18-879M: Optimization in Energy Networks Homework 4

Issued: Tuesday, February 18, 2020 Due: Tuesday, February 25, 2020, 11:59pm

Question 1. (100pts)

Implement the Dantzig-Wolfe decomposition algorithm in Matlab as described in the textbook. You may use linprog() to solve LP subproblems and master problem. You may also use decoupled and complicating definition matrices and vectors as the input to your function. Compare your implementation results with linprog() results. Document your code and explain in the write up in detail what you are doing to get full credit.

I tried to implement the Dantzig-Wolfe decomposition algorithm from both textbooks. For the method Dr. Luenberger has introduced, he used the revised simplex method to build the tableau and record the result by switching the variables from column and row. I just build a prototype program for Dr. Luenberger's method and used textbook example and my results are matched with the textbook. It is a good algorithm and I also make use of swapVars.m from last homework assignment.

For the method Dr. Conejo has introduced, it is my homework, so I made it general to use.

```
function [out] = DW_HW4(A,b,c,numLinks,borderRow,borderCol,k)
```

My function has 7 inputs and 1 output. Matrix A is the whole constrains matrix, which includes both linking constraints and inequality constraints. The matrix b is the right-hand side value of the inequality equations. The variable numLinks are the number of linking constraints. The matrix borderRow and borderCol separate matrix A in different parts of the subproblems and master problem. The variable k the number of the subproblem.

Now, I will use an example to explain how I separate my matrix by using those inputs.

$$C_T = X_1 - 4X_2 + 2X_3 - 7X_4 + 6X_5 - 5X_6 - 3X_7 + 2X_8$$

$$\underset{\text{A}}{\begin{vmatrix} 2 & 3 & 4 & 2 & 7 & 6 & 8 & 9 \\ 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 4 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \end{vmatrix}} \times \underset{}{\begin{vmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \\ X_8 \end{vmatrix}} = \underset{\text{b}}{\begin{vmatrix} 26 \\ 12 \\ 14 \\ 17 \\ 15 \\ 16 \\ 10 \\ 11 \end{vmatrix}}$$

borderRow:

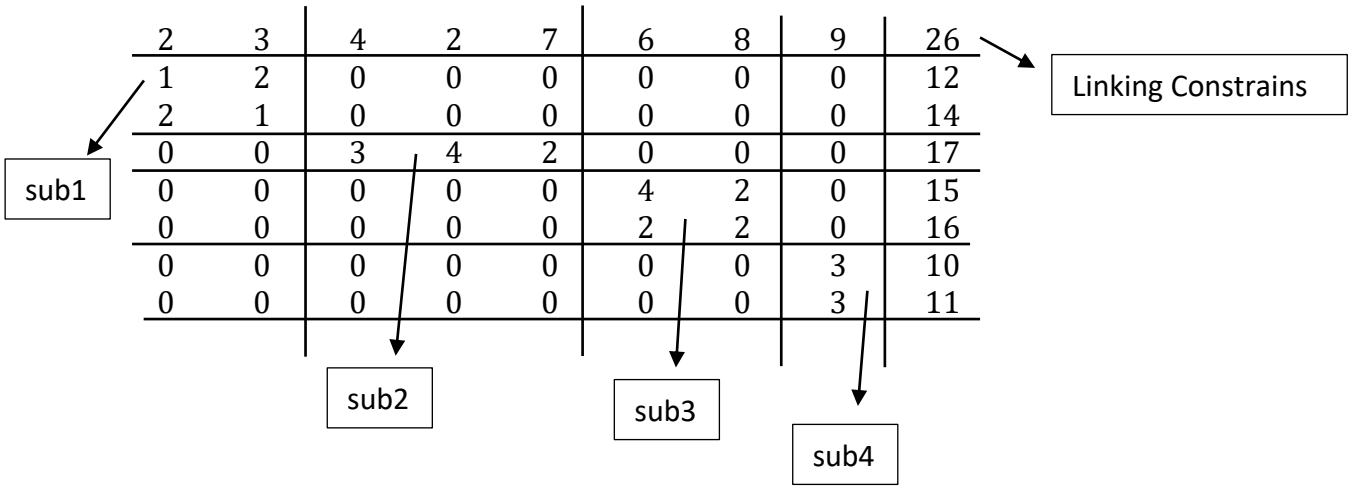
1	1	row 1 to row 1
2	3	row 2 to row 3
4	4	row 4 to row 4
5	6	row 5 to row 6
7	8	row 7 to row 8

borderCol

1	2	column 1 to column 2
3	5	column 3 to column 5
6	7	column 6 to column 7
8	8	column 8 to column 8

k = 4 there are totally four subproblems
numLinks = 1 there are totally one linking constrain

How borderRow and borderCol separate the matrix:



So, we can break one problem into four subproblems.

Programming Process

Step 0: Initialization.

First, I picked two initial extreme points randomly. We can set two object functions randomly. So, I can find two different points from these two different objective functions by using linprog with each subproblem's constraints.

```
c_random1=ones(numVarT,1); % use 1 for a cost object function
c_random2=zeros(numVarT,1)+(-1); % use -1 for a cost object function
```

Since we got those two initial points, I can use these two points to find r1 and r2, z1 and z2.

```
% find a feasible solution from random cost function
for i=2:1:k+1
    c1=c_random1(borderCol(i-1,1):borderCol(i-1,2),1);
    numVar=borderCol(:,2)-borderCol(:,1)+1;
    LB=zeros(numVar(i-1,1),1);
    X_init1=linprog(c1,A(borderRow(i,1):borderRow(i,2),...
        borderCol(i-1,1):borderCol(i-1,2)),...
        b(borderRow(i,1):borderRow(i,2),1),[],[],LB,[]);
    X1=[X1,X_init1.'];
end

% find another feasible solution from another random cost function
for i=2:1:k+1
    c1=c_random2(borderCol(i-1,1):borderCol(i-1,2),1);
    numVar=borderCol(:,2)-borderCol(:,1)+1;
    LB=zeros(numVar(i-1,1),1);
    X_init2=linprog(c1,A(borderRow(i,1):borderRow(i,2),...
        borderCol(i-1,1):borderCol(i-1,2)),...
        b(borderRow(i,1):borderRow(i,2),1),[],[],LB,[]);
    X2=[X2,X_init2.'];
```

This is how I found the two extreme points. Firstly, I set the cost function for each subproblem all 1 or all -1. And then, I use each subproblem constraints to find each solution of the subproblem. Last, I combine each solution into one extreme point.

```
X=[X1;X2]; % two initial feasible solution
% find the initial master problem
r1=A_linking*X1.'; z1 = X1*c;
r2=A_linking*X2.'; z2 = X2*c;
master_object=[];A_master=[];
master_object=[z1,z2];A_master=[r1,r2];
```

We can find r and z for each initial point and stored X1 and X2 in X.

Step 1: Master problem solution.

$$\underset{u_1,\ldots,u_{p^{(\nu)}}}{\text{minimize}} \quad \sum_{s=1}^{p^{(\nu)}} z^{(s)} u_s$$

$$\sum_{s=1}^{p^{(\nu)}} r_i^{(s)} u_s = b_i : \ \lambda_i; \qquad i = 1,\ldots,m$$

$$\sum_{s=1}^{p^{(\nu)}} u_s = 1 : \ \sigma$$

$$u_s \geq 0; \ s = 1,\ldots,p^{(\nu)}$$

We can get the master objective function and master function constrains by using the z1 and z2, r1 and r2 from step 0. We can also use the basic theorem of the convex:

Definition. A set C in E^n is said to be *convex* if for every $\mathbf{x}_1, \mathbf{x}_2 \in C$ and every real number α, $0 < \alpha < 1$, the point $\alpha\mathbf{x}_1 + (1 - \alpha)\mathbf{x}_2 \in C$.

We can express one feasible point in the by using multiple extreme points in convex and the coefficients add up to 1 as another constrains for the master problem.

```
%step 1 master problem solution
[master_row,master_col]=size(master_object);U=ones(1,master_col);
LB=zeros(master_col,1);
[u,FVAL,EXITFLAG,OUTPUT,LAMBDA] = linprog(master_object,A_master,b_master
lamda=-LAMBDA.ineqlin; sigma = -LAMBDA.eqlin;
```

We can use linprog to find dual variable values, which is lamda and sigma in this algorithm. We can also find the weight of each point, which is u_s in this algorithm.

Step 2: Relaxed problem solution.

Generate a solution of the relaxed problem by solving the r subproblems (k = 1, . . . , r) below.

$$\underset{x_{n_{k-1}+1}, x_{n_{k-1}+2}, \ldots, x_{n_k}}{\text{minimize}} \quad \sum_{j=n_{k-1}+1}^{n_k} \left(c_j - \sum_{i=1}^{m} \lambda_i^{(\nu)} a_{ij} \right) x_j$$

```
%step 2 relaxed problem solution
%objective funtion for of subproblems
v=[];
for i=1:1:k
    v_x=c(borderCol(i,1):borderCol(i,2),1).'-lamda*A_linking(1,...
        borderCol(i,1):borderCol(i,2));
    [row_v,col_v]=size(v_x);
    v_x(row_v,numVarT)=0;
    v=[v;v_x];
end
```

Next, I can find the objective function of the relaxed problem by using the lamda we obtained from step 1. I stored each individual function in a matrix.

```
% find the solution for each subproblem, and combine them for
% the feasible solution
X_sub=[];
for i=2:1:k+1
  numVar=borderCol(:,2)-borderCol(:,1)+1;
  LB=zeros(numVar(i-1,1),1);
  X_s = linprog(v(i-1,1:numVar(i-1,1)),A(borderRow(i,1):borderRow(i,2),...
        borderCol(i-1,1):borderCol(i-1,2)), b(borderRow(i,1):...
        borderRow(i,2),1),[],[],LB,[]);
  X_sub=[X_sub,X_s.'];
end
```

The constrains for each subproblem are the same as the original. We can use linprog for each subproblem to find a feasible point. After we find the point, we can find the objective function value of the original problem is z and the value of the complicating constraint r.

```
% find r1 and z for the new feasible solution
X=[X;X_sub]; r1=A_linking*X_sub.'; z = X_sub*c;
[row_vv,col_vv]=size(v); v_t=[];
for i=1:1:row_vv
     v_t=[v_t,v(i,1:numVar(i,1))];
end
```

We can plug the current solution into the current subproblem's objective function to get the value v.

Step 3: Convergence checking.

If the v is smaller than the sigma, this means we can use the current solution of the relaxed problem to improve the solution of the master problem. So, we can update our object function and master constraint by add new z and r calculated from the new solution.

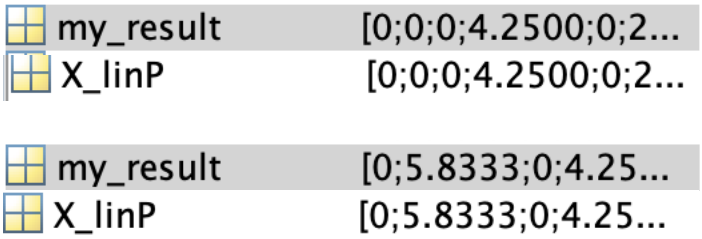
```
if v_t_val<sigma %if v< sigma
    master_object=[master_object,z]; % update the master function
    A_master=[A_master,r1]; % update the master constrain
    U = [U,1]; LB=[LB;0]; % add a new u and lower bound
    v=v+1; p=p+1;     % iteration+1, new constrains+1
```

If the v is bigger or equal than the sigma, we get the optimal solution.

```
    [row_u,col_u]=size(u);
    X_result=zeros(1,col_u);
    for i=1:1:row_u
        X_result=X_result+u(i,1)*X(i,:); % get the result
    end
    out=X_result.'; % output the result
    return
```

So, my final result is each solution multiplies by its weight.

Test



My result matches the result from linprog.

Reference

David G. Luenberger, *"Linear and Nonlinear Programming"*, Addison-Wesley, 1984/2003

A.J. Conejo, E. Castillo, R. Minguez, R. Garcia- Bertrand, "*Decomposition Techniques in Mathematical Programming",* Springer, 2006