Esercizio 1

Gradient method (exact line search)

```
Q=[6 0 -4 0;0 6 0 -4;-4 0 6 0;0 -4 0 6];
c = [ 1 -1 2 -3]';

disp('eigenvalues of Q')
eig(Q)

x0 = [0 0 0 0]';
tolerance = 10^(-6);
x = x0;
X=[];

for ITER=1:1000
    v = 0.5*x'*Q*x + c'*x;
    g = Q*x + c ;
    X=[X;ITER,x',v,norm(g)];

    if norm(g) < tolerance
        break
    end

    d = -g;
    t = norm(g)^2/(d'*Q*d) ;
    x = x + t*d ;
end

disp(X)
```

Gradient method (inexact line search)

```
alpha = 0.1; gamma = 0.9; tbar = 1;
x0 = [ 10 -10]';
tolerance = 10^(-3);
x = x0;
X=[];

for ITER=1:1000
    [v, g] = f(x);

    X=[X;ITER,x',v,norm(g)];

    if norm(g) < tolerance
        break
    end

    d = -g;
    t = tbar ;

    while (f(x+t*d) > v + alpha*g'*d*t)
        t = gamma*t ;
    end

    x = x + t*d;
end

disp(X)

function [v, g] = f(x)
v = x(1)^4 + x(2)^4 - 2*x(1)^2 + 4*x(1)*x(2)-2*x(2)^2 ;
g = [4*x(1)^3-4*x(1)+4*x(2);
     4*x(2)^3+4*x(1)-4*x(2)];
end
```

Conjugate Gradient method

```
Q = [6 1 0 2; 1 2 0 0; 0 0 4 2; 2 0 2 8];
c = [-1; 8; 6; 9];

disp('Eigenvalues of Q:')
eig(Q)

x0 = [0 0 0 0]';
tolerance = 10^(-6);
x = x0;
X = [];

for ITER=1:10
    v = 0.5*x'*Q*x + c'*x;
    g = Q*x + c;

    X = [X; ITER, x', v, norm(g)];

    if norm(g) < tolerance
        break
    end

    if ITER == 1
        d = -g;
    else
        beta = (g'*Q*d_prev)/(d_prev'*Q*d_prev);
        d = -g + beta*d_prev;
    end

    t = -(g'*d)/(d'*Q*d);

    x = x + t*d;
    d_prev = d;
end

disp(X)
```

Penalty method (exact)

```
global A b eps;
A = [2 1 ; -1 -1 ; -1 0 ];
b = [ 4 ; -1 ; 0 ];
tau = 0.1;
eps0 = 5;
tolerance = 1e-6;

eps = eps0;
x = [4 0]';
X=[];

for ITER=1:1000
    [x,pval] = fminunc(@p_eps,x);
    infeas = max(A*x-b);

    X=[X;ITER,eps,x',infeas,pval];

    if infeas < tolerance
        break
    else
        eps = tau*eps;
    end
end

disp(X)

function v= p_eps(x)
    global A b eps;
    v = x(1)^2 -log(x(1)+x(2)) ;

    for i = 1 : size(A,1)
        v = v + (1/eps)*(max(0,A(i,:)*x-b(i)))^2;
    end
end
```

Logarithmic Barrier method

```
global Q c A b eps;
Q = [2 -1 0 ;-1 2 1; 0 1 2 ] ;
c = [-3 -4 -5]';
A = [2 1 1;-1 0 0; 0 1 0 ;0 0 -1 ];
b = [20 -2 3 -4]';
eig(Q)
delta = 1e-3 ;
tau = 0.5 ;
eps1 = 1 ;
x0 = [3;2;5];

x = x0;
eps = eps1;
m = size(A,1);
X=[];

for ITER=1:1000
    [x,pval] = fminunc(@logbar,x);
    gap = m*eps;

    X=[X;ITER,eps,x',gap,pval];

    if gap < delta
        break
    else
        eps = eps*tau;
    end
end

disp(X)

function v = logbar(x)
    global Q c A b eps
    v = 0.5*x'*Q*x + c'*x ;

    for i = 1 : length(b)
        v = v - eps*log(b(i)-A(i,:)*x) ;
    end
end
```

Newton method

```
alpha=0.1; gamma=0.9; tbar =1;
x0 = [0 0]';
tolerance = 10^(-3) ;
x = x0;
X = [];

for ITER=1:100
    [v, g, H] = f(x);
    X=[X;ITER,x',v,norm(g)];

    if norm(g) < tolerance
        break
    end

    d = -inv(H)*g;
    t=tbar;
    while (f(x+t*d) > f(x) + alpha*t*d'*g)
        t=gamma*t;
    end

    x = x + t*d;
end

disp(X)

function [v, g, H] = f(x)
    v = 2*x(1)^4 + 3*x(2)^4 + 2*x(1)^2 + 4*x(2)^2 + x(1)*x(2)

    g = [ 8*x(1)^3 + 4*x(1) + x(2) - 3;
         12*x(2)^3 + 8*x(2) + x(1) - 2];

    H = [ 24*x(1)^2+4 1;
          1 36*x(2)^2+8];
end
```

Esercizio 2

Linear SVM with soft margin

```
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
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

nA = size(A,1);
nB = size(B,1);

T = [A; B];
C = 10;
y = [ones(nA,1); -ones(nB, 1)];
l = length(y);
Q = zeros(l,l);

for i = 1:l
    for j = 1 : l
        Q(i,j) = y(i)*y(j)*T(i,:)*T(j,:)';
    end
end

la = quadprog(Q, -ones(l,1), [], [], y', 0, zeros(l,1), C*ones(l,1));

w = zeros(2,1);
for i = 1:l
    w = w + la(i)*y(i)*T(i,:)';
end

indpos = find(la > 10^(-3));
ind = find(la(indpos) < C - 10^(-3));
i = indpos(ind(1));
b = 1/y(i) - w'*T(i,:)';

la
w
b

%opzionale
%calcolo errori xi
for i=1:l
    if (la(i) > C-0.001 & la(i) < C+0.001)
        xi(i) = 1 - y(i)*(T(i,:)*w+b);
    else
        xi(i)=0;
    end
end
xi'
```

Non-linear ε-SV regression

```
x = data(:,1);
y = data(:,2);
l = length(x);
epsilon = 3;
C = 5;

X = zeros(l,l);
for i = 1 : l
    for j = 1 : l
        X(i,j) = kernel(x(i),x(j));
    end
end

Q = [ X -X ; -X X ];
c = epsilon*ones(2*l,1) + [-y;y];

sol = quadprog(Q, c, [], [], ...
    [ones(1,l) -ones(1,l)], 0, ...
    zeros(2*l,1), C*ones(2*l,1));

la_p = sol(1:l);
la_n = sol(l+1:2*l);

ind = find(la_p > 1e-3 & la_p < C-1e-3);
if isempty(ind)==0
    i = ind(1);
    b = y(i) - epsilon;
else
    ind = find(la_n > 1e-3 & la_n < C-1e-3);
    i = ind(1);
    b = y(i) + epsilon ;
end

for j = 1 : l
    b = b - (la_p(j)-la_n(j))*kernel(x(i),x(j));
end

sv = [find(la_p > 1e-3); find(la_n > 1e-3)];
sv = sort(sv);

disp('Support vectors')
disp([sv,x(sv),y(sv),la_n(sv),la_p(sv)])
b

function v = kernel(x,y)
    p = 4 ;
    v = (x'*y + 1)^p;
end
```

k-means with 2-norm

```
k=3;
InitialCentroids=[1,1;2,2;3,3];

[x,cluster,v] = kmeans1(data,k,InitialCentroids)

function [x,cluster,v] = kmeans1(data,k,InitialCentroids)
    l = size(data,1);
    x = InitialCentroids;
    cluster = zeros(l,1);

    for i = 1 : l
        d = inf;

        for j = 1 : k
            if norm(data(i,:)-x(j,:)) < d
                d = norm(data(i,:)-x(j,:));
                cluster(i) = j;
            end
        end
    end

    v_old = 0;
    for i = 1 : l
        v_old = v_old + norm(data(i,:)-x(cluster(i),:))^2 ;
    end

    while true
        for j = 1 : k
            ind = find(cluster == j);
            if isempty(ind)==0
                x(j,:) = mean(data(ind,:),1);
            end
        end

        for i = 1 : l
            d = inf;
            for j = 1 : k
                if norm(data(i,:)-x(j,:)) < d
                    d = norm(data(i,:)-x(j,:));
                    cluster(i) = j;
                end
            end
        end

        v = 0;
        for i = 1 : l
            v = v + norm(data(i,:)-x(cluster(i),:))^2 ;
        end

        if v_old - v < 1e-5
            break
        else
            v_old = v;
        end
    end
end
```

Multistart approach

```
k = 3;
N = 50;

best_v = Inf;
best_start = [];
best_centroids = [];
best_clusters = [];


for i = 1:N
    x = data(randperm(size(data, 1), k), :);

    [centroids, clusters, v] = kmeans1(data, k, x);

    if v < best_v
        best_v = v;
        best_start = x;
        best_centroids = centroids;
        best_clusters = clusters;
    end
end

disp(best_start);
disp(best_centroids);
disp(best_clusters);
disp(num2str(best_v));

function [x, cluster, v] = kmeans1(data, k, InitialCentroids)...
```

Esercizio 3

Linear case

```
C = [1 1 -1 ; 1 1 0 ] ;
A =[ 1 1 1 ; -1 -1 0 ; 0 -1 0 ];
b = [4 0 2]';

MINIMA=[];
LAMBDA=[];
DEG = [];

for alfa = 0 : 0.01 : 1
    [x,fval,exitflag,output,lambda] = linprog(alfa*C(1,:)+(1-alfa)*C(2,:),A,b);
    MINIMA=[MINIMA; alfa x'];
    LAMBDA=[LAMBDA;lambda.ineqlin'];

    S=find(lambda.ineqlin < 0.01);
    if size(S, 1) > 0.1
        DEG = [DEG; alfa, x', lambda.ineqlin'];
    end
end

fprintf('\t alpha \t x(1) \t x(2) \t x(3) \t LAMBDA \n\n');
[MINIMA , LAMBDA]

fprintf('soluzioni degeneri\n')
fprintf('\t alpha \t x(1) \t x(2) \t x(3) \t LAMBDA \n\n');
DEG
```

Esercizio 4

Matrix Game

```
C=[1,4,-1,5,2; 2 1 3 3 5; 2 3 -2 3 1;1 1 5 2 3];
m = size(C,1);
n = size(C,2);
c=[zeros(m,1);1];
A=[C', -ones(n,1)]; b=zeros(n,1);
Aeq=[ones(1,m),0]; beq=1;
lb= [zeros(m,1);-inf];
ub=[ ];

[sol,Val,exitflag,output,lambda] = linprog(c, A,b, Aeq, beq, lb, ub);

x = sol(1:m)
y = lambda.ineqlin
```