

# Esercizio 1

## Gradient method (exact line search)

```
1 Q=[6 0 -4 0;0 6 0 -4;-4 0 6 0;0 -4 0 6];
2 c = [ 1 -1 2 -3]';
3
4 disp('eigenvalues of Q')
5 eig(Q)
6
7 x0 = [0 0 0 0]';
8 tolerance = 10^(-6);
9 x = x0;
10 X=[];
11
12 for ITER=1:1000
13     v = 0.5*x'*Q*x + c'*x;
14     g = Q*x + c ;
15     X=[X;ITER,x',v,norm(g)];
16
17     if norm(g) < tolerance
18         break
19     end
20
21     d = -g;
22     t = norm(g)^2/(d'*Q*d) ;
23     x = x + t*d ;
24 end
25
26 disp(X)
```

## Gradient method (inexact line search)

```
1 alpha = 0.1; gamma = 0.9; tbar = 1;
2 x0 = [ 10 -10]';
3 tolerance = 10^(-3);
4 x = x0;
5 X=[];
6
7 for ITER=1:1000
8     [v, g] = f(x);
9
10    X=[X;ITER,x',v,norm(g)];
11
12    if norm(g) < tolerance
13        break
14    end
15
16    d = -g;
17    t = tbar ;
18
19    while (f(x+t*d) > v + alpha*g'*d*t)
20        t = gamma*t ;
21    end
22
23    x = x + t*d;
24 end
25
26 disp(X)
27
28 function [v, g] = f(x)
29 v = x(1)^4 + x(2)^4 - 2*x(1)^2 + 4*x(1)*x(2)-2*x(2)^2 ;
30 g = [4*x(1)^3-4*x(1)+4*x(2);
31      4*x(2)^3+4*x(1)-4*x(2)];
32 end
```

## Conjugate Gradient method

```
1 Q = [6 1 0 2; 1 2 0 0; 0 0 4 2; 2 0 2 8];
2 c = [-1; 8; 6; 9];
3
4 disp('Eigenvalues of Q:')
5 eig(Q)
6
7 x0 = [0 0 0 0]';
8 tolerance = 10^(-6);
9 x = x0;
10 X = [];
11
12 for ITER=1:10
13     v = 0.5*x'*Q*x + c'*x;
14     g = Q*x + c;
15
16     X = [X; ITER, x', v, norm(g)];
17
18     if norm(g) < tolerance
19         break
20     end
21
22     if ITER == 1
23         d = -g;
24     else
25         beta = (g'*Q*d_prev)/(d_prev'*Q*d_prev);
26         d = -g + beta*d_prev;
27     end
28
29     t = -(g'*d)/(d'*Q*d);
30
31     x = x + t*d;
32     d_prev = d;
33 end
34
35 disp(X)
```

## Penalty method (exact)

```
1 global A b eps;
2 A = [2 1 ; -1 -1 ; -1 0 ];
3 b = [ 4 ; -1 ; 0 ];
4 tau = 0.1;
5 eps0 = 5;
6 tolerance = 1e-6;
7
8 eps = eps0;
9 x = [4 0]';
10 X=[];
11
12 for ITER=1:1000
13     [x,pval] = fminunc(@p_eps,x);
14     infeas = max(A*x-b);
15
16     X=[X;ITER,eps,x',infeas,pval];
17
18     if infeas < tolerance
19         break
20     else
21         eps = tau*eps;
22     end
23 end
24
25 disp(X)
26
27 function v= p_eps(x)
28     global A b eps;
29     v = x(1)^2 -log(x(1)+x(2)) ;
30
31     for i = 1 : size(A,1)
32         v = v + (1/eps)*(max(0,A(i,:)*x-b(i)))^2;
33     end
34 end
```

## Logarithmic Barrier method

```
1  global Q c A b eps;
2  Q = [2 -1 0 ; -1 2 1; 0 1 2 ] ;
3  c = [-3 -4 -5]';
4  A = [2 1 1; -1 0 0; 0 1 0 ; 0 0 -1 ];
5  b = [20 -2 3 -4]';
6  eig(Q)
7  delta = 1e-3 ;
8  tau = 0.5 ;
9  eps1 = 1 ;
10 x0 = [3;2;5];
11
12 x = x0;
13 eps = eps1;
14 m = size(A,1);
15 X=[];
16
17 for ITER=1:1000
18     [x,pval] = fminunc(@logbar,x);
19     gap = m*eps;
20
21     X=[X;ITER,eps,x',gap,pval];
22
23     if gap < delta
24         break
25     else
26         eps = eps*tau;
27     end
28 end
29
30 disp(X)
31
32 function v = logbar(x)
33     global Q c A b eps
34     v = 0.5*x'*Q*x + c'*x ;
35
36     for i = 1 : length(b)
37         v = v - eps*log(b(i)-A(i,:)*x) ;
38     end
39 end
```

## Newton method

```
1  alpha=0.1; gamma=0.9; tbar =1;
2  x0 = [0 0]';
3  tolerance = 10^(-3) ;
4  x = x0;
5  X = [];
6
7  for ITER=1:100
8      [v, g, H] = f(x);
9      X=[X;ITER,x',v,norm(g)];
10
11      if norm(g) < tolerance
12          break
13      end
14
15      d = -inv(H)*g;
16      t=tbar;
17      while (f(x+t*d) > f(x) + alpha*t*d'*g)
18          t=gamma*t;
19      end
20
21      x = x + t*d;
22  end
23
24  disp(X)
25
26 function [v, g, H] = f(x)
27     v = 2*x(1)^4 + 3*x(2)^4 + 2*x(1)^2 + 4*x(2)^2 + x(1)*x(2);
28
29     g = [ 8*x(1)^3 + 4*x(1) + x(2) - 3;
30          12*x(2)^3 + 8*x(2) + x(1) - 2];
31
32     H = [ 24*x(1)^2+4 1;
33          1 36*x(2)^2+8];
34 end
```

## Esercizio 2

### Linear SVM with soft margin

```
1  A = [6.55 0.85; 6.55 1.71; 7.06 0.31; 2.76 0.46; 0.97 8.23; 9.5 0.34; 4
2  B = [9.59 3.40; 5.85 2.23; 7.51 2.55; 5.05 7; 8.9 9.59; 8.40 2.54; 8.14
3
4  nA = size(A,1);
5  nB = size(B,1);
6
7  T = [A; B];
8  C = 10;
9  y = [ones(nA,1); -ones(nB, 1)];
10 l = length(y);
11 Q = zeros(l,1);
12
13 for i = 1:l
14     for j = 1 : l
15         Q(i,j) = y(i)*y(j)*T(i,:)*T(j,:)' ;
16     end
17 end
18
19 la = quadprog(Q, -ones(l,1), [], [], y', 0, zeros(l,1), C*ones(l,1));
20
21 w = zeros(2,1);
22 for i = 1:l
23     w = w + la(i)*y(i)*T(i,:)' ;
24 end
25
26 indpos = find(la > 10^(-3));
27 ind = find(la(indpos) < C - 10^(-3));
28 i = indpos(ind(1));
29 b = 1/y(i) - w'*T(i,:)' ;
30
31 la
32 w
33 b
34
35 %opzionale
36 %calcolo errori xi
37 for i=1:l
38     if (la(i) > C-0.001 & la(i) < C+0.001)
39         xi(i) = 1 - y(i)*(T(i,:)*w+b);
40     else
41         xi(i)=0;
42     end
43 end
44 xi'
```

## Non-linear $\epsilon$ -SV regression

```
35 x = data(:,1);
36 y = data(:,2);
37 l = length(x);
38 epsilon = 3;
39 C = 5;
40
41 X = zeros(l,l);
42 for i = 1 : l
43     for j = 1 : l
44         X(i,j) = kernel(x(i),x(j));
45     end
46 end
47
48 Q = [ X -X ; -X X ];
49 c = epsilon*ones(2*l,1) + [-y;y];
50
51 sol = quadprog(Q, c, [], [], ...
52     [ones(1,l) -ones(1,l)], 0, ...
53     zeros(2*l,1), C*ones(2*l,1));
54
55 la_p = sol(1:l);
56 la_n = sol(l+1:2*l);
57
58 ind = find(la_p > 1e-3 & la_p < C-1e-3);
59 if isempty(ind)==0
60     i = ind(1);
61     b = y(i) - epsilon;
62 else
63     ind = find(la_n > 1e-3 & la_n < C-1e-3);
64     i = ind(1);
65     b = y(i) + epsilon ;
66 end
67
68 for j = 1 : l
69     b = b - (la_p(j)-la_n(j))*kernel(x(i),x(j));
70 end
71
72 sv = [find(la_p > 1e-3); find(la_n > 1e-3)];
73 sv = sort(sv);
74
75 disp('Support vectors')
76 disp([sv,x(sv),y(sv),la_n(sv),la_p(sv)])
77 b
78
79 function v = kernel(x,y)
80     p = 4 ;
81     v = (x'*y + 1)^p;
82 end
```

## k-means with 2-norm

```
22 k=3;
23 InitialCentroids=[1,1;2,2;3,3];
24
25 [x,cluster,v] = kmeans1(data,k,InitialCentroids)
26
27 function [x,cluster,v] = kmeans1(data,k,InitialCentroids)
28     l = size(data,1);
29     x = InitialCentroids;
30     cluster = zeros(l,1);
31
32     for i = 1 : l
33         d = inf;
34
35         for j = 1 : k
36             if norm(data(i,:)-x(j,:)) < d
37                 d = norm(data(i,:)-x(j,:));
38                 cluster(i) = j;
39             end
40         end
41     end
42
43     v_old = 0;
44     for i = 1 : l
45         v_old = v_old + norm(data(i,:)-x(cluster(i),:))^2 ;
46     end
47
48     while true
49         for j = 1 : k
50             ind = find(cluster == j);
51             if isempty(ind)==0
52                 x(j,:) = mean(data(ind,:),1);
53             end
54         end
55
56         for i = 1 : l
57             d = inf;
58             for j = 1 : k
59                 if norm(data(i,:)-x(j,:)) < d
60                     d = norm(data(i,:)-x(j,:));
61                     cluster(i) = j;
62                 end
63             end
64         end
65
66         v = 0;
67         for i = 1 : l
68             v = v + norm(data(i,:)-x(cluster(i),:))^2 ;
69         end
70
71         if v_old - v < 1e-5
72             break
73         else
74             v_old = v;
75         end
76     end
77 end
```

## Multistart approach

```
21 k = 3;
22 N = 50;
23
24 best_v = Inf;
25 best_start = [];
26 best_centroids = [];
27 best_clusters = [];
28
29
30 for i = 1:N
31     x = data(randperm(size(data, 1), k), :);
32
33     [centroids, clusters, v] = kmeans1(data, k, x);
34
35     if v < best_v
36         best_v = v;
37         best_start = x;
38         best_centroids = centroids;
39         best_clusters = clusters;
40     end
41 end
42
43 disp(best_start);
44 disp(best_centroids);
45 disp(best_clusters);
46 disp(num2str(best_v));
47
48 function [x, cluster, v] = kmeans1(data, k, InitialCentroids)...
```

## Esercizio 3

### Linear case

```
1 C = [1 1 -1 ; 1 1 0 ] ;
2 A = [ 1 1 1 ; -1 -1 0 ; 0 -1 0 ];
3 b = [4 0 2]';
4
5 MINIMA=[];
6 LAMBDA=[];
7 DEG = [];
8
9 for alfa = 0 : 0.01 : 1
10     [x,fval,exitflag,output,lambda] = linprog(alfa*C(1,:)+(1-alfa)*C(2,:),A,b);
11     MINIMA=[MINIMA; alfa x'];
12     LAMBDA=[LAMBDA;lambda.ineqlin'];
13
14     S=find(lambda.ineqlin < 0.01);
15     if size(S, 1) > 0.1
16         DEG = [DEG; alfa, x', lambda.ineqlin'];
17     end
18 end
19
20 fprintf('\t alpha \t x(1) \t x(2) \t x(3) \t LAMBDA \n\n');
21 [MINIMA , LAMBDA]
22
23 fprintf('soluzioni degeneri\n')
24 fprintf('\t alpha \t x(1) \t x(2) \t x(3) \t LAMBDA \n\n');
25 DEG
```

## Esercizio 4

### Matrix Game

```
1 C=[1,4,-1,5,2; 2 1 3 3 5; 2 3 -2 3 1;1 1 5 2 3];
2 m = size(C,1);
3 n = size(C,2);
4 c=[zeros(m,1);1];
5 A=[C', -ones(n,1)]; b=zeros(n,1);
6 Aeq=[ones(1,m),0]; beq=1;
7 lb= [zeros(m,1);-inf];
8 ub=[ ];
9
10 [sol,Val,exitflag,output,lambda] = linprog(c, A,b, Aeq, beq, lb, ub);
11
12 x = sol(1:m)
13 y = lambda.ineqlin
```