2 Existence of optimal solutions and optimality conditions

Existence of global optima
Theorem (Weierstrass)
If the objective function f is continuous and the feasible region Ω is closed and bounded, then there exists a global optimum.
Proof. Let $v^* = \inf_{x \in \Omega} f(x)$. Define a minimizing sequence $\{x^k\} \subseteq \Omega$ s.t. $f(x^k) \to v^*$.
Since $\{x^k\}$ is bounded, the Bolzano-Weierstrass theorem guarantees that there exists a subsequence $\{x^{k_p}\}$ converging to some point x^*. Since Ω is closed, we get $x^* \in \Omega$.
Finally, $f(x^{k_p}) \to f(x^*)$ since f is continuous. Therefore, $f(x^*) = v^*$, i.e., x^* is a global optimum.

Corollary 1
If all the functions f, g_i, h_j are continuous, the domain D is closed and the feasible region Ω is bounded, then there exists a global optimum.

Corollary 2
If the objective function f is continuous, the feasible region Ω is closed and there exists $\alpha \in \mathbb{R}$ such that the α-sublevel set
$S_\alpha(f) = \{x \in \Omega : f(x) \leq \alpha\}$
is nonempty and bounded, then there exists a global optimum.

Corollary 3
If the objective function f is continuous and coercive, i.e.,
$\lim_{\|x\| \to \infty} f(x) = +\infty$,
and the feasible region Ω is closed, then there exists a global optimum.
Proof. Any sublevel set of f is bounded, then use Corollary 2.

Corollary 4
IIf f is strongly convex and Ω is closed, then there exists a global optimum.
IIf f is strongly convex and Ω is closed and convex, then there exists a unique global optimum.

3 Lagrangian Dual

```
clear; close all; clc;


%% Primal problem

x = -1 : 0.01 : 3;
y = 2*x.^4+x.^3-20*x.^2+x;
[xopt,vp]=fminbnd(@(x) 2*x^4+x^3-20*x^2+x,-1,3);
```

```
plot(x,y,'b-',x,vp*ones(length(x),1),'b-','LineWidth',1.5);
title('Primal problem: min 2x^4+x^3-20x^2+x s.t. x^2-2x-3 \leq 0');
axis([-1 3 -45 15]);

pause

%% Dual problem

x = -4 : 0.01 : 4;
phi = [];
figure;
for lam = 0 : 0.01 : 5
y = 2*x.^4+x.^3-20*x.^2+x + lam*(x.^2-2.*x-3);
[~,vl1]=fminbnd(@(x) 2*x^4+x^3-20*x^2+x+lam*(x^2-2*x-3),-3,0);
[~,vl2]=fminbnd(@(x) 2*x^4+x^3-20*x^2+x+lam*(x^2-2*x-3),0,3);
vl = min(vl1,vl2);
phi = [phi ; vl];
plot(x,y,'r-',x,vl*ones(length(x),1),'r-',x,vp*ones(length(x),1),'b-','LineWidth',1.5);
title(['Lagrangian function with \lambda = ',num2str(lam)]);
axis([-4 4 -70 50]);
if lam == 0
pause
else
pause(0.03)
end
end

vd = max(phi);
figure;
plot(0:0.01:5,phi,'r-',0:0.01:5,vd*ones(length(0:0.01:5),1),'r-',...
0:0.01:5,vp*ones(length(0:0.01:5),1),'b-','LineWidth',1.5);
title('Dual problem');
```

4 SVM for classification problems

%% Exercise 4.1

```
close all; clear; clc;

%% data

A=[ 0.4952 6.8088
...
2.2699 7.1371];

B=[7.2450 3.4422
...
9.8621 4.3674 ];


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

% training points
T = [A ; B];

%% Linear SVM - primal model

% define the optimization problem
Q = [ 1 0 0 ;
0 1 0 ;
0 0 0 ];
D = [-A -ones(nA,1);
B ones(nB,1) ] ;
d = -ones(nA+nB,1) ;

% solve the problem
options = optimset('Largescale','off','display','off');
sol = quadprog(Q,zeros(3,1),D,d,[],[],[],[],[],options);

w = sol(1:2)
b = sol(3)

% plot the solution
xx = 0:0.1:10 ;
uu = (-w(1)/w(2)).*xx - b/w(2);
vv = (-w(1)/w(2)).*xx + (1-b)/w(2);
vvv = (-w(1)/w(2)).*xx + (-1-b)/w(2);
plot(A(:,1),A(:,2),'bo',B(:,1),B(:,2),'ro',xx,uu,'k-',xx,vv,'b-',xx,vvv,'r-','Linewidth',1.5)
axis([0 10 0 10])
title('Optimal separating hyperplane (primal model)')
```

%% Classification problems - Exercise 4.2

```
close all; clear; clc;

%% data

A=[ 0.4952 6.8088
...
2.2699 7.1371];

B=[7.2450 3.4422
...
9.8621 4.3674 ];

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

% training points
T = [A ; B];

%% Linear SVM - dual model

% define the problem
y = [ones(nA,1) ; -ones(nB,1)]; % labels
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

% solve the problem
options = optimset('Largescale','off','display','off');
la = quadprog(Q,-ones(l,1),[],[],y',0,zeros(l,1),[],[],options);

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

% compute scalar b
ind = find(la > 1e-3) ;
i = ind(1) ;
bD = 1/y(i) - wD'*T(i,:)'

% plot the solution
```

```
xx = 0:0.1:10 ;
uuD = (-wD(1)/wD(2)).*xx - bD/wD(2);
vvD = (-wD(1)/wD(2)).*xx + (1-bD)/wD(2);
vvvD = (-wD(1)/wD(2)).*xx + (-1-bD)/wD(2);
figure
plot(A(:,1),A(:,2),'bo',B(:,1),B(:,2),'ro',...
xx,uuD,'k-',xx,vvD,'b-',xx,vvvD,'r-','Linewidth',1.5)
axis([0 10 0 10])
title('Optimal separating hyperplane (dual model)')

%% support vectors

supp = find(la > 1e-3);
suppA = supp(supp <= nA);
suppB = supp(supp > nA)-nA;

hold on
plot(A(suppA,1),A(suppA,2),'bo',...
B(suppB,1),B(suppB,2),'ro','Linewidth',5)
legend('Support vectors of A','Support vectors of B','Location','NorthEastOutside')
```

%% Classification problems - Exercise 4.4

```
close all; clear; clc;

%% data

A=[ 0.4952 6.8088
...
4.5000 2.0000];

B=[7.2450 3.4422
...
2.0000 3.0000 ];
nA = size(A,1);
nB = size(B,1);

% training points
T = [A ; B];

%% Linear SVM with soft margin - dual model

% define the problem
C = 10 ;
y = [ones(nA,1) ; -ones(nB,1)]; % labels
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

% solve the problem
options = optimset('Largescale','off','display','off');
la = quadprog(Q,-ones(l,1),[],[],y',0,zeros(l,1),C*ones(l,1),[],options);

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

% compute scalar b
ind = find((la > 1e-3) & (la < C-1e-3));
i = ind(1);
bD = 1/y(i) - wD'*T(i,:)'

%% plot the solution

xx = 0:0.1:10;
uuD = (-wD(1)/wD(2)).*xx - bD/wD(2);
vvD = (-wD(1)/wD(2)).*xx + (1-bD)/wD(2);
vvvD = (-wD(1)/wD(2)).*xx + (-1-bD)/wD(2);
plot(A(:,1),A(:,2),'b.',B(:,1),B(:,2),'r.',...
xx,uuD,'k-',xx,vvD,'b-',xx,vvvD,'r-','Linewidth',1.5)
axis([0 10 0 10])
title('Optimal separating hyperplane (soft margin)')

%% support vectors

supp = find(la > 1e-3);
suppA = supp(supp <= nA);
suppB = supp(supp > nA)-nA;

hold on
plot(A(suppA,1),A(suppA,2),'bo',...
B(suppB,1),B(suppB,2),'ro','Linewidth',5)
legend('Support vectors of A','Support vectors of B','Location','NorthEastOutside')
```

%% Classification problems - Exercise 4.5

```
%% Nonlinear SVM

% parameter
C = 1 ;
```

```
% Gaussian kernel
gamma = 1 ;
K = zeros(l,l);
for i = 1 : l
for j = 1 : l
K(i,j) = exp(-gamma*norm(T(i,:)-T(j,:))^2);
end
end

% define the problem
Q = zeros(l,l);
for i = 1 : l
for j = 1 : l
Q(i,j) = y(i)*y(j)*K(i,j) ;
end
end

% solve the problem
options = optimset('Largescale','off','display','off');
[la,ov] = quadprog(Q,-ones(l,1),[],[],y',0,zeros(l,1),C*ones(l,1),[],options);

% compute b
ind = find((la > 1e-3) & (la < C-1e-3));
i = ind(1);
b = 1/y(i) ;
for j = 1 : l
b = b - la(j)*y(j)*K(i,j);
end

%% plot the surface

AA=[];
BB=[];
for xx = -2 : 0.05 : 2
for yy = -2 : 0.05 : 2
s = 0;
for i = 1 : l
s = s + la(i)*y(i)*exp(-gamma*norm(T(i,:)-[xx yy])^2);
end
s = s + b;
if s > 0
AA = [AA ; xx yy];
else
BB = [BB ; xx yy];
end
end
end
plot(A(:,1),A(:,2),'bo',B(:,1),B(:,2),'ro','Linewidth',5)
hold on
plot(AA(:,1),AA(:,2),'b.',BB(:,1),BB(:,2),'r.','Linewidth',0.01);
```

```
title(['Separating surface with Gaussian kernel (C = ',num2str(C),' and \gamma =
',num2str(gamma),')'])
```

5 Regression problems

%% Regression problems - Exercise 5.1

```
data = [-5.0000 -96.2607
...
5.0000 89.1652];

x = data(:,1) ;
y = data(:,2) ;
l = length(x) ;
n = 4 ;

% Vandermonde matrix
A = [ ones(l,1) x x.^2 x.^3 ];

%% 2-norm problem

z2 = (A'*A)\(A'*y)
p2 = A*z2;

%% 1-norm problem

% define the problem
c = [ zeros(n,1); ones(l,1) ];
D = [ A -eye(l); -A -eye(l) ];
d = [ y; -y ];

% solve the problem
sol1 = linprog(c,D,d) ;
z1 = sol1(1:n)
p1 = A*z1;

%% inf-norm problem

% define the problem
c = [ zeros(n,1); 1 ];
D = [ A -ones(l,1); -A -ones(l,1) ];

% solve the problem
solinf = linprog(c,D,d) ;
zinf = solinf(1:n)
pinf = A*zinf;
```

```
%% plot the solutions

plot(x,y,'b.',x,p2,'r-',x,p1,'k-',x,pinf,'g-')
legend('Data','2-norm','1-norm','inf-norm',...
'Location','NorthWest');
```

%% Regression problems - Exercise 5.2

```
close all; clear; clc;

%% data

data = [ 0    2.5584
...
   10.0000    7.2537];

x = data(:,1) ;
y = data(:,2) ;
l = length(x) ; % number of points

%% linear regression - primal problem

% parameter
epsilon = 0.5 ;

% define the problem
Q = [ 1 0
      0 0 ];

c = [0;0];

D = [-x -ones(l,1)
      x  ones(l,1)];

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

% solve the problem
sol = quadprog(Q,c,D,d);

% compute w
w = sol(1);

% compute b
b = sol(2);

% find regression and epsilon-tube
z = w.*x + b ;
zp = w.*x + b + epsilon ;
zm = w.*x + b - epsilon ;

%% plot the solution

plot(x,y,'b.',x,z,'k-',x,zp,'r-',x,zm,'r-');
legend('Data','regression','\epsilon-tube',...
    'Location','NorthWest')
```

%% Regression problems - Exercise 5.3

```
data = [ 0    2.5584
...
   10.0000    7.2537];

x = data(:,1) ;
y = data(:,2) ;
l = length(x) ; % number of points

%% linear regression - primal problem with slack variables

% parameters
epsilon = 0.2 ;
C = 10 ;

% define the problem
Q = [ 1              zeros(1,2*l+1)
      zeros(2*l+1,1) zeros(2*l+1) ];

c = [ 0 ; 0 ; C*ones(2*l,1)];

D = [-x -ones(l,1) -eye(l)   zeros(l)
      x  ones(l,1)  zeros(l) -eye(l)];

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

% solve the problem
sol = quadprog(Q,c,D,d,[],[],[-inf;-inf;zeros(2*l,1)],[]);

% compute w
w = sol(1);

% compute b
b = sol(2);

% compute slack variables xi+ and xi-
xip = sol(3:2+l);
xim = sol(3+l:2+2*l);

% find regression and epsilon-tube
z = w.*x + b ;
zp = w.*x + b + epsilon ;
zm = w.*x + b - epsilon ;

%% plot the solution

plot(x,y,'b.',x,z,'k-',x,zp,'r-',x,zm,'r-');
legend('Data','regression',...
    '\epsilon-tube','Location','NorthWest')
```

%% Regression problems - Exercise 5.4

```
close all; clear; clc;

%% data
```

```
data = [ 0    2.5584
...
   10.0000    7.2537];

x = data(:,1) ;
y = data(:,2) ;
l = length(x) ; % number of points

%% linear regression - dual problem

% parameters
epsilon = 0.2 ;
C = 10;

% define the problem
X = zeros(l,l);
for i = 1 : l
    for j = 1 : l
        X(i,j) = x(i)*x(j);
    end
end
Q = [ X -X ; -X X ];
c = epsilon*ones(2*l,1) + [-y;y];

% solve the problem
sol = quadprog(Q,c,[],[],[ones(1,l) -ones(1,l)],0,zeros(2*l,1),C*ones(2*l,1));
lap = sol(1:l);
lam = sol(l+1:2*l);

% compute w
w = (lap-lam)'*x ;

% compute b
ind = find(lap > 1e-3 & lap < C-1e-3);
if ~isempty(ind)
    i = ind(1);
    b = y(i) - w*x(i) - epsilon ;
else
    ind = find(lam > 1e-3 & lam < C-1e-3);
    i = ind(1);
    b = y(i) - w*x(i) + epsilon ;
end

% find regression and epsilon-tube
z = w.*x + b ;
zp = w.*x + b + epsilon ;
zm = w.*x + b - epsilon ;

%% plot the solution

% find support vectors
sv = [find(lap > 1e-3);find(lam > 1e-3)];
sv = sort(sv);

plot(x,y,'b.',x(sv),y(sv),...
    'ro',x,z,'k-',x,zp,'r-',x,zm,'r-');

legend('Data','Support vectors',...
    'regression','\epsilon-tube',...
    'Location','NorthWest')
```

%% Regression problems - Exercise 5.5

```
close all; clear; clc;

%% data

data = [-5.0000  -96.2607
...
    5.0000   89.1652];

x = data(:,1) ;
y = data(:,2) ;
l = length(x) ; % number of points

%% nonlinear regression - dual problem

epsilon = 10 ;
C = 10;

% define the problem
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];

% solve the problem
sol = quadprog(Q,c,[],[],...
    [ones(1,l) -ones(1,l)],0,...
    zeros(2*l,1),C*ones(2*l,1));
lap = sol(1:l);
lam = sol(l+1:2*l);

% compute b
ind = find(lap > 1e-3 & lap < C-1e-3);
if ~isempty(ind)
    i = ind(1);
    b = y(i) - epsilon;
    for j = 1 : l
        b = b - (lap(j)-lam(j))*kernel(x(i),x(j));
    end
else
    ind = find(lam > 1e-3 & lam < C-1e-3);
    i = ind(1);
    b = y(i) + epsilon ;
    for j = 1 : l
        b = b - (lap(j)-lam(j))*kernel(x(i),x(j));
    end
end

% find regression and epsilon-tube
z = zeros(l,1);
for i = 1 : l
    z(i) = b ;
    for j = 1 : l
        z(i) = z(i) + (lap(j)-lam(j))*kernel(x(i),x(j));
    end
end
```

```
zp = z + epsilon ;
zm = z - epsilon ;

%% plot the solution

% find support vectors
sv = [find(lap > 1e-3);find(lam > 1e-3)];
sv = sort(sv);

plot(x,y,'b.',x(sv),y(sv),...
    'ro',x,z,'k-',x,zp,'r-',x,zm,'r-');

legend('Data','Support vectors',...
    'regression','\epsilon-tube',...
    'Location','NorthWest')

%% kernel function

function v = kernel(x,y)

p = 4 ;
v = (x'*y + 1)^p;

end
```

%% Regression problems - Exercise 5.6

```
close all; clear; clc;

%% data

data = [ 0    0.0620
...
   10.0000    0.9881];

x = data(:,1) ;
y = data(:,2) ;
l = length(x) ; % number of points

%% nonlinear regression - dual problem

epsilon = 0.2 ;
C = 10;

% define the problem
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];

% solve the problem
sol = quadprog(Q,c,[],[],[ones(1,l) -ones(1,l)],0,zeros(2*l,1),C*ones(2*l,1));
lap = sol(1:l);
lam = sol(l+1:2*l);

% compute b
```

```
ind = find(lap > 1e-3 & lap < C-1e-3);
if ~isempty(ind)
    i = ind(1);
    b = y(i) - epsilon;
    for j = 1 : l
        b = b - (lap(j)-lam(j))*kernel(x(i),x(j));
    end
else
    ind = find(lam > 1e-3 & lam < C-1e-3);
    i = ind(1);
    b = y(i) + epsilon ;
    for j = 1 : l
        b = b - (lap(j)-lam(j))*kernel(x(i),x(j));
    end
end

% find regression and epsilon-tube
z = zeros(l,1);
for i = 1 : l
    z(i) = b ;
    for j = 1 : l
        z(i) = z(i) + (lap(j)-lam(j))*kernel(x(i),x(j));
    end
end
zp = z + epsilon ;
zm = z - epsilon ;

%% plot the solution

% find support vectors
sv = [find(lap > 1e-3);find(lam > 1e-3)];
sv = sort(sv);

plot(x,y,'b.',x(sv),y(sv),...
    'ro',x,z,'k-',x,zp,'r-',x,zm,'r-');

legend('Data','Support vectors',...
    'regression','\epsilon-tube',...
    'Location','NorthEast')

%% kernel function

function v = kernel(x,y)

gamma = 1 ;
v = exp(-gamma*norm(x-y)^2);

end
```

6 Clustering problems

%% Clustering problem - Exercise 6.1

```
data = [ 1.2734    6.2721
 ...
    3.7431    2.9852];

l = size(data,1); % number of patterns

% plot patterns
plot(data(:,1),data(:,2),'ko');
axis([0 10 0 10])
title('k-means algorithm');
hold on

k = 3; % number of clusters

% initialize centroids
x = [5 7; 6 3 ; 4 3];  % part a)
% x = [5 7; 6 3 ; 4 4];  % part b)
% x = 10*rand(3,2);      % part c)

% plot centroids
plot(x(1,1),x(1,2),'b^',...
    x(2,1),x(2,2),'r^',...
    x(3,1),x(3,2),'g^');

pause

% initialize clusters
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

% plot cluster
c1 = data(cluster==1,:);
c2 = data(cluster==2,:);
c3 = data(cluster==3,:);
plot(c1(:,1),c1(:,2),'bo',c2(:,1),c2(:,2),'ro',...
    c3(:,1),c3(:,2),'go');

% compute the objective function value
v = 0;
for i = 1 : l
    v = v + norm(data(i,:)-x(cluster(i),:))^2 ;
end
title(['k-means algoritm: objective function = ',num2str(v)]);

pause
```

```matlab
while true

    % delete old centroids
    plot(x(1,1),x(1,2),'w^',...
        x(2,1),x(2,2),'w^',...
        x(3,1),x(3,2),'w^');

    % update centroids
    for j = 1 : k
        ind = find(cluster == j);
        if ~isempty(ind)
            x(j,:) = mean(data(ind,:));
        end
    end

    % plot new centroids
    plot(x(1,1),x(1,2),'b^',...
        x(2,1),x(2,2),'r^',...
        x(3,1),x(3,2),'g^');

    pause

    % update clusters
    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

    % plot cluster
    c1 = data(cluster==1,:);
    c2 = data(cluster==2,:);
    c3 = data(cluster==3,:);
    plot(c1(:,1),c1(:,2),'bo',c2(:,1),c2(:,2),...
        'ro',c3(:,1),c3(:,2),'go');


    % update objective function
    vnew = 0;
    for i = 1 : l
        vnew = vnew + norm(data(i,:)-x(cluster(i),:))^2 ;
    end
    title(['k-means algoritm: objective function = ',num2str(vnew)]);

    pause

    % stopping criterion
    if v - vnew < 1e-5
        break
    else
        v = vnew;
    end

end

v
```

%% Clustering problem - Exercise 6.2

```
data = [ 1.2734    6.2721
   ...
    3.7431    2.9852];

l = size(data,1); % number of patterns

% plot patterns
plot(data(:,1),data(:,2),'ko');
axis([0 10 0 10])
title('k-median algoritm');
hold on

pause

k = 3; % number of clusters

% initialize centroids
x = [5 7; 6 3 ; 4 3];  % part a)
% x = [5 7; 6 3 ; 4 4];  % part b)
% x = 10*rand(3,2);      % part c)

% plot centroids
plot(x(1,1),x(1,2),'b^',...
    x(2,1),x(2,2),'r^',...
    x(3,1),x(3,2),'g^');

pause

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

% plot cluster
c1 = data(cluster==1,:);
c2 = data(cluster==2,:);
c3 = data(cluster==3,:);
plot(c1(:,1),c1(:,2),'bo',c2(:,1),c2(:,2),'ro',c3(:,1),c3(:,2),'go');

% compute the objective function value
v = 0;
for i = 1 : l
    v = v + norm(data(i,:)-x(cluster(i),:),1);
end
title(['k-median algoritm: objective function = ',num2str(v)]);

pause

while true
```

```
    % delete old centroids
    plot(x(1,1),x(1,2),'w^',...
        x(2,1),x(2,2),'w^',...
        x(3,1),x(3,2),'w^');

    % update centroids
    for j = 1 : k
        ind = find(cluster == j);
        if ~isempty(ind)
            x(j,:) = median(data(ind,:));
        end
    end

    % plot new centroids
    plot(x(1,1),x(1,2),'b^',...
        x(2,1),x(2,2),'r^',...
        x(3,1),x(3,2),'g^');

    pause

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

    % plot cluster
    c1 = data(cluster==1,:);
    c2 = data(cluster==2,:);
    c3 = data(cluster==3,:);
    plot(c1(:,1),c1(:,2),'bo',c2(:,1),c2(:,2),'ro',c3(:,1),c3(:,2),'go');

    % update objective function
    vnew = 0;
    for i = 1 : l
        vnew = vnew + norm(data(i,:)-x(cluster(i),:),1);
    end
    title(['k-median algoritm: objective function = ',num2str(vnew)]);

    pause

    % stopping criterion
    if v - vnew < 1e-5
        break
    else
        v = vnew;
    end

end

v
```

7 Solution methods for unconstrained optimization

%% Unconstrained optimization -- Exercise 7.1

```
clear; close all; clc;

%% data

Q = [6     0    -4     0
     0     6     0    -4
    -4     0     6     0
     0    -4     0     6];
c  = [ 1 -1  2 -3 ]';
x0 = [ 10  0  0  0 ]';
tolerance = 1e-6 ;

%% method

fprintf('Gradient method with exact line search\n\n');
fprintf('iter \t f(x) \t\t\t ||grad f(x)||\n\n');

iter = 0 ;

% starting point
x = x0;

while true
    v = 0.5*x'*Q*x + c'*x;
    g = Q*x + c ;
    fprintf('%2.0f \t %2.13f \t %1.9f\n',iter,v,norm(g));

    % stopping criterion
    if norm(g) < tolerance
       break
    end

    %   search direction
    d = -g;

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

    %   new point
    x = x + t*d ;
    iter = iter + 1 ;
end
```

%% Unconstrained optimization -- Exercise 7.2

```
clear; close all; clc;

%% data

% the objective function is defined in f.m

alpha = 0.1;
gamma = 0.9;
tbar = 1;
```

```
x0 = [ 0 ; 0];
tolerance = 1e-3 ;

%% method

fprintf('Gradient method with Armijo inexact line search\n\n');
fprintf('iter \t f(x) \t\t ||grad f(x)||\n\n');

iter = 0 ;
x = x0 ;

while true
    [v, g] = f(x);
    fprintf('%2.0f \t %1.9f \t %1.7f\n',iter,v,norm(g));

    % stopping criterion
    if norm(g) < tolerance
        break
    end

    % search direction
    d = -g;

    % Armijo inexact line search
    t = tbar ;
    while f(x+t*d) > v + alpha*g'*d*t
        t = gamma*t ;
    end

    % new point
    x = x + t*d ;
    iter = iter + 1 ;
end
```

%% Unconstrained optimization -- Exercise 7.3

```
clear; close all; clc;

%% data

Q = [6     0    -4     0
     0     6     0    -4
    -4     0     6     0
     0    -4     0     6];
c = [ 1 -1  2 -3 ]';
x0 = [ 0 0 0 0 ]';
tolerance = 1e-6 ;

%% method

fprintf('Conjugate Gradient method\n\n');
fprintf('iter \t f(x) \t\t ||grad f(x)||\n\n');

iter = 0;

% starting point
x = x0;

while true
```

```
    v = 0.5*x'*Q*x + c'*x;
    g = Q*x + c ;
    fprintf('%1.0f \t %1.4f \t %1.4e\n',iter,v,norm(g));

    % stopping criterion
    if norm(g) < tolerance
        break
    end

    %   search direction
    if iter == 0
        d = -g;
    else
        beta = (norm(g)^2)/(norm(g_prev)^2);
        d = -g + beta*d_prev;
    end

    %   step size
    t = (norm(g)^2)/(d'*Q*d);

    %   new point
    iter = iter + 1;
    x = x + t*d;
    d_prev = d ;
    g_prev = g ;
end
```

%% Unconstrained optimization -- Exercise 7.4

```
clear; close all; clc;

%% data

% the objective function is defined in f.m

alpha = 0.1;
gamma = 0.9;
tbar = 1;
x0 = [ 0 0 ]';
tolerance = 1e-3 ;

%% method

fprintf('Newton method with line search\n\n');
fprintf('iter \t f(x) \t\t ||grad f(x)||\n\n');

iter = 0 ;
x = x0 ;

while true
    [v, g, H] = f(x);
    fprintf('%1.0f \t %1.7f \t %1.4e\n',iter,v,norm(g));

    % stopping criterion
    if norm(g) < tolerance
        break
    end

    % search direction H*d = -g
    d = -H\g;
```

```
    % Armijo inexact line search
    t = tbar ;
    while f(x+t*d) > v + alpha*g'*d*t
        t = gamma*t ;
    end

    % new point
    x = x + t*d ;
    iter = iter + 1 ;

end
```

%% Unconstrained optimization -- Exercise 7.5

```
clear; close all; clc;

%% data

% the objective function is defined in f.m

x0 = [ 0 ; 0 ];
t0 = 5 ;
beta = 0.5 ;
epsilon = 1e-5 ;

D = [ 1 0 -1 0 ;
      0 1 0 -1 ] ;

% D = [ 1 0 -1 ;
%       0 1 -1 ] ;

%% method

fprintf('Directional direct-search method\n\n');
fprintf(' x(1) \t\t x(2) \t\t f(x) \n\n');

x = x0;
t = t0;

v = f(x) ;
fprintf('%1.6f \t %1.6f \t %1.6f\n',x(1),x(2),v);
plot(x(1), x(2),'r.');
axis([-6 6 -6 6])
hold on ;
pause
iter = 0;

while t > epsilon
    iter = iter + 1 ;
    newv = v ;
    i = 0 ;
    while (newv >= v) && (i < size(D,2))
        i = i + 1 ;
        newx = x+t*D(:,i);
        newv = f(newx) ;
        if newv >= v
            plot(newx(1),newx(2),'bs');
            pause
        end
```

```
    end
    if newv < v
        x = newx ;
        v = newv ;
        plot(x(1), x(2),'r.');
        fprintf('%1.6f \t %1.6f \t %1.6f\n',x(1),x(2),v);
        pause
    else
        t = beta*t ;
    end
end
```

8 Solution methods for constrained optimization:

%% Constrained optimization -- Exercise 8.2

```
clear; close all; clc;

%% data

global Q c A b eps;

%% data

Q = [ 1 0 ; 0 2 ] ;
c = [ -3 ; -4 ] ;
A = [-2 1 ; 1 1 ; 0 -1 ];
b = [ 0 ; 4 ; 0 ];

tau = 0.1 ;
eps0 = 5 ;
tolerance = 1e-6 ;

%% method

fprintf('Penalty method\n\n');
fprintf('iter \t eps \t\t x(1) \t\t x(2) \t\t max(Ax-b)\n\n');

options = optimoptions('fminunc','GradObj','on',...
    'Algorithm','quasi-newton','Display','off');

eps = eps0;
x = [0;0];
iter = 0;

while true
    x = fminunc(@p_eps,x,options);
    infeas = max(A*x-b);
    fprintf('%2.0f \t %1.2e \t %1.6f \t %1.6f \t %1.3e\n',iter,eps,x(1),x(2),infeas);
    if infeas < tolerance
        break
    else
        eps = tau*eps;
        iter = iter + 1 ;
    end
end
```

```
%% penalized function

function [v,g] = p_eps(x)

    global Q c A b eps;

    v = 0.5*x'*Q*x + c'*x ;
    g = Q*x + c ;

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

end
```

%% Constrained optimization -- Exercise 8.3

```
clear; close all; clc;

%% data

global Q c A b eps;

Q = [ 1 0 ; 0 2 ] ;
c = [ -3 ; -4 ] ;
A = [-2 1 ; 1 1 ; 0 -1 ];
b = [ 0 ; 4 ; 0 ];

delta = 1e-6 ;
tau = 0.1 ;
eps1 = 1 ;
x0 = [ 1 ; 1 ];

%% method

fprintf('Logarithmic barrier method\n\n');
fprintf('eps \t\t x(1) \t\t x(2) \t\t gap \n\n');

options = optimoptions('fminunc','GradObj','on',...
    'Algorithm','quasi-newton','Display','off');

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

while true
    x = fminunc(@logbar,x,options);
    gap = m*eps;
    fprintf('%1.2e \t %1.6f \t %1.6f \t %1.2e\n',eps,x(1),x(2),gap);
    if gap < delta
        break
    else
        eps = eps*tau;
    end
end

%% logarithmic barrier function

function [v,g] = logbar(x)
```

```
    global Q c A b eps

    v = 0.5*x'*Q*x + c'*x ;
    g = Q*x + c ;

    for i = 1 : length(b)
        v = v - eps*log(b(i)-A(i,:)*x) ;
        g = g + (eps/(b(i)-A(i,:)*x))*A(i,:)' ;
    end

end
```

9 Multiobjective optimization

%% Multiobjective optimization -- Exercise 9.1

```
close all; clear; clc;

%% data

C = [ 1  2 -3 ;
     -1 -1 -1 ;
     -4 -2  1 ];

A = [ 1  1  1 ;
      0  0  1 ;
      -eye(3) ];

b = [ 10 ; 5 ; 0 ; 0 ; 0 ] ;

% given point

% y = [ 5 ; 0 ; 5 ] ;
% y = [ 4 ; 4 ; 2 ] ;
y = [ 1 ; 4 ; 4 ] ;

%% solve the problem

n = size(C,2);
p = size(C,1);
m = size(A,1);

% check if y is a minimum

c = [zeros(n,1) ; -ones(p,1)] ;
P = [C eye(p);
    A zeros(m,p) ;
    zeros(n,n) -eye(p)] ;
q = [C*y ; b ; zeros(p,1)] ;
options = optimset('Display','off');
[~,v_minimum] = linprog(c,P,q,[],[],[],[],[],options)

% check if y is a weak minimum
```

```
c = [ zeros(n,1) ; zeros(p,1) ; -1 ] ;
P = [zeros(p,n) -eye(p) ones(p,1) ;
    C eye(p) zeros(p,1) ;
    A zeros(m,p) zeros(m,1);
    zeros(n,n) -eye(p) zeros(p,1) ] ;
q = [zeros(p,1) ; C*y ; b ; zeros(p,1)] ;
[~,v_weak_minimum] = linprog(c,P,q,[],[],[],[],[],options)
```

%% Multiobjective optimization -- Exercise 9.4

```
close all; clear; clc; hold on

%% data

A = [ -2   1 ;
     -1  -1 ;
      5  -1 ];
b = [ 0 ; 0 ; 6 ] ;

%% plot the feasible region

plot([0 2],[0 4],'-k');
plot([2 1],[4 -1],'-k');
plot([1 0],[-1 0],'-k');

%% solve the scalarized problem with 0 < alfa < 1

options = optimset('Display','off');
for alfa = 0.001 : 0.001 : 0.999
    x = linprog([1 ; 1-2*alfa],A,b,[],[],[],[],options) ;
    plot(x(1),x(2),'g.');
end

%% solve the scalarized problem with alfa = 0

alfa = 0;
x0 = linprog([1 ; 1-2*alfa],A,b,[],[],[],[],options) ;
plot(x0(1),x0(2),'ro');

%% solve the scalarized problem with alfa = 1

alfa = 1;
x1 = linprog([1 ; 1-2*alfa],A,b,[],[],[],[],options) ;
plot(x1(1),x1(2),'bo');
```

%% Multiobjective optimization -- Exercise 9.5

```
close all; clear; clc; hold on

%% data

A = [ -1   0 ;
      0  -1 ;
      1   1 ];
b = [ 0 ; 0 ; 2 ] ;
```

```
%% plot the feasible region

plot([0 0],[0 2],'-k');
plot([0 2],[2 0],'-k');
plot([2 0],[0 0],'-k');

%% solve the scalarized problem with 0 < alfa < 1

options = optimset('Display','off');
for alfa = 0.001 : 0.001 : 0.999
    x = quadprog([2*alfa 0 ; 0 2*alfa],[1-3*alfa ; 0],A,b,...
        [],[],[],[],[],options) ;
    plot(x(1),x(2),'g.');
end

%% solve the scalarized problem with alfa = 0

alfa = 0;
x0 = quadprog([2*alfa 0 ; 0 2*alfa],[1-3*alfa ; 0],A,b,...
    [],[],[],[],[],options) ;
plot(x0(1),x0(2),'ro');

%% solve the scalarized problem with alfa = 1

alfa = 1;
x1 = quadprog([2*alfa 0 ; 0 2*alfa],[1-3*alfa ; 0],A,b,...
        [],[],[],[],[],options) ;
plot(x1(1),x1(2),'bo');
```

%% Multiobjective optimization -- Exercise 9.6

```
close all; clear; clc; hold on

%% data

A = [ 0 -1 ;
     -2  1 ;
      2  1 ];
b = [ 0 ; 0 ; 4 ] ;

%% plot the feasible region

plot([0 2],[0 0],'-k');
plot([2 1],[0 2],'-k');
plot([1 0],[2 0],'-k');

%% solve the scalarized problem with 0 <= alfa <= 1

options = optimset('Display','off');
for alfa = 0 : 0.001 : 1
    x = quadprog([2 0 ; 0 2],[8*alfa-6 ; -4],A,b,...
        [],[],[],[],[],options) ;
    plot(x(1),x(2),'g.');
end
```

```
%% Multiobjective optimization -- Exercise 9.7

close all; clear; clc;

%% data

C = [ 1  2 -3 ;
     -1 -1 -1 ;
     -4 -2  1 ];

A = [ 1  1  1 ;
      0  0  1
      -eye(3) ];

b = [ 10 ; 5 ; 0 ; 0 ; 0 ] ;

%% ideal point

p = size(C,1);
n = size(C,2);
m = size(A,1);
options = optimset('Display','off');

z = zeros(p,1) ;
for i = 1 : p
    [~,z(i)] = linprog(C(i,:)',A,b,[],[],[],[],[],options);
end
z

%% goal method

% 1-norm

gm1 = linprog([zeros(n,1);ones(p,1)], [C -eye(p); -C -eye(p); A zeros(m,p)],[z;-z;b],...
    [],[],[],[],[],options);
gm1 = gm1(1:n)


% 2-norm

gm2 = quadprog(C'*C,-C'*z,A,b,[],[],[],[],[],options)

% inf-norm

[gminf, vinf] = linprog([zeros(n,1);1], [C -ones(p,1); -C -ones(p,1); A zeros(m,1)],[z;-z;b],...
    [],[],[],[],[],options);
gminf = gminf(1:n)


% check if gminf is a minimum

c = [zeros(n,1) ; -ones(p,1)] ;
P = [C eye(p);
     A zeros(m,p) ;
     zeros(n,n) -eye(p)] ;
q = [C*gminf ; b ; zeros(p,1)] ;
[~,v_minimum] = linprog(c,P,q,[],[],[],[],[],options)
```

10 Game theory

%% Noncooperative game theory -- Exercise 10.2

```
clear; close all; clc;
format rat;

%% cost matrix

C = [ 1  2 3
      3 -1 3
      3  2 1];

%% solve the LP problem

[m,n] = size(C) ;

[sol,v,~,~,lambda] = linprog([zeros(m,1);1],...
    [C' -ones(n,1)], zeros(n,1),...
    [ones(1,m) 0], 1,...
    [zeros(m,1); -inf], []);

% Nash equilibrium

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

%% Noncooperative game theory -- Exercise 10.5

```
clear; close all; clc;
global C1 C2

%% data

C1 = [ 3  3
       4  1
       6  0 ];

C2 = [ 3  4
       4  0
       3  5 ] ;

[m,n] = size(C1) ;

%% check if w is a Nash equilibrium

w = [ 1/3 1/3 1/3 1/2 1/2 ]';

gap(w)
reggap(w,1)
Dgap(w,1,10)

%% find a local minimum of the regularized gap function

fprintf('Regularized gap function - local minimum\n');
```

```matlab
alfa = 1 ;

% find a local minimum
options = optimset('Display','off');
[locmin,optval] = fmincon(@(z) reggap(z,alfa),w,[],[],...
    [ones(1,m) zeros(1,n) ; zeros(1,m) ones(1,n)], [1;1],...
    zeros(m+n,1),[],[],options)

%% try to find a global minimum of the regularized gap function
%  with a multistart approach

fprintf('Regularized gap function - multistart approach\n');

for i = 1 : 100

    % starting point
    x0 = rand(m+n,1);
    x0(1:m) = x0(1:m)/sum(x0(1:m));
    x0(m+1:m+n) = x0(m+1:m+n)/sum(x0(m+1:m+n));

    % find a local minimum
    [locmin,optval] = fmincon(@(z) reggap(z,alfa),x0,[],[],...
        [ones(1,m) zeros(1,n) ; zeros(1,m) ones(1,n)],...
        [1;1],zeros(m+n,1),ones(m+n,1),[],options);
    if optval < 1e-4
        locmin
        optval
        break
    end
end

%% try to find a global minimum of the D-gap function
%  with a multistart approach

fprintf('D-gap function - multistart approach\n');

alfa = 1 ;
beta = 10 ;

for i = 1 : 100

    % starting point
    x0 = rand(m+n,1);
    x0(1:m) = x0(1:m)/sum(x0(1:m));
    x0(m+1:m+n) = x0(m+1:m+n)/sum(x0(m+1:m+n));

    % find a local minimum
    [locmin,optval] = fminunc(@(z) Dgap(z,alfa,beta),x0,options);
    if optval < 1e-4
        locmin
        optval
        break
    end
end
```

$$C = \begin{pmatrix} 6 & 9 & 1 & 4 \\ 3 & 4 & 12 & 7 \end{pmatrix}$$

```
close all;
clear;
clc;


C = [6 9 1 4
     3 4 12 7]

Cr = [9 1 4
      4 12 7]

[m, n] = size(C);


% No Nash equilibrium

% Linear programming

f = [zeros(m,1)
     1];

A = [C' -ones(n,1)];
b = [zeros(n,1)];

Aeq = [ones(m,1)' 0];

beq = 1;

lb = [zeros(m,1)
      -inf];

up = [];

options = optimset('Display', 'off');
[sol,Val,exitflag,output,lambda] = linprog(f,A,b,Aeq,beq,lb,up,options)
x = sol(1:m)
y = lambda.ineqlin
% Player 1
```

No Dominated Strategies

```
% Player 2
```

Strategy 1 is dominated by Strategy 2

```
% Solve The linear programming
```

$$C = \begin{pmatrix} 1 & 5 & 11 & 9 \\ 10 & 9 & 8 & 7 \end{pmatrix}$$

```
close all;
clear;
clc;

C = [1 5 11 9
     10 9 8 7]

Cr = [1 5 11
      10 9 8]

[m,n] = size(C);

f = [zeros(m,1)
     1];

A = [C' -ones(n,1)];
b = [zeros(n,1)]'

Aeq = [ones(m,1)' 0];
beq = 1;

lb = [zeros(m,1)
     -inf];

up = [];

options = optimset('Display', 'off');

[sol,value, flag, output, lambda] = linprog(f,A,b,Aeq,beq,lb,up,options);

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

% No Nash Equilibrium

% Player 1

% Player 2
```

Strategy 4 is dominated by Strategy 3

$$C = \begin{pmatrix} 7 & 15 & 2 & 3 \\ 4 & 2 & 3 & 10 \\ 5 & 3 & 4 & 12 \end{pmatrix}$$

```
close all;
clear;
clc;

C = [7 15 2 3
     4 2 3 10
     5 3 4 12]

Cr = [7 15   3
      4  2  10
      5  3  12]
```

```
[m,n] = size(C);

f = [zeros(m,1)
     1];

A = [C' -ones(n,1)];
b = [zeros(n,1)];

Aeq = [ones(m, 1)' 0];
beq = 1;

lb = [zeros(m,1)
      -inf];
up = [];

options = optimset('Display', 'off');
[sol, value, flag, output, lambda] = linprog(f,A,b,Aeq,beq,lb,up,options);

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

% No Nash Equilibrium

% Player 1
```

No dominated Strategy. Strategy 3, x3 = 0.

```
% Player 2
```

Strategy 3 is dominated by Strategy 4. Strategy 1, y1 = 0. Strategy 3, y3 = 0.

$$C = \begin{pmatrix} 5 & 2 & 11 & 15 \\ 1 & 13 & 5 & 1 \end{pmatrix}$$

```
close all;
clear;
clc;

C = [5 2 11 15
     1 13 5 1]

[m,n] = size(C);


f = [zeros(m,1)
     1];

A = [C' -ones(n,1)];
b = [zeros(n,1)];

Aeq = [ones(m,1)' 0];
beq = 1;

lb = [zeros(m,1)
      -inf];
```

```
up = [];

options = optimset('Display', 'off');
[sol, value, flag, output, lambda] = linprog(f,A,b,Aeq,beq,lb,up,options)

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

% No Nash Equilibrium
% No Strictrly dominance

% Player 2 y1 = 0 and y3 =0
```

$$C = \begin{pmatrix} 3 & 10 & 3 & 8 \\ 13 & 6 & 7 & 2 \end{pmatrix}$$

```
close all;
clear;
clc;

C = [3 10 3 8
     13 6 7 2]

Cr = [ 3 10 3
      10  6 7]

[m,n] = size(C);

f = [zeros(m,1)
     1];

A = [C' -ones(n,1)];
b = [zeros(n,1)];

Aeq = [ones(m,1)' 0];
beq = 1;

lb = [zeros(m,1)
      -inf];
up = [];

options = optimset('Display', 'off');

[sol, value, flag, output, lambda] = linprog(f,A,b,Aeq,beq,lb,up,options);

x = sol(1:m)
y = lambda.ineqlin
% No Nash Equilibrium
% Player 1 No dominated strategy

% Player 2
```

Strategy 4 is dominated by Strategy 2

$$C = \begin{pmatrix} 10 & 7 & 12 & 10 \\ 7 & 10 & 6 & 7 \end{pmatrix}$$

```
close all;
clear;
clc;

C = [10 7  12 10
     7  10 6  7]

[m,n] = size(C);
f = [zeros(m,1)
     1];

A = [C' -ones(n,1)];
b = [zeros(n,1)];

Aeq = [ones(m,1)' 0];
beq = 1;

lb = [zeros(m,1)
     -inf];

up = [];

options = optimset('Display', 'off');
[sol, value, flag, output, lambda] = linprog(f,A,b,Aeq, beq, lb, up, options)
x = sol(1:m)
y = lambda.ineqlin

% No Nash equilibrium
% No dominated strategies
```

1. $C_1 = \begin{pmatrix} 8 & 1 & 3 \\ 6 & 3 & 1 \\ 5 & 2 & 0 \end{pmatrix}$ $\qquad C_2 = \begin{pmatrix} 9 & 5 & 6 \\ 3 & 7 & 8 \\ 1 & 2 & 3 \end{pmatrix}$

```
close all;
clear;
clc;

syms x y x1 x2 x3 y1 y2 y3 mu1 mu2

C1 = [8 1 3
      6 3 1
      5 2 0]

C2 = [9 5 6
      3 7 8
```

```
     1 2 3]

Cr1 = [8 3
      6 1]

Cr2 = [9 6
      3 8]

x = [x1
    x2
    x3];

y = [y1
    y2
    y3];

equation = [ C1*y + mu1 >= 0
           C2'*x + mu2 >= 0
           x.*(C1*y + mu1) == 0
           y.*(C2'*x + mu2) == 0
           x >= 0
           y >= 0
           x1 + x2 + x3 == 1
           y1 + y2 + y3 == 1]
```

```
variables = [x
           y
           mu1
           mu2]

solution = solve(equation, variables)


% Existence of Pure Nash Equilibrium (x1,y1) = (8,9)
% Nash Equilibrium
```

$$\begin{pmatrix} x_1(1) & x_2(1) & x_3(1) & y_1(1) & y_2(1) & y_3(1) & \mu_1(1) & \mu_2(1) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & -1 & -5 \end{pmatrix}$$

$$\begin{pmatrix} x_1(2) & x_2(2) & x_3(2) & y_1(2) & y_2(2) & y_3(2) & \mu_1(2) & \mu_2(2) \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & -5 & -1 \end{pmatrix}$$

$$\begin{pmatrix} x_1(3) & x_2(3) & x_3(3) & y_1(3) & y_2(3) & y_3(3) & \mu_1(3) & \mu_2(3) \end{pmatrix} = \begin{pmatrix} \frac{1}{5} & 0 & \frac{4}{5} & \frac{1}{4} & \frac{3}{4} & 0 & -\frac{11}{4} & -\frac{13}{5} \end{pmatrix}$$

```
% Dominance
```

Player 1

Strategy 3 is dominated by Strategy 2.

Player 2

Strategy 2 is dominated by Strategy 3