Appendix A

MATLAB Programs for Linear Programming Problems
function [pnuinf,pnuinfl,mz,xxx]=maximize(m,n,A,b,c,xu)
%maximize(m,n,A,b,c,xu) generates random points using a certain probability
%distribution with considering upper bounds which are produced by my_upfun
%and selects the best feasible solution of primal problem which is
%maximization.
%==============================================================================
% Initialization
% tot=0; k = 1; kk = 1; pnuinf=0; j=1; pnuinfl=0; mz=-inf; add=0;x=[];
clear functions
% 100 feasible solutions are generated under a certain probability
% distribution.
% while k <= 100
% for j = 1 : n
% x(j,k)= xu(j)*gamrnd(1,2);% xu(j)* betarnd(.75,.5);% .5*xu(j)+randn*(.5^ .5*xu
%{j});% 
% % geornd(.5)*xu(j);% poissrnd(.5*xu(j));% wblrnd(xu(j),3.5);%
% % rand*xu(j);% raylrnd(xu(j));% 
% % nbinrnd(2,.5);% xu(j)*chidrnd(1);% trnd(5)+xu(j);%
% %unifrnd(0,xu(j));% randraw('normaltrunc',[0,xu(j),xu{j},xu(j)],1); %
% end
% % Testing the feasibilty
% if x(:,k)>=0
% AX = A * x(:,k);
% if AX <= b
% z(k) = c' * x(:,k);
% add=add+1;
% k = k + 1;
% tot=tot+1;
% else
% pnuinf=pnuinf+1;
% x(:,k) =[];
% if pnuinf >= 50000 break \% The number of infeasible points are\%
% controlled
% end
% else
% pnuinfl=pnuinfl+1;
% x(:,k)=[];
% if pnuinfl >= 50000 break \% The number of infeasible (negative) poits\%
% are controlled.
% end
% \end
[mez,I] = max(z);
% the maximum of primal objective function value is selected also
% the associated solution in current iteration and forwarded it
xxx=x(:,I);
% to the main program.
function [dnuinf, dnuinf1, ming, yyy] = minimize(m, n, A, b, c, yl)
% minimize(m, n, A, b, c, yl) generates random points using a certain probability
% distribution with considering lower bounds which are produced by my_lowfun
% and select the best feasible solution of dual problem which is
% minimization.
%==============================================================================
% Initialization
k = 1; kk = 1; dnuinf = 0; dnuinf1 = 0; add = 0; tot = 0; y = [];
clear functions
% 100 feasible solutions are generated under a certain probability
% distribution.
while k <= 100
    for j = 1 : m
        y(j, k) = geornd(.5) + yl(j); % yl(j) + betarnd(.05, .05); % .5*yl(j) + randn;
% wblrnd(yl(j), 2.5); % gamrnd(1, 1.5) + yl(j); % raylrnd(yl(j)); %
% poissrnd(1) + yl(j); % wishrnd(1, 2); % nbinrnd(20, .5); % frnd(1, 1);
% randraw('normaltrunc',[0, yl(j), yl(j), 1]); %
    end
% Testing the feasibility
    if y(:, k) >= 0
        g(k) = b' * y(:, k);
        add = add + 1;
        k = k + 1;
        tot = tot + 1;
    else
        dnuinf = dnuinf + 1;
        y(:, k) = [];
        if dnuinf >= 50000 dnuinf % The number of infeasible points are controlled
            break
        end
    end
else
    dnuinf1 = dnuinf1 + 1;
    y(:, k) = [];
    if dnuinf1 >= 50000 break % The number of infeasible (negative) points are controlled.
        end
    end
end
[ming(kk), I] = min(g); % the minimum of dual objective function value is selected also
% the associated solution in current iteration and forwarded it
yyy = y(:, I); % to the main program.
function [yl]=low(m,n,A,c)
% low(m,n,A,c) produces lower bound of decision variables of a dual
% problem in linear programming problem.

A=A';
yyyf=[];
for j = 1 : m
    k=1;
    for i=1:n
        if A(i,j) == 0
            else o(i)=c(i)/A(i,j);
        end
    end
    yy=[]; yyf=[];
% Find intersections of constraints mutually.
    for ii = 1 : n-1
        for jj = ii+1 : n
            qq = [A(ii,:);A(jj,:)]\[c(ii);c(jj)];
            yy(:,k) = qq;
            if qq ~= inf
                C=A*qq;
                if (C-c) >= 0
                    yyf=[yyf yy(:,k)];
                    yy(:,k)=[];
                    k=k-1;
                end
            end
        end
    k=k+1;
end
% Find intersections of constraints in triple.
% if m >= 3
% for u = 1 : m-2
%    for v = u+1 : m-1
%        for w = v+1 : m
%            qqq = [A(u,:);A(v,:);A(w,:)]\[c(u);c(v);c(w)];
%            yy(:,k) = qqq;
%            k = k+1
%        end
%    end
% end
% if m >= 3
% for u = 1 : m-2
%    for v = u+1 : m-1
%        for w = v+1 : m
%            qqq = [A(u,:);A(v,:);A(w,:)]\[c(u);c(v);c(w)];
%            yy(:,k) = qqq;
%            k = k+1
%        end
%    end
% end
lb = o;

[ro col]=size(yy);
for r=1:col
    lb;
    yy(:,r);
    lb=[lb,yy(j,r)];
end
my = zeros(m,1);
ll=1;

% The infeasible intersections are ignored.
while ll <= length(lb)
    my(j) = lb(ll);
    if A * my >= c
        ll = ll+1;
        yyyf(:,j)=my;
    else
        lb(ll) = inf;
        ll = ll + 1; % the number of infeasible points are counted.
    end
end

if size(yyf) ~=[0 0]
    llb=[lb,yyf(j)];
else llb=lb;
end
yl(j)=min(llb);
if (yl(j)<0) | (yl(j)==inf)
    yl(j)=0 ;
end
end

for p=1:m
    if size(yyyf)~= [0 0]
        ylow = [yl(p) yyyf(p,:)];
        yl(p)=min(ylow);
        if (yl(p)<0) | (yl(p) ==0) yl(p)=0 ;
    end
end
end
yl % lower bounds are printed.
function \[xu]=upp(m,n,A,b)
% upp(m,n,A,b) produces upper bound of decision variables of a primal
% problem in linear programming problem.
%==========================================================================
% Initialization
x = (1:n)*0;
x = x';
t = 0;
for i = 1 : m
  for j = 1 : n
    if A(i,j) <= 0
      t = t+1;
    end
  end
end
if t > 0
  for j = 1 : n
    o=zeros(1,m);
    up=[];
    for i = 1 : m
      if (A(i,j) == 0) | (A(i,j) < 0)
        o(i)=A(i,j)(i);
      end
    end
    if for i=1:m
      % find the intersections of constraints mutually.
      k = 1;
      xx=zeros(n,1);
      for ii = 1 : n-1
        for jj = ii+1 : n
          gg = [A(:,ii),A(:,jj)];
          if (gg ~= inf) | (gg ~= -inf)
            xx([ii jj],k)=gg;
          end
          k = k+1;
        end
      end
      % Find the intersections of constraints in triple.
      % if n >= 3;
      % for u=1 : n-2
      %   for v=u+1 : n-1
      %     for w=v+1 : n
      %       ggg=[A(:,u),A(:,v),A(:,w)];
      %       if ggg ~= inf
      %         xx([u v w],k) = ggg;
      %       end
      %       k=k+1;
      %     end
      %   end
      % end
end
end
up = [o xx(j,:)];

% The infeasible intersection points are ignored.
% mx=zeros(n,1);
% ll=1;
% while ll <= length(up)
%     mx(j)=up(ll);
%     if A*mx <= b  ll=ll+1;
%     else up(ll)=0;  ll=ll+1;
% end
% xu(j)=max(up);
end

xu
else
    for j = 1 : n
        for i = 1 : m
            o(i) = b(i)/A{i,j};
        end
        xu(j) = min(o);
    end
    xu % upper bounds are printed.
end
function prdu(m,n,A,b,c,nu)
% prdu(m,n,A,b,c,nu) is the program to the stochastic search algorithm
% that invokes the functions which are "my_lowfun", "my_upfun", "minimize"
% and "maximize". This program restarts the minimize and maximize functions.
% nu is the number of restarting. This MATLAB function finds the best
% solutions to the primal and dual linear programming problems which are called
% near-optimal solutions.

% Here, m is the number of constraints, n is the number of variables,
% A is an m by n matrix including the variable coefficients in constraints
% b is the RHS vector and c is an n by 1 vector of coefficient variables
% in the objective function of primal problem.
%==========================================================================
t=cputime;
% Initialization
ttot=1;gggl=1000;

[yl]=my_lowfun(m,n,A,c);
[xu]=my_upfun(m,n,A,b);

% Restart the minimize and maximize functions while the length of
% gap(difference between primal and dual objective function values) is
% bigger than or equal to 0.001 (the initial value of gap is arbitrary) and
% the number of restart does not reach to its initial value.
while (gggl >= 0.001) & (ttot <= nu)
    [pnuinf,pnuinf1,mz,xxx]=maximize(m,n,A,b,c,xu);
    ppnuinf(ttot)=pnuinf;
    ppnuinf1(ttot)=pnuinf1;
    mmax(ttot)=mz;
    xxop(:,ttot)=xxx;
    [dnuinf,dnuinf1,ming,yyy]=minimize(m,n,A,b,c,yl);
    ddnuinf(ttot)=dnuinf;
    ddnuinf1(ttot)=dnuinf1;
    mmin(ttot)=ming;
    yyop(:,ttot)=yyy;
    zl=max(mmax);
    gl=min(mmin);
    gggl=gl-zl;
    ttot=ttot+1;
end
% output the near-optimal objective function values of primal and dual
% problems with their associated solution.
[maxprimal,P]=max(mmax)
xop=xxop(:,P)
[mindual,D]=min(mmin)
yop=yyop(:,D)
MGAP1=(mindual-maxprimal)
numberrestart=ttot-1
sumprimalinf=sum(ppnuinf);

primalinftnax=ppnuinf(P);
sumprimalinfl=sum(ppnuinf1);
primalinfmax=ppnuinf1(P);
sumdualinf=sum(ddnuinf);
dualinfmin=ddnuinf(D);
sumdualinfl=sum(ddnuinf1);
dualinfmin1=ddnuinf1(D);

for pt=2:ttot-1
    PN(pt)=pt*100;
    if mmax(pt)>pf(pt-1)
        pf(pt)=mmax(pt);
    else
        pf(pt)=pf(pt-1);
    end
end

% Output the graph of near-optimal values of primal and dual problems in
% iterations.

df(l)=mmin(l);DN(1)=100;
for dt=2:ttot-1
    DN(dt)=dt*100;
    if minin(dt)<df (dt-1)
        df(dt)=nimin(dt);
    else
        df(dt)=df(dt-1);
    end
end
plot(PN,pf,DN,df)
grid on

% Compute the number of infeasible points in the all iterations and output
% the number of infeasible points in that iteration which is given the near
% -optimal value for both primal and dual problems.

[maxprimal,P]=max(mmax)
time=cputime-t
ISCprimal=primalinftnax+primalinfmax
TOTprimal=sum(primalinf)+sum(primalinf1)
ISCdual=dualinfmin+dualinfmin1
TOTdual=sum(dualinf)+sum(dualinf1)
numbergeneratedpoints=ISCprimal+ISCdual+200
% This script runs the prdu function. Different examples are mentioned.
clc

% Seal's Example
%prdu(3,4, [.25 -60 -.04 9; .5 -90 -.02 3; 0 0 1 0], [0; 0; 1; .75; -150; .02; -6], 50)

% Kuhn's Example
prdu(3,4, [-2 -9 1 9; .33 1 -.33 -2; 2 3 -1 -12], [0; 0; 2; 2; 3; -1; -12], 50)

% Klee & Minty Example.
%prdu(4,4, [16 8 4 1; 8 4 1 0; 4 1 0 0; 1 0 0 0], [625; 125; 25; 5], [8; 4; 2; 1], 50)

% Further Example
%prdu(2,2, [5 20; 10 15], [400; 450], [45; 80], 50)
Appendix B

MATLAB Programs for Quadratic Programming Problems
function [pnuinf, pnuinf1, mz, xxx] = maxquad(m, n, A, b, c, Q, code2, xu, xl)
%maxquad(m, n, A, b, c, Q, code2, xu, xl) generates random points using a certain probability
%distribution with considering lower/upper bounds which are produced by
%my_lowfun/my-upfun and select the best feasible solution of the quadratic
%programming problem which is maximization.
%==========================================================================
% Initialization
k = 1; kk = 1; pnuinf = 0; j = 1; pnuinf1 = 0; mz = -inf; x = [];
clear functions

% 100 feasible solutions are generated under a certain probability
% distribution according to the constraints inequalities "\geq" or "\leq".

while k <= 100
    for j = 1 : n
        if code2 == 1
            x(j, k) = geornd(.5); % xu(j)* betarnd(.5,.75);% trnd(5)+xu(j);
            % xu(j)*gamrnd(.5,1);% poissrnd(xu(j));% .5*xu(j)+randn*.5*.5*xu(j));%
            % rand*xu(j);% wblrnd(xu(j),3.5);% raylrnd(xu(j));% nbinrnd(2,.5);%
            % poissrnd(5)+xu(j);% xu(j)*chi2rnd(1);% trnd(5)+xu(j);% unifrnd(0,xu(j));%
        else
            x(j, k) = xl(j) + gamrnd(.25, 1); % xl(j)+ betarnd(.75,.75);%
            % geornd(.95)+xl(j);% poissrnd(xl(j));% wblrnd(xl(j),.5);%
            % rand*xl(j);% raylrnd(xl(j));% nbinrnd(2,.5);% poissrnd(5)+xl(j);%
            % xl(j)*chi2rnd(1);% trnd(5)+xl(j);% unifrnd(0,xl(j));%
        end
        Testing the feasibility
        if x(:, k) >= 0
            AX = A * x(:, k);
            if code2 == 1
                if AX <= b
                    z(k) = c' * x(:, k) + 0.5 * x(:, k)' * Q * x(:, k);
                    k = k + 1;
                else
                    pnuinf = pnuinf + 1;
                    x(:, k) = [];
                    if pnuinf >= 50000
                        pnuinf
                        break
                    end
                end
            end
        else
            if AX >= b
                z(k) = c' * x(:, k) + 0.5 * x(:, k)' * Q * x(:, k);
                k = k + 1;
            else
                pnuinf = pnuinf + 1; /* The number of infeasible points are controlled */
                x(:, k) = [];
            end
        end
    end
end
if pnuinf >= 50000
    pnuinf
    break
end
end % (if AX >= b)
end % (if code2 == 1)

else % (if x(:,k) >= 0)
    pnuinfl = pnuinfl + 1;
    x(:,k) = [];
    if pnuinfl >= 50000
        pnuinfl
        break
    end
end % (if x(:,k) >= 0)
end % (while k <= 100)

[mz, I] = max(z);

xxx = x(:, I);
function [pnuinf,pnuinf1,mz,xxx]=minquad(m,n,A,b,c,Q,code2,xu,xl)
%
%minquad(m,n,A,b,c,Q,code2,xu,xl) generates random points using a certain probability
%distribution with considering lower/upper bounds which are produced by
%my_lowfun/my-upfun and select the best feasible solution of the quadratic
%programming problem which is minimization.
%
% Initialization
k = 1; kk = 1; pnuinf=0; j=1; pnuinf1=0; mz=+inf; x=[];
%clear functions
%
% 100 feasible solutions are generated under a certain probability
% distribution according to the constraints inequalities ">=" or "<=".

while k <= 100
    for j = 1 : n
        if code2 == 1
            x(j,k)= xu(j)*gamrnd(.05,1);% xu(j)*geornd(.5);
            % xu(j)*betarnd(.05,.05);% xu(j)+randn*((xu(j))^-.5);
            % xu(j)+trnd(1);% xu(j)*wblrnd(.05,.5);% rand*xu(j);
            % poissrnd(.25*xu(j));% raylrnd(xu(j));% nbinrnd(2,.5);
            % xu(j)*chi2rnd(1);% trnd(5)+xu(j);% unifrnd(0,xu(j));
        else
            x(j,k)= xl(j)+betarnd(.05,.75);% wblrnd(xl(j),3.5);
            % geornd(.5)+xl(j);% xl(j)+gamrnd(.25,1);% poissrnd(xl(j));
            % rand*xl(j);% raylrnd(xl(j));% nbinrnd(2,5);% poissrnd(5)+xl(j);
            % xl(j)*chi2rnd(1);% trnd(5)+xl(j);% unifrnd(0,xl(j));
        end
    end %for j=1:n

% Testing the feasibility
if x(:,k)>=0
    AX = A * x(:,k);
    if code2 == 1
        if AX <= b
            z(k) = c' * x(:,k) + 0.5 * x(:,k)' * Q * x(:,k);
            k = k + 1;
        else
            pnuinf=pnuinf+1; % The number of infeasible points are controlled
            x(:,k)=[];
            if pnuinf >= 50000
                pnuinf
                break
            end %if pnuinf
        end %if AX<=b
        else
            if AX >= b
                z(k) = c' * x(:,k) + 0.5 * x(:,k)' * Q * x(:,k);
                k = k + 1;
            else
                pnuinf=pnuinf+1;
            end
        end
    end
end
x(:,k) =[];
if pnuinf >= 50000
    pnuinf
    break
end
end %if AX>=b
end %if code2==1
else %if x(:,k)>=0
    pnuinf1=pnuinf1+1;
    x(:,k)= [];
    if  pnuinf1 >= 50000
        pnuinf1
        break
    end
end %if x(:,k)>=0
end %if while k<100
[mz,I] = min(z);
xxx=x(:,1);
function quad(m,n,A,b,c,Q,codel,code2,nu)
% quad(m,n,A,b,c,Q,codel,code2,nu) is a MATLAB program to the stochastic
% search algorithm for Quadratic programming problem that invokes the
% functions which are "quadlowfun", "quadupfun", "minquad" and "maxquad".
% This program restarts the minquad/maxquad functions. nu is the number of
% restarting. codel and code2 are switch to discriminant of the sign of
% inequalities of constraints, "=" or ">=".
% This MATLAB function finds the best solution to the Quadratic programming
% problems which is called near-optimal solution.
%The Quadratic programming problem we consider in this MATLAB function is:
%Minimum or Maximum 1/2 X'QX+ c'X,
% s. t. AX >= or <= b,
% X >= 0.
% X and c are in R^n, A is in R^m*n, b is in R^m and Q is a semi-positive n by n matrix.
%-------------------------------------------------------------------------------
t=cputime;
%Initilization
pnuinf=0; ttot=1;
% Invoke the functions which provide lower and upper bounds.
[xu]=quadupfun(m,n,A,b,code2);
[xl]=quadlowfun(m,n,A,b,code2);
% Restart the minquad/maxquad function while the number of restarts does
% not reach to its initial value.
while (ttot <= nu)
    if codel == 1
        [pnuinf,pnuinf1,mz,xxx]=minquad(m,n,A,b,c,Q,code2,xu,xl);
        mmin(ttot)=mz;
    else
        [pnuinf,pnuinf1,mz,xxx]=maxquad(m,n,A,b,c,Q,code2,xu,xl);
        mmax(ttot)=mz;
    end
    ppnuinf(ttot)=pnuinf;
    ppnuinf1(ttot)=pnuinf1;
    xxop(:,ttot)=xxx;
    ttot=ttot+1;
    if (pnuinf>=50000) | (pnuinf1>=50000)
        break
    end
end
% Determine the best solution as the near-optimal objective function value to
% the quadratic problem with its associated solution among all restarts.
    if codel == 1
        [minq,P]=min(mmin)
    else
        [maxq,P]=max(mmax)
    end
    xop=xxop(:,P)
    numberrestart=ttot-1
suminf = sum(ppnuinf); infmax = pppnuinf(P); suminf1 = sum(ppnuinf1); inflmax = pppnuinf1(P); if code1 == 1 pf(1) = mininf(1); else pf(1) = maxinf(1); end
% Output the graph of near-optimal values of Quadratic problem in % iterations.
P(N(1)) = 100;
for pt = 2:ttot-1
    PN(pt) = pt*100;
    if code1 == 1
        if mininf(pt) < pf(pt-1)
            pf(pt) = mininf(pt);
        else
            pf(pt) = pf(pt-1);
        end
    else
        if maxinf(pt) > pf(pt-1)
            pf(pt) = maxinf(pt);
        else
            pf(pt) = pf(pt-1);
        end
    end % if code1 == 1
end % for pt

plot(PN, pf)
grid on

% Output the number of infeasible points in the all iterations and output % the number of infeasible points in that iteration which is given the near % optimal value problem. ISC = infmax + inflmax T = suminf + suminf1
time = cputime - t
function [xl]=low(m,n,A,b,code2)
% low(m,n,A,b,code2) produces lower bound of decision variables of a
% quadratic programming problem with ">=" inequalities.
% Initialization
xxxf=[];
k=1;
xxf=[];
xx=zeros(n,1);

% Find intersections of constraints mutually.
for ii = 1 : n-1
    for jj = ii+1 : n
        gg = [A(:,ii),A(:,jj)]\b;
        xx([ii jj],k) = gg;
        if (gg ~= inf) & (gg ~= -inf)
            B = A * xx(:,k);
            if (b-B) >= 0
                if (B-b) >= 0
                    xxf=[xxf xx(:,k)];
                end
            end
        end
        k=k+1;
    end
end

for j = 1 : n
    for i=1:m
        if A(i,j)==0  o(i)=inf;
        else  o(i)=A(i,j)\b(i);
    end
end

lb = o;
[ro col]=size(xx);
for r=1:col
    lb=[lb,xx(j,r)];
end
mx = zeros(n,1);
l1=1;
The infeasible intersections are ignored.
while l1 <= length(lb)
    mx(j) = lb(l1);
    if mx >= 0
        if A * mx >= b
            ll = l1+1;
            xxf=[xxf mx];
        else lb(l1) = inf;
            ll = ll + 1;
        end
    else
        lb(l1)=inf;  % the number of infeasible points are counted.
        ll=ll+1;
    end
end
end % (while)

if (size(xxf) == [0 0]) & (size(xxxf) == [0 0])
    if xxf >= 0
        if A*xxf >= b
            lb=[lb,xxxf(j,:),xxf(j,:)];
        end
    end
end
if xx>=0
    if A*xx >= b
        lb=[lb,xx(j,:)];
    end
end
for iii=1:m-l
    for jjj=iii + l :m
        qq=rA(iii,:)\[b(iii);b(jjj)];
        if qq >= 0
            if A*qq >= b
                lb=[lb,qq(j)];
            end
        end
    end
end
xl(j)=min(lb);
if (xl(j)<0) | (xl(j)== inf)
    xl(j)=0;
end
end %(for j=1:n)
for p=1:n
    if size(xxxf) ==[0 0]
        xlow=[xl(p) xxxf(p,:)];
        xl(p)=min(xlow);
        if (xl(p)<0) | (xl(p)== inf)
            xl(p)=0;
        end
    end
end %(for p=1:n)
if code2 == 2
    xl
end
function [xu]=upp(m,n,A,b,code2)
% upp(m,n,A,b,code2) produces upper bound of decision variables of a
% quadratic programming problem with "<=" inequalities.
%==========================================================================
% Initialization
xxxf=[];
k=1;
xxf=[];
xx=zeros(n,1);
% Find intersections of constraints mutually.
for ii = 1 : n-1
    for jj = ii+1 : n
        gg = [A(:,ii),A(:,jj)]\b;
        xx([ii jj],k) = gg;
        if (gg ~= inf) | (gg ~= -inf)
            B = A * xx(:,k);
            if (b-B) >= 0
                if (B-b) <= 0
                    xxf=[xxf xx(:,k)];
                end
            end
            k=k+1;
        end % (if gg ~= inf |..)
    end % (for jj)
end % (for ii)

for j = 1 : n
    for i=1:m
        if A(i,j)==0 o(i)=inf;
        else o(i)=A(i,j)\b(i);
        end
    end %for i=1:m
    up = o;
    [ro col]=size(xx);
    for r=1:col
        up=[up,xx(j,r)];
    end
    mx = zeros(n,1);
    ll=1;
    % The infeasible intersections are ignored.
    while ll <= length(up)
        mx(j) = up(ll);
        if mx >= 0
            if A * mx <= b
                ll = ll+1;
                xxf=[xxf mx];
            else up(ll) = 0;
                ll = ll + 1;
            end
        else
            up(ll)=0;
        end
        ll=ll+1;
    end
end

xxf
end % (while)

if (size(xxf) == [0 0]) & (size(xxxf) == [0 0])
  if xxf >= 0
    if A*xxf <= b
      up=[up,xxxf(j,:),xxf(j,:)];
    end
  end
end

if xx>=0
  if A*xx <= b
    up=[up,xx(j,:)];
  end
end

for lll=1:m-1
  for jjj=iii+l:m
    qq=[A(iii,:);A(jjj,:)]\[b(iii);b(jjj)];
    if qq >= 0
      if A*qq <= b
        up=[up,qq(j)];
      end
    end
  end
end

xu(j)=max(up);
if (xu(j)<0) | (xu(j)== inf)
xu(j)=0;
end

end % (for j=1:n)

for p=1:n
  if size(xxxf) == [0 0]
    xupp=[xu(p) xxxf(p,:)];
    xu(p)=max(xupp);
    if (xu(p)<0) | (xu(p)== inf)
      xu(p)=0;
    end
  end
end % (for p=1:n)
i f code2 == 1
  xu
end
% This script runs the quad function. Different examples are mentioned.
% clear functions
% code1=1 minimizing, code1=2 maximizing &
% code2=1 less than or equal, code2=2 greater than or equal.

% Example 1
quad(2,2,[1 1;1 0],[5;3],[-8;-16],[2 0;0 8],1,1,10)

% Example 2
quad(5,5,[1 -1 -3 -1 2;-1 -1 2 1 -2;0 -7 -2 -5;-1 5 1 -3 -1;... 1 0 -1 1 -4],[6;2;3;6;4],[-1;1;0;0;1],[3 -1 1 0 0;1 4 0 0 0;... 1 0 4 0 0;0 0 0 8 0;0 0 0 0 4],1,1,50)

% Example 3
quad(4,5,[1 1 -1 0 1;0 1 2 -1 0;1 0 4 -1 -1;1 1 0 2 0],...
[6;8;5;3],[4;3;1;1;0],[2 -1 0 0 0;-1 4 0 0 0;0 0 6 1 0;... 0 0 1 4 2;0 0 0 2 2],1,1,50)
Appendix C
MATLAB Programs for General Non-Linear Programming Problems
function G = constraint(x)
    % (code2==1 inequalities <=) & (code2==2 inequalities >=)
    
    % In this file the constraint functions (g_i(X)>= or <= 0) of 3 examples
    % which are mentioned in the thesis have given.
    %==================================================================
    % (Example No. 4.3) Inqs. are <= , x0=2,2]
    G = [x(1)'-2+x(2)'^2-16; (x(2)-x(1))^2+x(1)-6;x(1)+x(2)-2];
    % (Example No. 4.4) Inqs. are <= , x0=[0,3,1]'
    %G = [x(1)'^2+x(2)'^2-4;x(2)-x(1))^2+x(1)-6;x(1)+x(2)-2];
    % (Example No.4.5) Inqs. are >= , x0=[1,1]
    %G = [-(x(1)'^2+x(2)-x(1)+x(2)'^2];
function \([mz,xx,nuinf,nuinf1]=maxnonlin(n,xu,code2)\)

% In this MATLAB function 100 feasible solutions under certain probability
% distribution considering upper/lower bounds according to the constraint
% inequalities are generated and determine the maximum objective function
% value with its associated solution.

% Initialization

k=1; nuinf=0;nuinf1=0; x=[];
clear functions
while k<=100
    for j=1:n
        if code2==1
            x(j,k)= wblrnd(2*xu(j),2);% xu(j)*gamrnd(2,3);% evrnd(xu(j),1);
            % xu(j)* betarnd(.05,.05);% 1.5*xu(j)+xu(j)*randn;
            % rand*(xu(j));% xu(j)*geornd(.5);% xu(j)*poissrnd(1);
        else
            x(j,k)= xu(j)*betarnd(.75,.25);% gamrnd(1.5, 1);% geornd(.5);
            % 1.5*xu(j)+xu(j)*randn;% poissrnd(xu(j));% rand*(xu(j));
        end
    end
    %Testing the feasibility of generated points according to the
    %constraint inequalities.
    if x(:,k)>=0
        cval=feval(@constraint,x(:,k));
        if code2==1
            if cval<=0
                z(k)=feval(@objective,x(:,k));
                k=k+1;
            else
                nuinf=nuinf+1; % ignoring the infeasible points and
                x(:,k)=[]; % counting that.
                if nuinf>=50000
                    nuinf
                    break
                end
            end %if cval<=0
        else
            if cval>=0
                z(k)=feval(@objective,x(:,k));
                k=k+1;
            else
                nuinf=nuinf+1;
                x(:,k)=[];
                if nuinf>=50000
                    nuinf
                    break
                end
            end %if cval>=0
        end %code2==1
    end
else
    % Determine the maximum objective function value in the current iteration
    % with the associated solution.
    [mz,I]=max(z);
    xx=x(:,I);
end  %while k<=100

nuinf1=nuinf1+1;
x(:,k)=[];
if nuinf1>=50000
    break
end
end  %while k<=100
function [mz, xx, nuinf, nuinf1] = minnonlin(n, xu, code2)
% In this MATLAB function 100 feasible solutions under certain probability
% distribution considering upper/lower bounds according to the constraint
% inequalities are generated and determine the minimum objective function
% value with its associated solution.
%==========================================================================
% Initialization
k = 1; nuinf = 0; nuinf1 = 0; x = []; tt = 0;
clear functions
% Generating random points
while k <= 100
    for j = 1:n
        if code2 == 1
            x(j, k) = geornd(.5); xu(j) * randn; % poissrnd(1); xu(j) * betarnd(1, 1);
            % xu(j) * gamrnd(2, 2); % evrnd(xu(j), 1); rand*(xu(j));
            % wblrnd(xu(j), 2.5); % xu(j) * poissrnd(1);
        else
            x(j, k) = gamrnd(8, .5); % xu(j) * betarnd(1, 1); % evrnd(xu(j), 2);
            % sqrt(wblrnd(xu(j), .5)); % geornd(.75); % lognrnd(xu(j), xu(j));
            % raylnd(1); % wishrnd(1, 1); %
            % randraw('normaltrunc', [0, xu(j), .5 * xu(j), 1, 1]); %
            %.75 * xu(j) + xu(j) * randn; % poissrnd(xu(j)); rand*(xu(j));
        end
    end
% Testing the feasibility of generated points according to the constraint inequalities.
% if x(:, k) >= 0
    cval = feval(@constraint, x(:, k));
% if code2 == 1
    if cval <= 0
        z(k) = feval(@objective, x(:, k));
        k = k + 1;
    else
        nuinf = nuinf + 1; % counting that.
        x(:, k) = [];
        if nuinf >= 50000
            nuinf
            break
        end
    end
% if cval <= 0 else
    if cval >= 0
        z(k) = feval(@objective, x(:, k));
        k = k + 1;
    else
        nuinf = nuinf + 1;
        x(:, k) = [];
        if nuinf >= 50000
            nuinf
            break
    end
end
137
end
end % if cval>=0
end %code2==1
end %x(:,k)>=0
nuinfl=nuinfl+1;
x(:,k)=||;
if nuinfl>=500000
nuinfl
break
end
end %while k<=100

% Determine the minimum objective function value in the current iteration
% with the associated solution.
[mz,I]=min(z);
xx=x(:,I);
function Nonlinear(n,nu,xo,codel,code2)
% Nonlinear(n,nu,xo,codel,code2) is a MATLAB program to the stochastic
% search algorithm for General nonlinear programming problem that invokes the
% functions which are "fsolve", "minnonlin or maxnonlin".
% This program restarts the "minnonlin or maxnonlin" functions. nu is the number of
% restarting. codel and code2 are switch to discriminant of the sign of
% inequalities of constraints, "<=" or ">=" and xo is initial point for solving
% the constraint functions as equality system which is needed in the argument
% of fsolve function .
% This MATLAB function finds the best solution to the general nonlinear programming
% problems which is called near-optimal solution.
% The General nonlinear programming problem we consider in this MATLAB function is:
% Minimum/Maximum f(X),
% s. t. g_i(X) >= or <= 0, for i=1,...,m.
% X >= 0 or X in R^n.
% f and g_i's functions are nonlinear.
%-----------------------------------------------
% Initialization
% t=cputime;
% solve the constraint function considering initial point (xo) invoking fsolve.
% xu=fsolve(@constraint,xo,optimset('fsolve'))
% Restart the minnonlin/maxnonlin function while the number of restarts does
% not reach to its initial value.
% while ttot<=nu
% if codel == 1
% [mz,xx,nuinf,nuinf1]=minnonlin(n,xu,code2);
% mmin(ttot)=mz;
% else
% [mz,xx,nuinf,nuinf1]=maxnonlin(n,xu,code2);
% mmax(ttot)=mz;
% end
% Nuinf(ttot)=nuinf;
% Nuinf1(ttot)=nuinf1;
% xxop(:,ttot)=xx;
% ttot=ttot+1;
% if (nuinf>=50000) | (nuinf1>=500000)
% break
% nuinf
% nuinf1
% end
% end %while ttot<= ...
% Determine the best solution as the near-optimal objective function value to
% the nonlinear programming problem with its associated solution among all restarts.
% Compute the number of infeasible points in the all iterations and output
% the number of infeasible points in that iteration which is given the near
% -optimal value problem.
% if codel==1
% [minob,P]=min(mmin)
else
    [maxob,P]=max(mmax)
end
xop=xxop(:,P)
numberrestart=ttot-1
suminfeasible=sum(Nuinf)
nuinfopt=Nuinf(P)
suminfeasiblel=sum(Nuinf1)
nuinflopt=Nuinf1(P)
if codel==1
    pf(l)=mmin(l);
else
    pf(l)=mmax(l);
end

% Output the graph of near-optimal values of nonlinear problems in % iterations.
PN(1)=100;
for pt=2:ttot-1
    PN(pt)=pt*100;
    if codel==1
        if mmin(pt)<pf(pt-1)
            pf(pt)=mmin(pt);
        else
            pf(pt)=pf(pt-1);
        end
    else
        if mmax(pt)>pf(pt-1)
            pf(pt)=mmax(pt);
        else
            pf(pt)=pf(pt-1);
        end
    end
end %if codel==1
end %for pt=2:

plot(PN,pf)
grid on

time=cputime-t
function Z = objective(x)
% (code1==1 Minimization) & (code1==2 Maximization)

% In this file the objective functions of 3 examples which are mentioned in
% the thesis have given.
%=================================================================================
% Example No. 4.3 Minimization
Z = [(1-x(1))^2-10*(x(2)-x(1))^2+x(1)^2-2*x(1)*x(2)+exp((-x(1)-x(2)))];

% Example No. 4.4 Maximization
%Z = [x(1)^3+2*x(2)^2*x(3)+2*x(3)];

% Example No. 4.5 Minimization
%Z = [(x(1)-2)^2+(x(2)-1)^2];
% This script runs the Nonlinear function. Different examples are mentioned.

clc
clear functions
% code1=1 minimizing, code1=2 maximizing &
% code2=1 less than or equal, code2=2 greater than or equal.

Nonlinear(2,10,[1,1],1,1) %Example 4.3
%Nonlinear(3,10,[0,3,1],2,1) %Example 4.4
%Nonlinear(2,10,[2,2],1,1) %Example 4.5