Assignment 4 Solutions
Robotics 811, Fall 2015

Problem 1: Consider the following differential equation over the interval $[0, 1]$:

$$\frac{dy}{dx} = \frac{2}{x^2(1-y)}, \qquad \textbf{with } y(1) = -1.$$

(a) Obtain an exact analytic solution $y(x)$ to the differential equation.

$$\begin{aligned} & \frac{dy}{dx} = \frac{2}{x^2(1-y)} \\ \Rightarrow \quad & (1-y)dy = \frac{2}{x^2}dx \\ \Rightarrow \quad & \int (1-y)dy = \int \frac{2}{x^2}dx \\ \Rightarrow \quad & y - \frac{1}{2}y^2 = -\frac{2}{x} + C \end{aligned} \tag{1}$$

Since $y(1) = -1$,

$$(-1) - \frac{1}{2}(-1)^2 = -2 + C \quad \Rightarrow \quad C = \frac{1}{2} \tag{2}$$

Putting this into Eq. (1)

$$\begin{aligned} \Rightarrow \quad & y - \frac{1}{2}y^2 = -\frac{2}{x} + \frac{1}{2} \\ \Rightarrow \quad & y^2 - 2y - \frac{4}{x} + 1 = 0 \\ \Rightarrow \quad & y = 1 \pm \sqrt{1 - (-\frac{4}{x} + 1)} = 1 \pm \frac{2}{\sqrt{x}} \end{aligned} \tag{3}$$

Again, since $y(1) = -1$,

$$y = 1 - \frac{2}{\sqrt{x}} \tag{4}$$

(b) Implement and use Euler's method to solve the differential equation numerically. Use a step size of 0.05. How accurate is your numerical solution? (Compare the numerical solution to the exact solution.)

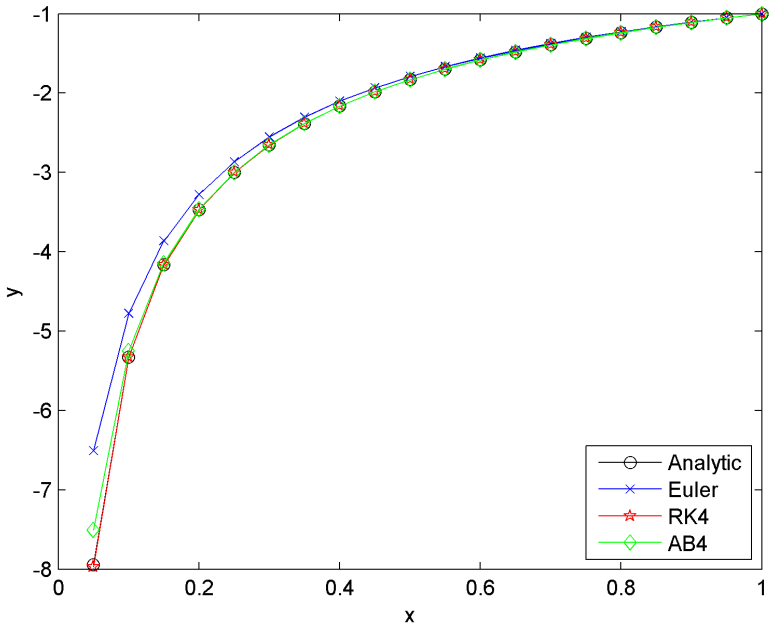
(c) Implement and use a fourth-order Runge-Kutta method to solve the differential equation numerically. Again, use a step size of 0.05. Again, how accurate is your numerical solution?

(d) Finally, implement and use fourth-order Adams-Bashforth for the differential equation. Again, use a step size of 0.05. Initialize the iteration with the following four values:

$$\begin{aligned} y(1.15) &= -0.865009616, \\ y(1.10) &= -0.906925178, \\ y(1.05) &= -0.951800146, \\ y(1) &= -1. \end{aligned}$$

Once again, how accurate is your numerical solution?

```
function HW4_problem1()

x_init = 1;
y_init = -1;
h = -0.05;
x = x_init:h:0;
n = length(x);

slope = @(x,y)(2/(x^2*(1-y)));

% (a) Alalytic method
y_Analytic = 1 - 2 ./ sqrt(x);

% (b) Euler's method
y_Euler = zeros(n, 1);
y_Euler(1) = y_init;

for i = 1 : n-1
y_Euler(i+1) = y_Euler(i) + slope(x(i),y_Euler(i)) * h;
end

% (c) fourth-order Runge-Kutta
y_RK4 = zeros(n, 1);
y_RK4(1) = y_init;

for i = 1 : n-1
k1 = slope(x(i),y_RK4(i));
k2 = slope((x(i)+h/2),(y_RK4(i)+k1*h/2));
k3 = slope((x(i)+h/2),(y_RK4(i)+k2*h/2));
k4 = slope((x(i)+h),(y_RK4(i)+k3*h));

y_RK4(i+1) = y_RK4(i) + (k1+2*k2+2*k3+k4)*h/6;
end

% (d) fourth-order Adams-Bashforth
kk(1) = slope(1.00, -1);
kk(2) = slope(1.05, -0.951800146);
kk(3) = slope(1.10, -0.906925178);
kk(4) = slope(1.15, -0.865009616);

y_AB4 = zeros(n, 1);
y_AB4(1) = y_init;

for i = 1 : n-1
y_AB4(i+1) = y_AB4(i) + (55*kk(1)-59*kk(2)+37*kk(3)-9*kk(4))*h/24;
kk = [ slope(x(i+1), y_AB4(i+1)) kk(1:end-1) ];
end

% plot all results
plot( x(1:end-1), y_Analytic(1:end-1), 'ko-', x(1:end-1), y_Euler(1:end-1), 'bx-', ...
x(1:end-1), y_RK4(1:end-1), 'rp-', x(1:end-1), y_AB4(1:end-1), 'gd-' );
legend( 'Analytic', 'Euler', 'RK4', 'AB4', 'Location', 'SouthEast' );

end
```

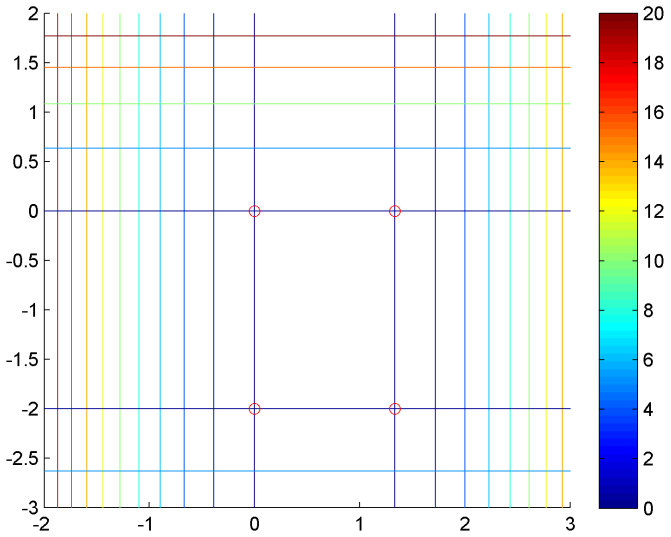


x	Analytic	Euler's	RK4	AB4
1.00	-1.0000	-1.0000	-1.0000	-1.0000
0.95	-1.0520	-1.0500	-1.0520	-1.0520
0.90	-1.1082	-1.1041	-1.1082	-1.1082
0.85	-1.1693	-1.1627	-1.1693	-1.1693
0.80	-1.2361	-1.2267	-1.2360	-1.2360
0.75	-1.3094	-1.2969	-1.3094	-1.3094
0.70	-1.3905	-1.3743	-1.3905	-1.3904
0.65	-1.4807	-1.4602	-1.4807	-1.4806
0.60	-1.5820	-1.5565	-1.5820	-1.5819
0.55	-1.6968	-1.6651	-1.6968	-1.6966
0.50	-1.8284	-1.7891	-1.8284	-1.8281
0.45	-1.9814	-1.9326	-1.9814	-1.9810
0.40	-2.1623	-2.1010	-2.1623	-2.1616
0.35	-2.3806	-2.3025	-2.3806	-2.3794
0.30	-2.6515	-2.5497	-2.6515	-2.6494
0.25	-3.0000	-2.8627	-3.0000	-2.9959
0.20	-3.4721	-3.2769	-3.4722	-3.4633
0.15	-4.1640	-3.8615	-4.1643	-4.1414
0.10	-5.3246	-4.7757	-5.3263	-5.2486
0.05	-7.9443	-6.5071	-7.9734	-7.5043
0.00	-Inf	-11.8354	-Inf	-15.4715

Problem 2: Consider the function $f(x, y) = x^3 + y^3 - 2x^2 + 3y^2 - 8$.

(a) Find all critical points of f by sketching in the (x, y) plane the iso-contours $\frac{\partial f}{\partial x} = 0$ and $\frac{\partial f}{\partial y} = 0$. Then classify the critical points into local minima, local maxima, and

saddle points by considering nearby gradient directions.



$$\frac{\partial f}{\partial x} = 3x^2 - 4x = 0 \quad \Rightarrow \quad x = \left\{0, \frac{4}{3}\right\}$$
$$\frac{\partial f}{\partial y} = 3y^2 + 6y = 0 \quad \Rightarrow \quad y = \{-2, 0\}$$

Thus, the critical points are $(0,-2), (0,0), (\frac{4}{3},-2)$, and $(\frac{4}{3},0)$.

The Hessian of f is

$$Hf = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix} = \begin{pmatrix} 6x-4 & 0 \\ 0 & 6y+6 \end{pmatrix}$$

Since Hf is a diagonal matrix, eigenvalues of Hf are $6x-4$ and $6y+6$.

At $(0,-2)$, $6x-4 = -4 < 0$ and $6y+6 = -6 < 0$.
Since both eigenvalues are less than 0, Hf is negative definite.
$\Rightarrow$ $(0,-2)$ is a local maximum.

At $(0,0)$, $6x-4 = -4 < 0$ and $6y+6 = 6 > 0$.
Since the signs of eigenvalues are different, Hf is neither positive definite nor negative definite.
$\Rightarrow$ $(0,0)$ is a saddle point.

At $(\frac{4}{3},-2)$, $6x-4 = 4 > 0$ and $6y+6 = -6 < 0$.
Since the signs of eigenvalues are different, Hf is neither positive definite nor negative definite.
$\Rightarrow$ $(\frac{4}{3},-2)$ is a saddle point.

At $(\frac{4}{3},0)$, $6x-4 = 4 > 0$ and $6y+6 = 6 > 0$.
Since both eigenvalues are greater than 0, Hf is positive definite.
$\Rightarrow$ $(\frac{4}{3},0)$ is a local minimum.

(b) Show how steepest descent would behave starting from the point $(x,y) = (1,-1)$. How many steepest descent steps are needed to converge to a local minimum?

The gradient of f is

$$\nabla f = \begin{pmatrix} 3x^2 - 4x \\ 3y^2 + 6y \end{pmatrix}$$

At the starting point $(1, -1)$,

$$\nabla f = \begin{pmatrix} -1 \\ -3 \end{pmatrix}$$

Therefore, steepest descent searches for the next point from $(1, -1)$ toward the direction $(-1, -3)$. The next point $(x', y') = (1, -1) - t(-1, -3) = (1 + t, -1 + 3t)$.

$$\begin{aligned} f(x', y') &= x'^3 + y'^3 - 2x'^2 + 3y'^2 - 8 \\ &= (1+t)^3 + (-1+3t)^3 - 2(1+t)^2 + 3(-1+3t)^2 - 8 \end{aligned}$$

$$\begin{aligned} \frac{df}{dt} &= 3(1+t)^2 + 9(-1+3t)^2 - 4(1+t) + 18(-1+3t) \\ &= 84t^2 + 2t - 10 = 0 \\ &\Rightarrow t = \frac{1}{3} \text{ or } -\frac{5}{14} \end{aligned}$$

Since $t > 0$, $t = \frac{1}{3}$ minimizes $f(x', y')$. Thus, the next point $(x', y') = (1+t, -1+3t) = (\frac{4}{3}, 0)$, which is a local minimum.

The steepest descent method finds a local minimum at the first iteration. This occurs because the steepest descent direction happens to be in the direction of the local minimum, so of course the minimum of that line search will be the actual local minimum.

Problem 3: Let Q be a real symmetric positive definite $n \times n$ matrix.

(a) Show that any two eigenvectors of Q corresponding to *distinct* eigenvalues of Q are Q-orthogonal. Show this directly from the definition of eigenvector.

Let the two eigenvectors of Q be v_1 and v_2 with corresponding eigenvalues λ_1 and λ_2. We then have

$$v_1^T Q v_2 = v_1^T \lambda_2 v_2 = \lambda_2 v_1^T v_2 \tag{5}$$

$$v_2^T Q v_1 = v_2^T \lambda_1 v_1 = \lambda_1 v_2^T v_1 \tag{6}$$

Since Q is symmetric,

$$(v_2^T Q v_1)^T = v_1^T Q^T v_2 = v_1^T Q v_2 \tag{7}$$

From Eqs. (5), (6), and (7),

$$\lambda_1 v_1^T v_2 = \lambda_1 (v_2^T v_1)^T = (v_2^T Q v_1)^T = v_1^T Q v_2 = \lambda_2 v_1^T v_2 \tag{8}$$

Since $\lambda_1 \neq \lambda_2$, from Eq. (5),

$$v_1^T v_2 = 0 \quad \Rightarrow \quad \lambda_2 v_1^T v_2 = 0 \quad \Rightarrow \quad v_1^T Q v_2 = 0 \tag{9}$$

(b) You will now enhance the argument from part (a) to establish Q-orthogonality more generally:

It is well known that the eigenvectors of Q can be chosen to be orthogonal in the usual sense. Using this fact, show that *any* two such eigenvectors of Q are in fact also Q-orthogonal.

From Eq. (5),

$$v_1^T v_2 = 0 \quad \Rightarrow \quad \lambda_2 v_1^T v_2 = 0 \quad \Rightarrow \quad v_1^T Q v_2 = 0 \tag{10}$$

Problem 4

(a) Show that in the purely quadratic form of the conjugate gradient method, $d_k^T Q d_k = -d_k^T Q g_k$. Using this show that to obtain x_{k+1} from x_k it is necessary to use Q only to evaluate g_k and Qg_k.

("Purely quadratic" means the conjugate gradient method applied to functions of the form $f(x) = c + b^T x + \frac{1}{2} x^T Q x$, where Q is a real symmetric positive definite $n \times n$ matrix.)

From the conjugate gradient method,

$$d_k = -g_k + \beta_{k-1} d_{k-1} \tag{11}$$

Multiplying both sides of Eq. (11) by $d_k^T Q$ yields,

$$d_k^T Q d_k = -d_k^T Q g_k + \beta_{k-1} d_k^T Q d_{k-1} \tag{12}$$

Since d_k and d_{k-1} are Q-orthogonal $d_k^T Q d_{k-1} = 0$; thus,

$$d_k^T Q d_k = -d_k^T Q g_k \tag{13}$$

The conjugate gradient method iteratively applies following updates

$$\alpha_k = -\frac{g_k^T d_k}{d_k^T Q d_k} \tag{14}$$

$$x_{k+1} = x_k + a_k d_k \tag{15}$$

$$g_{k+1} = Q x_{k+1} + b \tag{16}$$

$$\beta_k = \frac{g_{k+1}^T Q d_k}{d_k^T Q d_k} \tag{17}$$

$$d_{k+1} = -g_{k+1} + \beta_k d_k \tag{18}$$

Putting Eq. (13) into Eq. (14) gives

$$\alpha_k = \frac{g_k^T d_k}{d_k^T Q g_k} \tag{19}$$

From Eq. (17)

$$\beta_k = \frac{g_{k+1}^T Q d_k}{d_k^T Q d_k} = \frac{(g_{k+1}^T Q d_k)^T}{d_k^T Q d_k} = \frac{d_k^T Q^T g_{k+1}}{d_k^T Q d_k} = \frac{d_k^T Q g_{k+1}}{d_k^T Q d_k} \tag{20}$$

Putting Eq. (13) into Eq. (20) gives

$$\beta_k = -\frac{d_k^T Q g_{k+1}}{d_k^T Q g_k} \tag{21}$$

Considering modified updates in Eqs. (19), (15), (16), (21), and (18), we need to use Q only to evaluate g_k and Qg_k.

(b) Show that in the purely quadratic form Qg_k can be evaluated by taking a unit step from x_k in the direction of the negative gradient and then evaluating the gradient there. Specifically, if $y_k = x_k - g_k$ and $p_k = \nabla f(y_k)$, then $Qg_k = g_k - p_k$.

In the pure quadratic form, $g_k = \nabla f(x_k) = Qx_k + b$ and $p_k = \nabla f(y_k) = Qy_k + b$. Since $y_k = x_k - g_k$,

$$\begin{aligned} g_k - p_k &= (Qx_k + b) - (Qy_k + b) \\ &= (Qx_k + b) - (Q(x_k - g_k) + b) \\ &= Qg_k \end{aligned}$$

(c) Combine the results of parts (a) and (b) to derive a conjugate gradient method for general functions f much in the spirit of the algorithm presented in class, but which does not require knowledge of the Hessian of f or a line search. (Recall that a line search is used to find the minimum of f along a particular direction.)

In problem (a), we have following modified updates.

$$\alpha_k = \frac{g_k^T d_k}{d_k^T Q g_k} \tag{22}$$

$$x_{k+1} = x_k + a_k d_k \tag{23}$$

$$g_{k+1} = Qx_{k+1} + b \tag{24}$$

$$\beta_k = -\frac{d_k^T Q g_{k+1}}{d_k^T Q g_k} \tag{25}$$

$$d_{k+1} = -g_{k+1} + \beta_k d_k \tag{26}$$

From problem (b), $g_k = \nabla f(x_k)$, $p_k = \nabla f(x_k - g_k)$, and $Qg_k = g_k - p_k$. Putting these into the modified updates yields

$$\alpha_k = \frac{g_k^T d_k}{d_k^T (g_k - p_k)} \tag{27}$$

$$x_{k+1} = x_k + a_k d_k \tag{28}$$

$$g_{k+1} = \nabla f(x_{k+1}) \tag{29}$$

$$p_{k+1} = \nabla f(x_{k+1} - g_{k+1}) \tag{30}$$

$$\beta_k = -\frac{d_k^T (g_{k+1} - p_{k+1})}{d_k^T (g_k - p_k)} \tag{31}$$

$$d_{k+1} = -g_{k+1} + \beta_k d_k \tag{32}$$

In these updates, knowledge of the Hessian of f or a line search is not required.

Problem 5: In searching for the minimum of $f(x) = c + b^T x + \frac{1}{2} x^T Q x$, along the n Q-orthogonal directions $\{d_0, d_1, \ldots, d_{n-1}\}$, does the order of the directions matter? Is it necessary to search the same direction more than once? Explain your answers. (Assume that each line search locates its minimum exactly, and that Q is real symmetric positive definite.)

The order of directions does not matter since the minimum point can be written as a linear combination of the d_is. Also, the Expanding Subspace Theorem shows that minimizing along d_i does not undo the the minimization along d_j. Hence, we do not need to search along a direction more than once.

Problem 6: Find the rectangle of a given perimeter that has the greatest area, by solving the Lagrange multiplier first-order necessary conditions. Verify the second-order sufficiency conditions.

Let the two pairs of sides of the rectangle be of lengths x and y. Let the given perimeter be P. Then we have

$$F(x, y, \lambda) = xy + \lambda(P - 2(x + y)) \tag{33}$$

Solving for $\nabla F = 0$, we have

$$\frac{\partial F}{\partial x} = 0 \rightarrow x = 2\lambda \tag{34}$$

$$\frac{\partial F}{\partial y} = 0 \rightarrow y = 2\lambda \tag{35}$$

$$\frac{\partial F}{\partial \lambda} = 0 \rightarrow P = 2x + 2y \tag{36}$$

From the above equations we have $x = y = \frac{P}{4}$. To show that the point $(\frac{P}{4}, \frac{P}{4})$ is a maximum, we need to show that $L(x^*)$ is negative definite on the subspace M where $L(x^*) = \nabla^2 f(x^*) + \lambda \nabla^2 h(x^*)$ and M is the subspace $\{\ y : \nabla h(x^*)^T y = 0\ \}$.

We have

$$\nabla^2 f(x^*) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\nabla^2 h(x^*) = \begin{bmatrix} \frac{\partial^2 h}{\partial x^2} & \frac{\partial^2 h}{\partial x \partial y} \\ \frac{\partial^2 h}{\partial y \partial x} & \frac{\partial^2 h}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\rightarrow L(x^*) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Since $\nabla h(x) = \begin{bmatrix} -2, -2 \end{bmatrix}^T$, we have $M = \left\{ y : y = c \begin{bmatrix} -1, 1 \end{bmatrix}^T \right\}$. Thus, the second order sufficiency conditions are satisfied since $y^T L(x^*) y = -2c^2 < 0$ when y is nonzero.

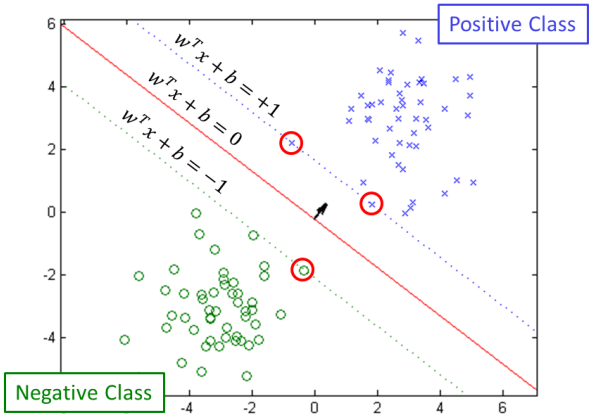


Figure 1: Figure depicting an SVM example

Problem 7: In this problem, you will explore how Lagrange multipliers are used to convert between the primal and dual formulations of the Support Vector Machine (SVM) algorithm. SVMs are a very common classification algorithm used in the machine learning field. The name is derived from the fact that from a whole data set, only a select few points (the "support vectors") are used to determine the decision boundary.

Suppose we have a set of l n-D points, $\{\mathbf{x}_1, \ldots, \mathbf{x}_l \in \mathbb{R}^n\}$. Suppose these points are split into two different classes and that we have the class labels for each of these points: $y_i \in \{-1, +1\}, i = 1, \ldots, l$. We want to draw a decision boundary between these points that can separate the -1 class from the $+1$ class. In particular, we want to draw the boundary such that the margin between the two classes is as large as possible.

You can see an example of this in Figure 1. Here, the blue x's and the green o's represent the positive and negative classes respectively. The decision boundary (red solid line) is represented by $\mathbf{w}^T\mathbf{x} + b = 0$, and the black vector shows the direction and magnitude of w. The margin is represented by the blue and green dotted lines, which correspond to $\mathbf{w}^T\mathbf{x} + b = +1$ and $\mathbf{w}^T\mathbf{x} + b = -1$ respectively. Here, 1 is chosen because it's a convenient number. The support vectors (circled in red) are the points that lie exactly on the margin.

Note that the distance between two hyperplanes of the form $\mathbf{w}^T\mathbf{x} + b = c_1$ and $\mathbf{w}^T\mathbf{x} + b = c_2$ is given by $D = |c_1 - c_2|/\|w\|$. Therefore, we can see that in order to correctly classify our points and maximize our margin, we need to solve the following constrained optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$
$$\text{s.t.} \quad \mathbf{w}^T\mathbf{x}_i + b \geq +1, \quad \text{if } y_i = +1$$
$$\mathbf{w}^T\mathbf{x}_i + b \leq -1, \quad \text{if } y_i = -1$$

In the case where our data is not linearly separable, we may want to allow our algorithm to make some mistakes during training time. That is, we want to allow $\mathbf{w}^T\mathbf{x}_i + b$ to be a little less than $+1$ for some of the positive training points or a little

greater than -1 for some of the negative ones. However, we want to minimize the total error of our algorithm during training. This results in the following problem formulation:

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i \tag{37}$$

$$\textbf{s.t.} \quad \mathbf{w}^T\mathbf{x}_i + b \geq +1 - \xi_i, \quad \textbf{if } y_i = +1 \tag{38}$$

$$\mathbf{w}^T\mathbf{x}_i + b \leq -1 + \xi_i, \quad \textbf{if } y_i = -1 \tag{39}$$

$$\xi_i \geq 0, \quad \forall i \tag{40}$$

Here, the ξ_i's represent the error we make on a given sample. C represents the trade off between making fewer mistakes and creating a larger margin.

(a) Convert the inequality constraints 38 and 39 found in the SVM optimization problem into an inequality constraint of the form $g_i\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T\right) \leq 0$ that does not involve "if" conditions. (Hint: use the value of the class labels, y_i)

Multiplying the inequality constraints 38 and 39 by their respective labels gives

$$y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq 1 - \xi_i, \quad \forall i$$

and we no longer need the "if" conditions. Moving terms around such that $g_i \leq 0$ gives

$$g_i\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T\right) = 1 - \xi_i - y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \leq 0, \quad \forall i$$

(b) Write down the Lagrangian for our optimization problem $\mathcal{L}\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T, \boldsymbol{\alpha}, \boldsymbol{\beta}\right)$.

We have

$$f\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T\right) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i$$

$$g_i\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T\right) = 1 - \xi_i - y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \leq 0$$

$$h_i\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T\right) = -\xi_i \leq 0$$

Plugging in, we have

$$\begin{aligned}\mathcal{L}\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T, \boldsymbol{\alpha}, \boldsymbol{\beta}\right) &= f\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T\right) + \sum_i \alpha_i g_i\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T\right) + \sum_i \beta_i h_i\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T\right) \\ &= \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i + \sum_i \alpha_i\left[1 - \xi_i - y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right)\right] - \sum_i \beta_i\xi_i\end{aligned}$$

(c) Recall from above that solving the SVM formulation directly as described above is equivalent to solving the following optimization problem:

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i\geq 0,\beta_i\geq 0} \mathcal{L}\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T, \boldsymbol{\alpha}, \boldsymbol{\beta}\right).$$

This is called the primal form of the SVM. The dual optimization problem is as follows

$$\max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i\geq 0,\beta_i\geq 0} \min_{\mathbf{w},b,\boldsymbol{\xi}} \mathcal{L}\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T, \boldsymbol{\alpha}, \boldsymbol{\beta}\right)$$

and solving it turns out to be equivalent to solving the primal.[1]

We want to solve the inner minimization term analytically, and then maximize over the new function numerically. To minimize the Lagrangian, set the partial derivatives with respect to w, b and ξ_i to zero. One of these equations will give you an expression for w. Plug that into the Lagrangian and simplify. The other two equations will help cancel things out. You should end up with a new Lagrangian, $\mathcal{L}^*$ in terms of only α_i's, y_i's and $\mathbf{x}_i$'s:

$$\mathcal{L}^*(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} \alpha_i\alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

and the constraints

$$\sum_{i=1}^{l} \alpha_i y_i = 0$$
$$0 \leq \alpha_i \leq C, \quad \forall i$$

Show your work.

$$0 = \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i$$
$$\implies \mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$
$$0 = \frac{\partial \mathcal{L}}{\partial b} = -\sum_i \alpha_i y_i$$
$$0 = \frac{\partial \mathcal{L}}{\partial \xi_i} = C - \alpha_i - \beta_i$$

[1] The two problems are not always equivalent, but they are equivalent in our case because the cost functions and the constraints are convex. You don't need to worry about why that is for this problem.

$$\begin{aligned}
\mathcal{L}\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T, \boldsymbol{\alpha}, \boldsymbol{\beta}\right) &= \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i + \sum_{i=1}^{l}\alpha_i\left[1-\xi_i - y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right)\right] - \sum_{i=1}^{l}\beta_i\xi_i \\
&= \frac{1}{2}\left(\sum_i \alpha_i y_i x_i\right)^T\left(\sum_i \alpha_i y_i x_i\right) + C\sum_{i=1}^{l}\xi_i \\
&\quad + \sum_{i=1}^{l}\alpha_i\left[1-\xi_i - y_i\left(\left(\sum_i \alpha_i y_i x_i\right)^T\mathbf{x}_i + b\right)\right] - \sum_{i=1}^{l}\beta_i\xi_i \\
&= \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j x_i^T x_j + \sum_i \alpha_i - \sum_{i,j}\alpha_i\alpha_j y_i y_j x_i^T x_j - \sum_i \alpha_i y_i b + \sum_{i=1}^{l}(C-\alpha_i-\beta_i)\xi_i \\
&= \sum_i \alpha_i + \left(\frac{1}{2}-1\right)\sum_{i,j}\alpha_i\alpha_j y_i y_j x_i^T x_j - b\cancelto{0}{\left(\sum_i \alpha_i y_i\right)} + \sum_{i=1}^{l}\cancelto{0}{(C-\alpha_i-\beta_i)}\xi_i \\
&= \sum_i \alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j x_i^T x_j
\end{aligned}$$

Since $C - \alpha_i - \beta_i = 0$, $\alpha_i \geq 0$, $\beta_i \geq 0$, we know that α_i can be at most C. Therefore, $0 \leq \alpha_i \leq C$.

(d) You will now implement your own SVM solver. Recall that we are trying to solve

$$\begin{aligned}
\max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i\geq 0,\beta_i\geq 0}\ \min_{\mathbf{w},b,\boldsymbol{\xi}} \mathcal{L}\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T, \boldsymbol{\alpha}, \boldsymbol{\beta}\right) &= \max_{\boldsymbol{\alpha}:0\leq\alpha_i\leq C}\mathcal{L}^*(\boldsymbol{\alpha}) \\
&= \max_{\boldsymbol{\alpha}:0\leq\alpha_i\leq C}\left(\sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j\right)
\end{aligned}$$

subject to $\sum_i \alpha_i y_i = 0$. This is a quadratic programming problem and you can solve it using the MATLAB `quadprog` function. Before you do this, you need to get our optimization problem into the right form:

Therefore, find $\mathbf{H}$ and $\mathbf{f}$ such that $\mathcal{L}^*(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{H}\boldsymbol{\alpha} + \mathbf{f}^T\boldsymbol{\alpha}$.

Solution:

$$\mathbf{H} = -\begin{bmatrix} y_1 y_1 x_1^T x_1 & \cdots & y_1 y_n x_1^T x_n \\ \vdots & \ddots & \vdots \\ y_n y_1 x_n^T x_1 & \cdots & y_n y_n x_n^T x_n \end{bmatrix}$$

$\mathbf{f}$ = a vector of 1s, length of α

(e) Now implement your own SVM solver. It should take n-D data, corresponding class labels, and C as input. It should output $\mathbf{w}$ and b.

```
function [w,b] = svm(x,y,C)
% This function trains an SVM classifier given:
% x – an dxn matrix, where d is the dimension of the data and n is the
%       number of samples
```

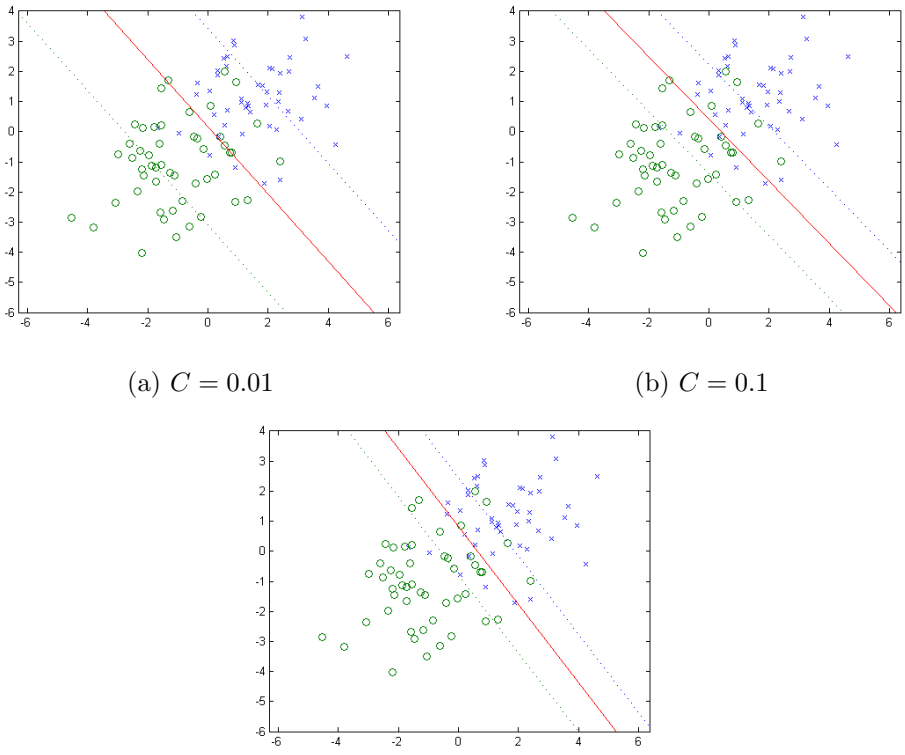
```
% y - an nx1 vector, which contains labels for the n data points
% C - a constant representing the trade-off between larger error and larger
%       margin

N = length(y);

% constructing relevant inputs to quadprog
H = (y*y').*(x'*x);
f = -ones(N,1);
A = [];
b = [];
Aeq = y';
beq = 0;
lb = zeros(N,1);
ub = C*ones(N,1);

% optimizing objective function
tic
disp('starting quadprog')
alpha = quadprog(H,f,A,b,Aeq,beq,lb,ub,[], optimset('Algorithm', ...
    'interior-point-convex', 'Display', 'off'));
toc
% finding support vectors
% (adding in an epsilon to account for machine precision)
epsilon = 1e-5;
SVs = (alpha > 0 + epsilon) & (alpha < C - epsilon);

% computing w and b
w = ((alpha.*y)'*x')';
b = mean(y(SVs)' - w'*x(:,SVs));
```

(i) 2d data is provided in two separate text files - one each for the positive (`2d_positive_class.txt`) and negative classes (`2d_negative_class.txt`). Implement your SVM solver using $C = 0.01, 0.1, 1$. Plot the data (using different symbols for the positive and negative classes), as well as your decision boundary $\{\mathbf{x} : \mathbf{w}^T\mathbf{x} + b = 0\}$, the positive margin $\{\mathbf{x} : \mathbf{w}^T\mathbf{x} + b = +1\}$ and the negative margin $\{\mathbf{x} : \mathbf{w}^T\mathbf{x} + b = -1\}$ for each value of C.

Solution:

```
% load data
pos = load('2d_positive_class.txt');
neg = load('2d_negative_class.txt');

% set up variables
data = [pos;neg]';
labels = [ones(size(pos,1),1); -ones(size(neg,1),1)];
C = [0.01,0.1,1];

for i = 1:3
    % train svm
    [w,b] = svm(data,labels,C(i));

    % plotting data
    figure;
    plot(pos(:,1),pos(:,2),'x',neg(:,1),neg(:,2),'o')
```

```
    hold on

    % getting axes nice
    xlim([2*floor(min(neg(:,1))/2) 2*ceil(max(pos(:,1))/2)])
    ylim([2*floor(min(neg(:,2))/2) 2*ceil(max(pos(:,2))/2)])
    axis equal
    xAxisLim = get(gca,'xlim');
    yAxisLim = get(gca,'ylim');

    x = xAxisLim;

    y = ( - b - w(1)*x)/w(2);            % decision boundary
    pos_marg = (1 - b - w(1)*x)/w(2);    % positive margin
    neg_marg = (-1 - b - w(1)*x)/w(2);   % negative margin

    % plotting decision boundary and margins
    plot(x,y,'-r',x,pos_marg,':',x,neg_marg,':');
    xlim(xAxisLim)
    ylim(yAxisLim)
end
```



(a) $C = 0.01$ (b) $C = 0.1$

(c) $C = 1$

Figure 2: SVM learned decision boundaries for $C = 0.01, 0.1, 1$

(ii) What happens to the margin as C decreases? What do you expect to happen as $C \to 0$? Use Eq. 37 and/or look at the dual cost function carefully to help with your reasoning.

Solution: As shown in Figure 2, the margin increases (gets wider) as C decreases. This is because C represents the penalty on the error we make for a given sample –

where the error $\xi_i = \max\left\{0, 1 - y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right)\right\}$. For large C, we want a margin such that all the positive points are above the positive margin ($\mathbf{w}^T\mathbf{x}_i + b \geq +1$) and all the negative points are below the negative margin ($\mathbf{w}^T\mathbf{x}_i + b \leq -1$). Since any points within the margins do not satisfy this criteria, they have non-zero error ξ_i, and are penalized.

However, as $C \to 0$, we care less and less about the error term on each point. The remaining term in the objective function is for maximizing the margin. Therefore, as $C \to 0$, the margin will become arbitrarily large, and the error term on each point will be non-zero (possibly very large), but we won't care since the penalty for these errors is approaching 0.

(iii) We have supplied you with a dataset consisting of 28x28 pixel images of handwritten digits that are converted into vectors of intensity by concatenating the columns of the images, resulting in a 784 dimensional vector. The data is split into two separate files, one for the ones (`handwriting_positive_class.txt`) and one for the zeros (`handwriting_negative_class.txt`). Each row in each file corresponds to a single image. We have supplied you with a MATLAB function (`displayImage.m`) that takes as input one of the image vectors and displays it for you. This data was taken from the MNIST dataset [2].

Now use your SVM solver on the handwriting dataset. You algorithm will learn to classify between the zeros and ones. Split your data into 10 parts (MATLAB function `crossvalind` is helpful for this). For each part, train on the other nine and use the tenth for testing. This is called 10-fold cross validation. Use $C = 1$.[3] Report your average accuracy.

Solution: The average accuracy was 0.9972. See implementation below.

```
% load data
pos = load('handwriting_positive_class.txt');
neg = load('handwriting_negative_class.txt');

% set up variables
data = [pos;neg]';
labels = [ones(size(pos,1),1); -ones(size(neg,1),1)];
C = 1;

% k-fold cross validation
k = 10;
indices = crossvalind('kfold',length(labels),k);
accuracy = zeros(k,1);
for i = 1:k
    i
    % train on 9/10 of data
    [w,b] = svm(data(:,indices~=i),labels(indices~=i),C);

    % test on 1/10 of the data
    test = (w'*data(:,indices==i) + b)';

    % number of indices where test label (sign) and ground truth label
    % agree:
    correct = sum(test.*labels(indices == i) >= 0);
```

[2] Data taken from `http://yann.lecun.com/exdb/mnist/`. Data converted into a more accessible format with help from `http://ufldl.stanford.edu/wiki/index.php/Using_the_MNIST_Dataset`

[3] In practice, you should optimize over C by using cross-validation.

```
25
26      % compute accuracy
27      accuracy(i) = correct/length(test);
28  end
29
30  % average accuracy
31  average_accuracy = mean(accuracy)
```

(iv) Visualize one of the $\mathbf{w}$'s you learned in part (iii) as an image (you can use the provided `displayImage.m` function). Comment on what your algorithm learned, and what it thinks is important.

Solution: Figure 3 shows one of the $\mathbf{w}$'s learned in part (iii). We can see that if pixels in the center, vertical stripe will be weighted positively, while those in the circular region around will be weighted negatively. Therefore, images with bright pixels in the "1" area will get a SVM detector score that is a sum of a lot of positive weights, while the opposite will be true for images with bright pixels in the "0" area. We can also see that pixels in the corner have a weight of ~ 0, meaning that these pixels are not discriminative.

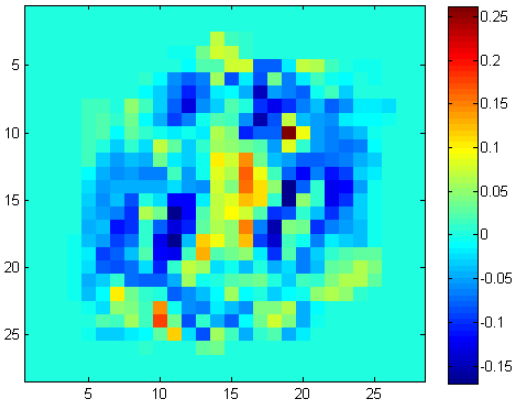


Figure 3: Sample learned $\mathbf{w}$