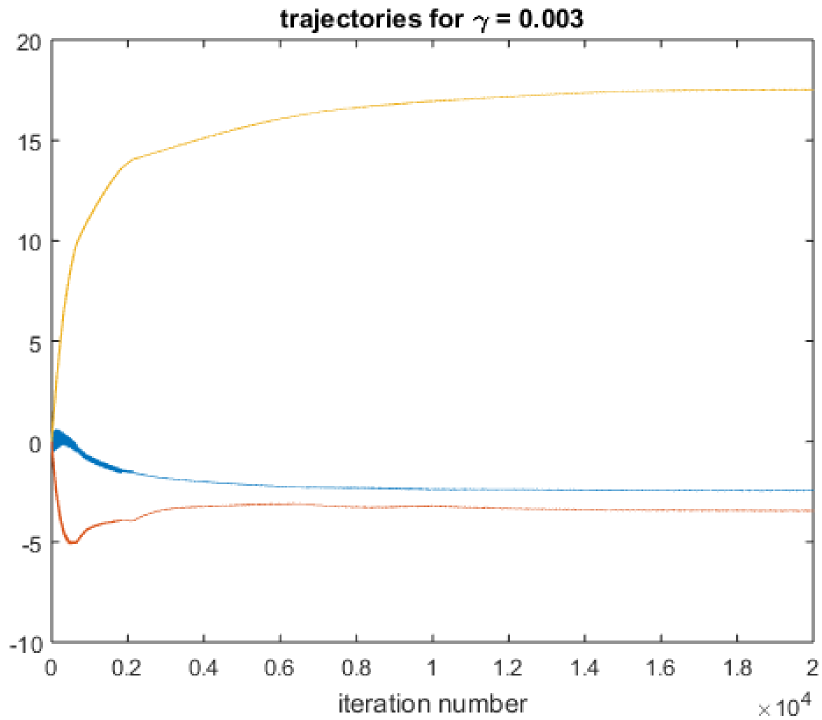
We can observe that both classifiers seem reasonable, though the SVM classifier does a better job (at least visually) of separating the data, and also achieves a slightly better classification error (4 incorrect classifications as opposed to 5 incorrect classifications for the LS classifier). Here is the code that produced these plots:

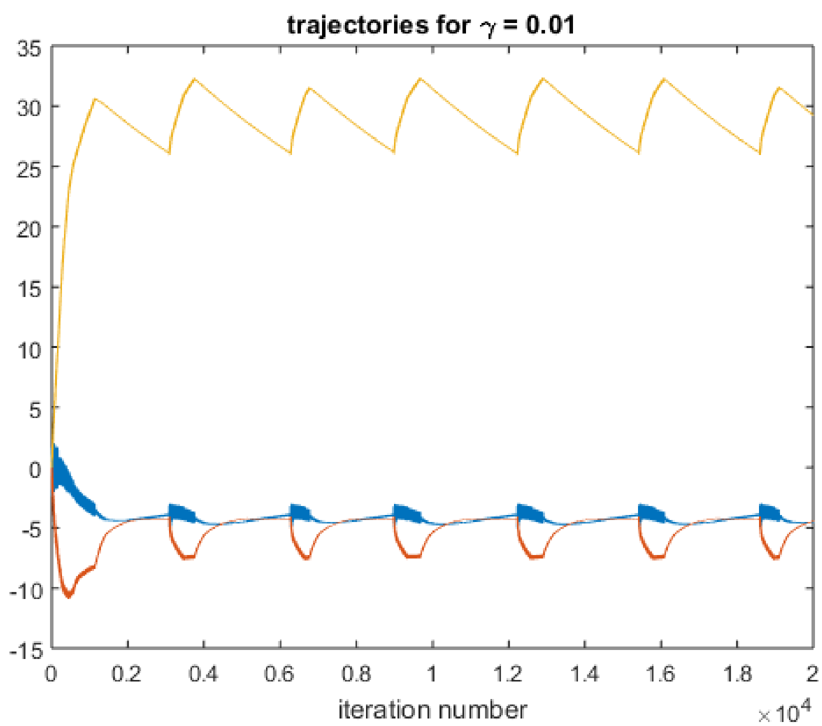
```
load fisheriris
inds = ~strcmp(species,'setosa');
A = meas(inds,[3 4]);
A = [A ones(size(A,1),1)];
y = species(inds);
b = zeros(size(y));

% get blues
figure(1); clf;
ib = find(strcmp(y,'versicolor'));
plot( A(ib,1), A(ib,2), 'b.', 'MarkerSize', 10 )
b(ib) = 1;

% get reds
ir = find(strcmp(y,'virginica'));
b(ir) = -1;

hold on
plot( A(ir,1), A(ir,2), 'r.', 'MarkerSize', 10 )
ax = axis;

legend('versicolor','virginica','Location','northwest')
xlabel('feature 3 (petal length)')
ylabel('feature 4 (petal width)')
axis equal; hold on;

%% solve using LS classifier
```

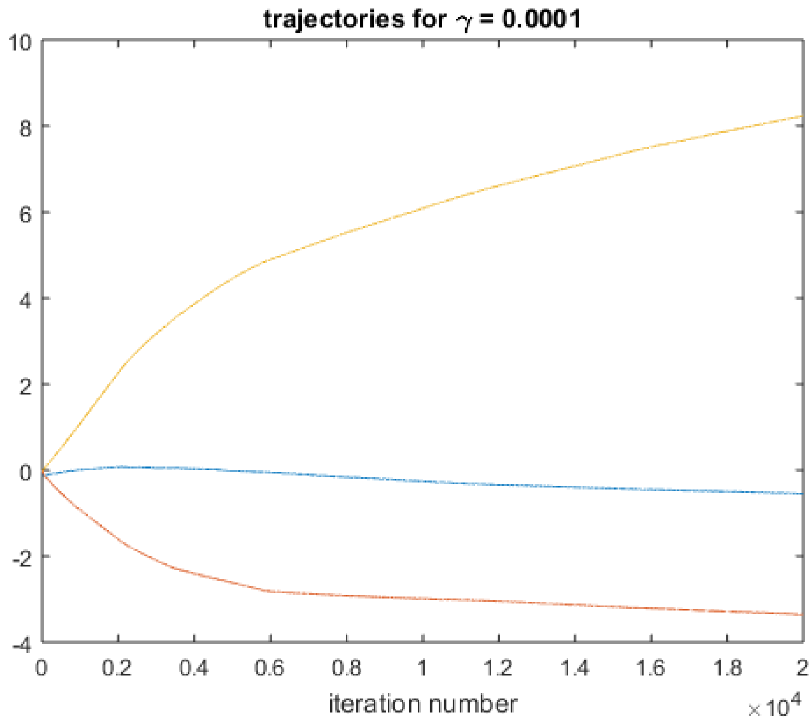
```
what = A\b;
plot( [0 -what(3)/what(1)], [-what(3)/what(2) 0], 'k-' );

% solve using SVM
w = [0;0;0];

lambda = .1;
N = 2e4;
tau = 0.003;

m = size(A,1);
wr = zeros(3,N);
fv = zeros(1,N);
for i = 1:N
    if ~mod(i,1e3)   % display iterations progress
        disp(i)
    end
    wr(:,i) = w;

    dir = 2*lambda*diag([1,1,0])*w;      % find descent direction
    for j = 1:m
        if b(j)*A(j,:)*w < 1
            dir = dir - b(j)*A(j,:)';
        end
    end
    fv(i) = sum(max(0,1-b.*(A*w)));
    w = w - tau*dir;
end

%% figs
plot( [0 -w(3)/w(1)], [-w(3)/w(2) 0], 'm-' );
figure(2);clf; plot(wr');   % plot trajectories as well
title('trajectories for \gamma = 0.003'); xlabel('iteration number')
```

c) Let's take a closer look at the convergence properties of $\boldsymbol{w}_t$. Plot the three components of $\boldsymbol{w}_t$ on the same axes, as a function of the iteration number t. Do the three curves each appear to be converging? Now produce the same plots with a larger stepsize ($\gamma = 0.01$) and a smaller stepsize ($\gamma = 0.0001$). What do you observe?

SOLUTION: See below for the three plots. We observe that the trajectories converge and settle nicely for $\gamma = 0.003$. When $\gamma = 0.01$, the trajectories oscillate (stepsize is too big). When $\gamma = 0.0001$, the trajectories still appear to be converging, but just much more slowly (stepsize can be increased).


trajectories for γ = 0.003
20
15
10
5
0
-5
-10
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
iteration number
$\times 10^4$


trajectories for γ = 0.01
35
30
25
20
15
10
5
0
-5
-10
-15
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
iteration number
$\times 10^4$


trajectories for γ = 0.0001
10
8
6
4
2
0
-2
-4
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
iteration number
$\times 10^4$