HW 5 Solution - ISE 754 Fall 2020

Contents

Question 1

Create Data

```
k = 50;
C = [0 92 50 56; 92  0 80 74; 50 80 0 18; 56 74 18 0];
mdisp(C)
```

```
C:   1   2   3   4
-:----------------
1:   0  92  50  56
2:  92   0  80  74
3:  50  80   0  18
```

```
4:  56  74  18   0
```

(b) Solve using UFLADD heuristic

```
[y,TC,X] = ufladd(k,C);
y,TC,mdisp(X)
```

```
y =

     2     3


TC =

   168


X:  1  2  3  4
-:-----------
1:  0  0  0  0
2:  0  1  0  0
3:  1  0  1  1
4:  0  0  0  0
```

(c) Formulate as MILP and solve

```
clear mp
mp = Milp('UFL')
[n m] = size(C);
kn = iff(isscalar(k),repmat(k,1,n),k(:)');
mp.addobj('min',kn,C)
for j = 1:m
   mp.addcstr(0,{':',j},'=',1)
end
for i = 1:n
   mp.addcstr({m,{i}},'>=',{i,':'})
end
mp.addub(1,1)
mp.addctype('B','C')
mp.dispmodel
ilp = mp.milp2ilp;
[x,TC,exitflag,output] = intlinprog(ilp{:});
x = mp.namesolution(x);
y = find(x.kn), TC, mdisp(x.C)
```

```
mp =

  Milp with properties:

    Model: [1×1 struct]
```

UFL:	lhs	B	B	B	B	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	rhs
Min:		50	50	50	50	0.00	92	50	56	92	0.00	80	74	50	80	0.00	18	56	74	18	0.00	
1:	1	0	0	0	0	1.00	1	1	1	0	0.00	0	0	0	0	0.00	0	0	0	0	0.00	1
2:	1	0	0	0	0	0.00	0	0	0	1	1.00	1	1	0	0	0.00	0	0	0	0	0.00	1
3:	1	0	0	0	0	0.00	0	0	0	0	0.00	0	0	1	1	1.00	1	0	0	0	0.00	1
4:	1	0	0	0	0	0.00	0	0	0	0	0.00	0	0	0	0	0.00	0	1	1	1	1.00	1
5:	0	4	0	0	0	-1.00	0	0	0	-1	0.00	0	0	-1	0	0.00	0	-1	0	0	0.00	Inf
6:	0	0	4	0	0	0.00	-1	0	0	0	-1.00	0	0	0	-1	0.00	0	0	-1	0	0.00	Inf
7:	0	0	0	4	0	0.00	0	-1	0	0	0.00	-1	0	0	0	-1.00	0	0	0	-1	0.00	Inf
8:	0	0	0	0	4	0.00	0	0	-1	0	0.00	0	-1	0	0	0.00	-1	0	0	0	-1.00	Inf
lb:		0	0	0	0	0.00	0	0	0	0	0.00	0	0	0	0	0.00	0	0	0	0	0.00	
ub:		1	1	1	1	1.00	1	1	1	1	1.00	1	1	1	1	1.00	1	1	1	1	1.00	

```
LP:                Optimal objective value is 50.000000.


Heuristics:        Found 1 solution using rounding.

                   Upper bound is 200.000000.

                   Relative gap is 15.92%.


Cut Generation:    Applied 6 implication cuts.

                   Lower bound is 168.000000.

                   Relative gap is 0.00%.



Optimal solution found.

Intlinprog stopped at the root node because the objective value is within a gap
tolerance of the optimal value, options.AbsoluteGapTolerance = 0 (the default
value). The intcon variables are integer within tolerance,
options.IntegerTolerance = 1e-05 (the default value).


y =

     1     2     4


TC =

  168.0000

```

:	1	2	3	4
1:	1	0	0	0
2:	0	1	0	0
3:	0	0	0	0
4:	0	0	1	1

Question 2

Read Data

```
fn = 'HW5data.xlsx';
inC = table2struct(readtable(fn,'Sheet','Customers'));
inP = table2struct(readtable(fn,'Sheet','Plants'));
```

Geolocate

```
city2lonlat = @(city,st) ...
   uscity('XY',mand(city,uscity('Name'),st,uscity('ST')));
for i = 1:length(inP)
   XYP(i,:) = city2lonlat(inP(i).City,inP(i).State);
end
XYC = uszip5('XY',mand([inC.Zip],uszip5('Code5')));
```

Calc (Plant + Customer) to Customer Transport Costs

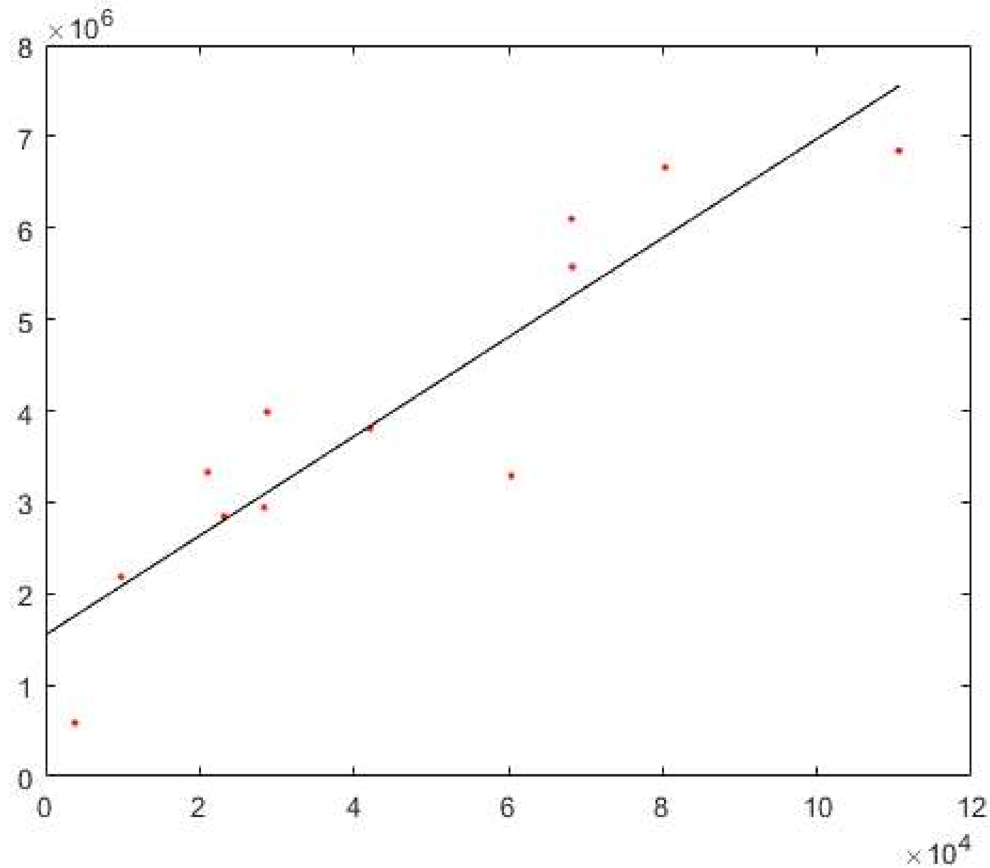
```
length([inC.Zip]) == length(unique([inC.Zip]))  % all customers in diff Zip
D = dists(XYP,XYC,'mi');                        % => no area adj needed
f = [inC.Demand];
F = sparse(argmin(D,1),1:length(inC),f);  % allocate customers to plants
r = sum([inP.DistCost])/sum(sum(F.*D))     % nominal network-wide $/ton-mi
D = dists([XYP; XYC],XYC,'mi');            % can ignore circuity
C = r*(f(:)'.*D);
```

```
ans =

  logical

   1


r =

    0.1812
```

Est Fixed Cost

```
x = sum(F,2);
y = [inP.ProdCost]';
yest = @(x,p) p(1) + p(2)*x;
fh = @(p) sum((y - yest(x,p)).^2);
ab = fminsearch(fh,[0 1])
k = ab(1), c_prod = ab(2)
plot(x,y,'r.')
hold on, fplot(@(x) yest(x,ab),[0 max(x)],'k-'), hold off
```

```
ab =

   1.0e+06 *

    1.5473    0.0001


k =

   1.5473e+06


c_prod =

   54.2893
```



Current TLC

```
yorig = 1:length(inP)
nNForig = length(yorig)
distCost_orig = sum([inP.DistCost])
fixedCost_orig = k * length(inP)
TLCorig = fixedCost_orig + distCost_orig
```

```
yorig =
```

```
     1     2     3     4     5     6     7     8     9    10    11    12


nNForig =

    12


distCost_orig =

    15474204


fixedCost_orig =

   1.8568e+07


TLCorig =

   3.4042e+07
```

New TLC

```
[ynew,TLCnew,X] = ufl(k,C); ynew, TLCnew
nNFnew = length(ynew)
```

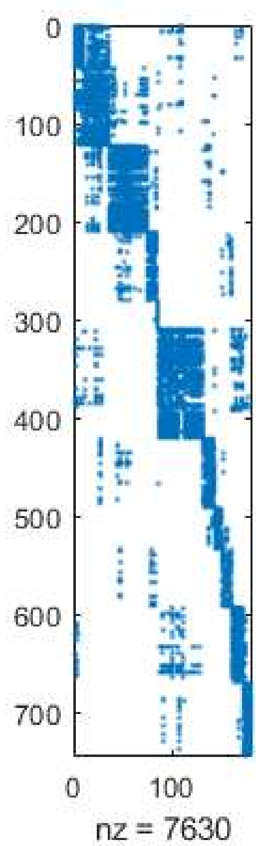
```
  Add: 30364130.242262
 Xchg: 30161453.077928
  Add: 30161453.077928
 Drop: 30161453.077928
Final: 30161453.077928

ynew =

    11    41    93   108   130   132   190   236


TLCnew =

   3.0161e+07


nNFnew =

     8
```

Question 3

Create set covering model

```
clear all, close all
s= uszip5(strcmp('NC',uszip5('ST')) & uszip5('Pop') >20000);
d = uszip5(strcmp('NC',uszip5('ST')) & uszip5('Pop') >0);
D = dists(d.XY,s.XY,'mi');
D = D + sqrt(d.LandArea/pi); % Add center to edge distance of demand region
rmax = 30;
c = ones(1,size(D,2));
A = false(size(D));
A(D < rmax) = true;
is0 = ~any(A,2);
A(is0,:) = [];
fprintf('Total pop %d; pop not covered %d; pct covered %f%%\n',...
   sum(d.Pop),sum(d.Pop(is0)),...
   100*(sum(d.Pop) - sum(d.Pop(is0)))/sum(d.Pop))
mp = Milp('Set Cover');
mp.addobj('min',c)
mp.addcstr(A,'>=',1)
mp.addctype('B')
spy(A)
```

```
Total pop 9535477; pop not covered 212117; pct covered 97.775497%
```



Use INTLINPROG

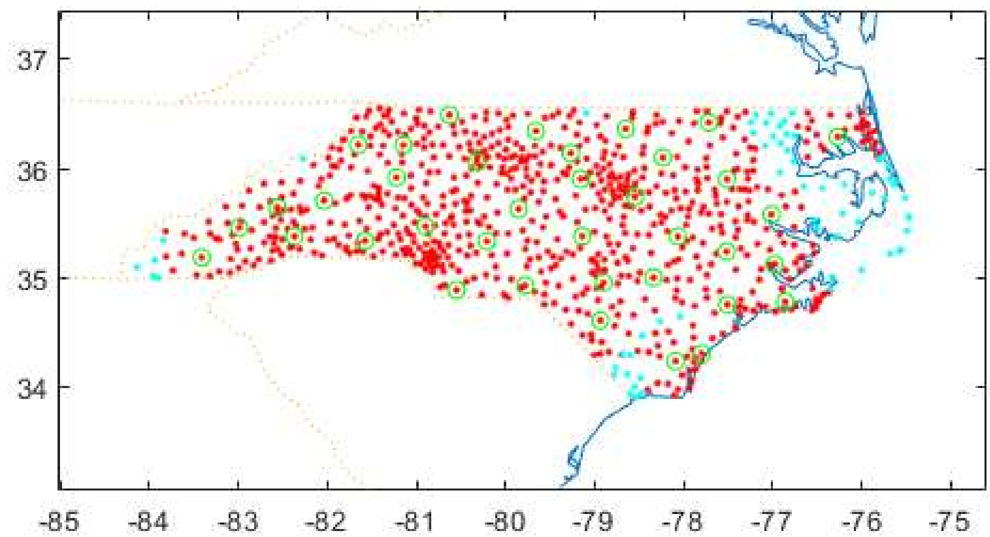
```
ilp = mp.milp2ilp
x = intlinprog(ilp{:});
```

```
nNF = sum(x)
```

```
ilp =

  1×8 cell array

  Columns 1 through 4

    {180×1 double}    {1×180 double}    {744×180 double}    {744×1 double}

  Columns 5 through 8

    {0×180 double}    {0×1 double}    {180×1 double}    {180×1 double}

LP:                Optimal objective value is 37.000000.



Optimal solution found.

Intlinprog stopped at the root node because the objective value is within a gap
tolerance of the optimal value, options.AbsoluteGapTolerance = 0 (the default
value). The intcon variables are integer within tolerance,
options.IntegerTolerance = 1e-05 (the default value).


nNF =

    37
```

Plot solution

```
idx = find(x);
makemap(d.XY)
pplot(d.XY(~is0,:),'r.')
pplot(d.XY(is0,:),'c.')
pplot(s.XY(idx,:),'go')
% pplot(s.XY(idx,:),s.Name(idx))
```



Question 4

EXAMPLE 4: UFL with n,m = 104

```
clear all
x = uszip5(mor({'NC'},uszip5('ST')) & uszip5('Pop') > 30000);
P = x.XY;
a = x.LandArea;
dafh = @(XY1,a1,XY2,a2) max(1.2*dists(XY1,XY2,'mi'),...
   0.675*max(sqrt(a1(:)),sqrt(a2(:)')));
Da = dafh(P,a,P,a);  % Area-adjusted distances
f = x.Pop';                 % person
r = 1/10000;                % $/person-mi
C = r*Da.*f;                % $
k = 500;  % repmat(500,1,size(C,1));
```

Demand and Capacity

Population for each EF already specified as 'f'

```
K = 4e5  % Maximum total population at NF
```

```
K =

      400000
```

Create MILP model of CFL

```
clear mp
mp = Milp('CFL')
mp.Model;
[n m] = size(C)
kn = iff(isscalar(k),repmat(k,1,n),k(:)');  % expand if k is constant value
mp.addobj('min',kn,C)  % min sum_i(ki*yi) + sum_i(sum_j(cij*xij))
for j = 1:m
   mp.addcstr(0,{':',j},'=',1)   % sum_i(xij) = 1
end
for i = 1:n
   mp.addcstr({K,{i}},'>=',{f,{i,':'}})  % m*yi >= sum_j(xij)  (weak form.)
end
mp.addub(1,1)
mp.addctype('B','C')         % only k are integer (binary)
mp.Model
```

```
mp = 

  Milp with properties:

    Model: [1×1 struct]


n =

   104


m =

   104


ans = 

  struct with fields:

     name: 'CFL'
    sense: 'minimize'
      obj: [1×10920 double]
       lb: [1×10920 double]
       ub: [1×10920 double]
    ctype: 'BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC…'
        A: [208×10920 double]
      lhs: [208×1 double]
      rhs: [208×1 double]
```

Solve using Gurobi

```
clear model params
model = mp.milp2gb;
params.outputflag = 1;
result = gurobi(model,params);
x = result.x;
x = mp.namesolution(x);
```

```
Academic license - for non-commercial use only
Gurobi Optimizer version 9.0.3 build v9.0.3rc0 (win64)
Optimize a model with 208 rows, 10920 columns and 21736 nonzeros
Model fingerprint: 0xc9f1ae36
Variable types: 10816 continuous, 104 integer (104 binary)
Coefficient statistics:
  Matrix range     [1e+00, 4e+05]
  Objective range  [7e+00, 2e+03]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 80402.136568
Presolve time: 0.03s
Presolved: 208 rows, 10920 columns, 21736 nonzeros
Variable types: 10816 continuous, 104 integer (104 binary)

Root relaxation: objective 7.546385e+03, 214 iterations, 0.02 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 7546.38534    0  101 80402.1366 7546.38534  90.6%     -    0s
H    0     0                    52712.826590 7546.38534  85.7%     -    0s
H    0     0                    47710.831214 7546.38534  84.2%     -    0s
H    0     0                    40687.377837 7546.38534  81.5%     -    0s
H    0     0                    25722.585854 9520.93659  63.0%     -    0s
     0     0 9520.93659    0   73 25722.5859 9520.93659  63.0%     -    0s
     0     0 9522.07380    0   73 25722.5859 9522.07380  63.0%     -    0s
H    0     0                    21295.407336 9522.07380  55.3%     -    0s
     0     0 10929.3146    0   64 21295.4073 10929.3146  48.7%     -    0s
H    0     0                    19378.635796 10929.3146  43.6%     -    0s
     0     0 10940.4299    0   64 19378.6358 10940.4299  43.5%     -    0s
     0     0 10942.5319    0   64 19378.6358 10942.5319  43.5%     -    0s
     0     0 11776.3457    0   64 19378.6358 11776.3457  39.2%     -    0s
H    0     0                    17245.251368 11776.3457  31.7%     -    0s
     0     0 11778.1252    0   64 17245.2514 11778.1252  31.7%     -    0s
     0     0 12486.3831    0   51 17245.2514 12486.3831  27.6%     -    0s
H    0     0                    15927.307621 12486.3831  21.6%     -    0s
H    0     0                    15749.922137 12486.3831  20.7%     -    0s
H    0     0                    15323.033031 12486.3831  18.5%     -    0s
H    0     0                    14837.799396 12486.3831  15.8%     -    0s
H    0     0                    14367.729128 12486.3831  13.1%     -    0s
     0     0 12503.5746    0   52 14367.7291 12503.5746  13.0%     -    0s
     0     0 12503.7465    0   51 14367.7291 12503.7465  13.0%     -    0s
     0     0 12700.3775    0   42 14367.7291 12700.3775  11.6%     -    0s
     0     0 12712.7315    0   41 14367.7291 12712.7315  11.5%     -    0s
     0     0 12714.2863    0   40 14367.7291 12714.2863  11.5%     -    0s
     0     0 12762.1705    0   37 14367.7291 12762.1705  11.2%     -    0s
H    0     0                    13807.711769 12762.1705  7.57%     -    0s
     0     0 12784.6572    0   26 13807.7118 12784.6572  7.41%     -    0s
     0     0 12791.1734    0   27 13807.7118 12791.1734  7.36%     -    0s
     0     0 12817.3977    0   40 13807.7118 12817.3977  7.17%     -    0s
     0     0 12817.3977    0   40 13807.7118 12817.3977  7.17%     -    0s
H    0     0                    13144.500748 12817.3977  2.49%     -    0s
     0     2 12817.3977    0   40 13144.5007 12817.3977  2.49%     -    0s
H   30    32                    12899.541723 12829.9544  0.54%  24.6    0s
H  214    70                    12897.602443 12829.9544  0.52%  17.4    0s
H  252    97                    12891.729098 12840.2536  0.40%  18.3    0s
```

```
H  257    97                    12880.443548 12840.2536  0.31%  18.2    0s

Cutting planes:
  Implied bound: 491
  Flow cover: 40
  Relax-and-lift: 50

Explored 490 nodes (8416 simplex iterations) in 1.01 seconds
Thread count was 6 (of 6 available processors)

Solution count 10: 12880.4 12891.7 12897.6 ... 15749.9

Optimal solution found (tolerance 1.00e-04)
Best objective 1.288044354803e+04, best bound 1.288044354803e+04, gap 0.0000%
```

CFL solution

```
TCcfl = result.objval
idxNFcfl = find(round(x.kn))  % Round in case y > 0 & y < eps
nNFcfl = sum(x.kn)
pop = round(sum(f.*x.C,2));
mdisp(pop(idxNFcfl),idxNFcfl,[],'Site')
```

```
TCcfl =

   1.2880e+04


idxNFcfl =

     6    20    34    42    47    50    71    76    78    82    91   100


nNFcfl =

    12


Site:     1
----:---------
   6:  400,000
  20:  400,000
  34:  400,000
  42:  362,695
  47:  378,386
  50:  256,657
  71:  400,000
  76:  400,000
  78:  400,000
  82:  337,098
  91:  226,842
 100:  305,169
```

(Fractional xij)

```
xijfrac = nnz(x.C > eps & x.C < 1-eps)
xijint = nnz(x.C > 1-eps)
xijfrac + xijint
m  % No. EF
```

```
xijfrac =

    12


xijint =

    98


ans =

   110


m =

   104
```

Compare CFL to UFL solution

Copy UFL solution from script results

```
TCufl = 1.2102e+04
idxNFufl = [6    18    39    50    66    82    91   100]
nNFufl = 8
vdisp('nNFcfl,nNFufl')
100*TCcfl/TCufl
```

```
TCufl =

       12102


idxNFufl =

     6    18    39    50    66    82    91   100


nNFufl =

     8
```

```
 :  nNFcfl  nNFufl
-:----------------
1:    12       8

ans =

  106.4324
```

Published with MATLAB® R2019b