Yuchen Shi
6962247146
Math 467
Dec 3rd, 2018

Project #2 Linear Programming

Part 1—Implementation of the basic simplex algorithm

On the basis of the basic simplex Matlab function we were given, I changed the outputs of the function to include two more variables: one is A, which is the matrix corresponding to the equality constraints Ax=b after all the row operations that were done in the function. Basically, it is the A after we arrive at the optimal solution. The other variable I added is b, which is the vector corresponding to the right hand side of the equality constraints after all the row operations that were done in the function. So, it is the b after we arrive at the optimal solution. I also added two if statement to check whether or not the vector b on the input has nonnegative entries and the columns corresponding to the BasicVar0 is an identity matrix before beginning the execution of the simplex algorithm. Then, I implemented it on two test cases from the exercise problems of the book:
The first problem is:

$$\begin{array}{rl} \text{maximize} & 2x_1 + 5x_2 \\ \text{subject to} & x_1 \le 4 \\ & x_2 \le 6 \\ & x_1 + x_2 \le 8 \\ & x_1, x_2 \ge 0. \end{array}$$

I first transform the problem into standard form by introducing three slack variables x3, x4, x5.

$$\begin{array}{rcccccll} \text{minimize} & -2x_1 & -5x_2 & -0x_3 & -0x_4 & -0x_5 & & \\ \text{subject to} & x_1 & & +x_3 & & & = & 4 \\ & & x_2 & & +x_4 & & = & 6 \\ & x_1 & +x_2 & & & +x_5 & = & 8 \\ & x_1, & x_2, & x_3, & x_4, & x_5 & \ge & 0. \end{array}$$

Indices of variables in the initial basic solution would be 3,4,5. Thus, BasicVar0 = [3,4,5]. I applied the my basic simplex Matlab code and got the output variable Status = 0, which means I find the optimal solution. The Solution variable = [2,6,2,0,0], which is consistent with the answer of the exercise problem.

The second problem is:

$$\begin{aligned} \text{maximize} \quad & 7x_1 + 6x_2 \\ \text{subject to} \quad & 2x_1 + x_2 \leq 3 \\ & x_1 + 4x_2 \leq 4 \\ & x_1, x_2 \geq 0. \end{aligned}$$

Again, I transform the problem into standard form, so the basic simplex method can be applied. I added two slack variables x3 and x4. Now the tableau looks like:

$$\begin{array}{cccccc} & \boldsymbol{a}_1 & \boldsymbol{a}_2 & \boldsymbol{a}_3 & \boldsymbol{a}_4 & \boldsymbol{b} \\ & 2 & 1 & 1 & 0 & 3 \\ & 1 & 4 & 0 & 1 & 4 \\ \boldsymbol{c}^\top & -7 & -6 & 0 & 0 & 0 \end{array}$$

This time, indices of variables in the initial basic solution is 3,4. BasicVar0 = [3,4]. I ran my basic simplex Matlab code and got the output variable Status = 0, which means I find the optimal solution again. The Solution variable = [8/7,5/7,0,0], which is the same as the answer of this exercise problem.

Part 2—Implementation of general linear programming algorithm

The basic simplex method requires an initial basic feasible solution, and we need to identify the initial basic variables to initiate the basic simplex algorithm. In general, an initial basic feasible solution is not always apparent. Therefore, we need a systematic way to find the initial basic feasible solution for general linear programming problems so that we could then call the basic simplex method. I chose the two-phase simplex method: If there are m equality constraints originally, I added an identity matrix of size m to the right of the original matrix A and introduced m artificial variables [y1…ym]. In phase one, the problem becomes

$$\begin{aligned} \text{minimize} \quad & y_1 + y_2 + \cdots + y_m \\ \text{subject to} \quad & [\boldsymbol{A}, \boldsymbol{I}_m] \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} = \boldsymbol{b} \\ & \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} \geq \boldsymbol{0}, \end{aligned}$$

Then, I could apply the basic simplex method to solve this artificial problem. If the solution to the artificial problem has all yi = 0 and the objective function = 0. The basic variables are in the first n components (n is the number of variables in the original problem). As a result, we find an initial basic feasible solution for the original linear programming problem. If the objective function does not equal to 0, then the original problem is not feasible. In phase two, I just use the basic feasible solution resulting from phase 1 and delete the components corresponding to artificial variables. I also truncate A the part corresponding to artificial variables. After that, I change the cost function back to the original costs and use the basic simplex method again to solve the original problem.

In the code that I wrote, I first merge the three matrixes given in the input vertically, and also merge the right hand side vertically. Since the second and third matrix correspond to Ax<0 and Ax>0, I added two sets of slack variables to turn this problem into standard format. I also check whether the input b1, b2, or b3 has negative entries. If they have, I multiply -1 on rows of matrix A and vector b where b's entry is negative. So, the b I have now would only have nonnegative entries while the original problem is not changed. Then I apply the two-phase simplex method as described above.

To check my code work or not, I test it on two test cases from the exercise problems of the book.
The first problem is:

$$\begin{aligned} \text{minimize} \quad & 2x_1 + 3x_2 \\ \text{subject to} \quad & 4x_1 + 2x_2 \geq 12 \\ & x_1 + 4x_2 \geq 6 \\ & x_1, x_2 \geq 0. \end{aligned}$$

Since my code turns problems into standard format automatically, I don't need to consider adding slack variables now and could just apply my general linear programming Matlab code. In this example, A1, A2, b1 and b2 would be empty. A3=[4,2;1,4], b3=[12;6], and c=[2,3]. The solution I get after running the code is Solution = [18/7,6/7,0,0], which is consistent with the solution of this problem.

The second problem is:

$$\begin{aligned} \text{maximize} \quad & 3x_1 + 5x_2 \\ \text{subject to} \quad & x_1 + x_2 \leq 4 \\ & 5x_1 + 3x_2 \geq 8 \\ & x_1, x_2 \geq 0. \end{aligned}$$

This time, I only need to change maximization into minimization, so the cost variable c=[-3,-5]. A1 and b1 are empty. A2=[1,1], A3=[5,3], b2=[4] and b3=[8]. The solution I get after running my code is Solution=[0,4,0,4], which is correct.

Part 3—Study of L1 versus L2 approximation

While studying the linear relationship between two variables, we could use a L1 regression or a L2 regression. They are the same in a sense that both are trying to find the slope a and intercept b that would minimize the difference between actual y value and fitted y value from the regression. The difference is that L1 regression is minimizing the following function:

$$g(a,b) = \sum_{k=1}^{N} |y_k - ax_k - b|$$

On the other hand, L2 regression is minimizing this function:

$$f(a,b) = \sum_{k=1}^{N} |y_k - ax_k - b|^2.$$

The L1 regression can be transformed into a linear programming problem:
Let Wk=Yk-aXk-b

$$\underset{w \in \mathbb{R}^N}{\text{minimize}} \sum_{k=1}^{N} |w_k|$$

Subject to Wk+aXk+b=Yk, k=1…N.
To deal with the absolute sign of Wk, we introduce two sets of variables Wk+ and Wk-. Both Wk+ and Wk- are larger than or equal to zero, |Wk| = Wk+ plus Wk-; while Wk = Wk+ minus Wk-. Now the problem is a standard linear programming problem.
I used my code from part 2, my input variables A2, A3, b2 and b3 are all empty. A1 is a N*(2N+4) matrix with the first 2N columns having 1 and -1 for the corresponding Wk+ and Wk- and 0 in other places. The last 4 columns are for the parameters a and b. Since a and b might be positive or negative, I also break them into a+, a-, b+ and b-. In our solution, if a+ = 0, then our parameter a = -a-; otherwise, a=a+. Similarly, if b+=0, then b = -b-; otherwise, b=b+. In the matrix A1, the 2N+1 and 2N+2 columns are all the Xk and -Xk respectively. The 2N+3 and 2N+4 columns are all 1 and -1 respectively. Our right hand side b1 would just be all the Yk values.

The L2 regression problem could be solved easily since the function we are minimizing is differentiable. We could just take derivative of f(a,b) with respect to a and b and set the two equations equal to zero. Solving this system of equations would give us the optimal values of a and b.
The result is:
a =

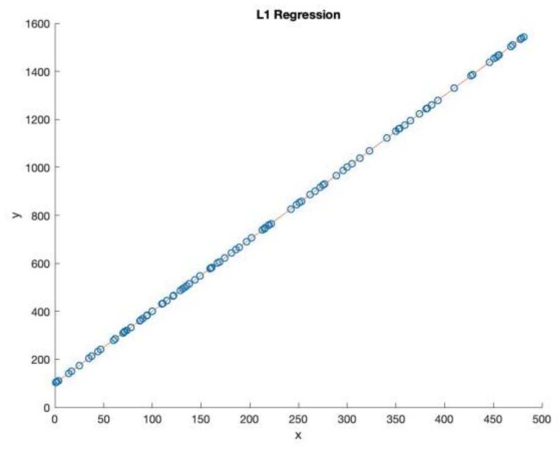
$$\frac{(N\Sigma xy - \Sigma x\, \Sigma y)}{N(\Sigma x^2) - (\Sigma x)^2}$$

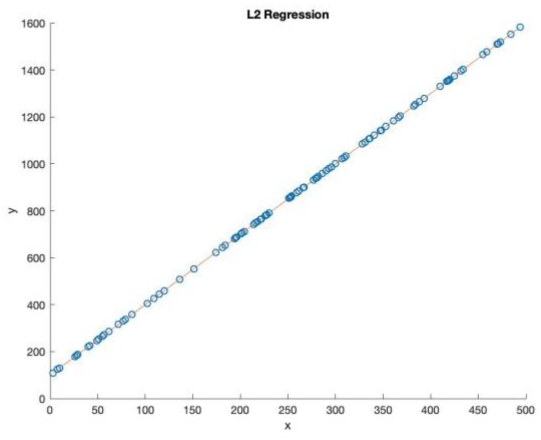
b = y_bar – a * x_bar

with y_bar and x_bar equal to the average of all Yk and Xk.

I generated two sets of data to test my code:
In the first set, I focus on data that have perfect linear relationship. I randomly generate 100 integers between 1 and 500 and save them as X. Then I calculate 3X+1000 for these data and save them as Y. Basically, Y=3X+100 would be my linear relationship. I ran my code using the algorithm described as above, the results for both L1 and L2 regression are correct. Both methods give me the correct value of a=3 and b=100.

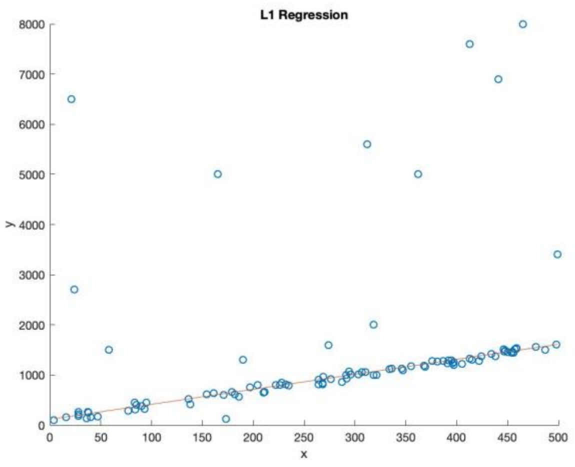
Here I plotted a scatterplot of X and Y and a line corresponding to the linear equation I got from parameters a and b.

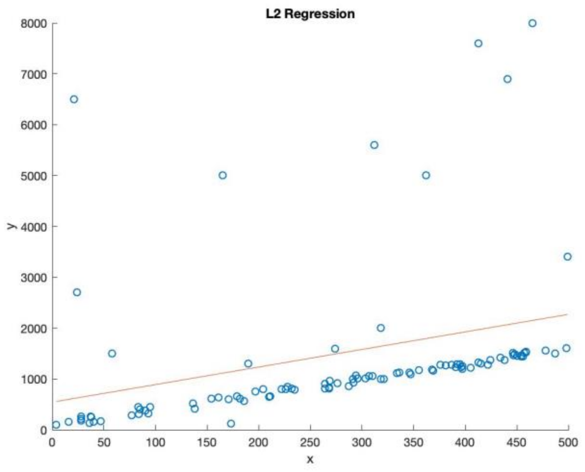




From these graphs, we could see that L1 and L2 regression are same in that they both give the correct value of the parameters a and b when the data has a perfect linear relationship.

Then I generate another set of data to include noises and outliers.
I added noise of random integers between -100 and 100 to all 100 values of Yk. And then I created some outliers by randomly changing 15 Yk values to be extremely large or extremely small. Now the data do not line in a straight line anymore because of the noises and outliers, so I apply my code again to see the difference between L1 and L2 regression.
The result of L1 regression is a=2.9920, b=115.1277. The result of L2 regression is a=3.4521, b=544.9590. The result of L2 regression deviates a lot from the original result with is 3X+100. While L1 regression still gives me values similar to the original result. Here is a scatterplot of X and Y and a line corresponding to the linear equation I got from parameters a and b.





This time, it is clear to see that L1 regression performs better than L2 regression. Because of the outliers, the least square regression line I got from L2 regression deviate from the main trend of the data a lot. On the contrary, the regression line I got from L1 regression still matches the main trend of the data in spite of outliers. Therefore, a big difference between L1 and L2 regression is that L1 regression is more resistant to outliers and thus more robust. L2 is more easily affected by outliers and could potentially give misleading results of parameters a and b. When dealing with data that have outliers, L1 regression will work better compared to L2 regression. The reason is that L2 regression squares the errors and will see much larger errors in outliers than L1 regression does, so L2 regression is more sensitive to outliers. Since both methods are trying to minimize errors, in the case of outliers, L2 regression adjust parameters a and b to a large extent to minimize the errors of the outliers, at the expense of many other common data. L1 regression didn't adjust too much because it sees smaller errors for outliers; it still focuses on minimizing the errors of the common data.

Appendix

Code for basic simplex algorithm:

```
function [A,b,Solution,BasicVar,Status]=basicsimplex(A,b,c,BasicVar0)
%
% Description: Basic simplex method for linear programming
% Usage: [Solution,BasicVar,Status]=basicsimplex(A,b,c,BasicVar0)
%   Inputs:
%       A          : Array of dimension m by n for the equality constraints
%                    Ax=b.
%       b          : Vector of dimension m for the right hand side of the
%                    equality constraints.
%       c          : The weights for the cost functional.
%       BasicVar0 : Indices of variables in the initial basic solution.
%   Outputs:
%       A          : Matrix A after we arrive at the optimal solution.
%       b          : Vector b after we arrive at the optimal solution.
%       Solution   : Optimal solution when exists.
%       BasicVar   : Indices of basis variables for the solution.
%       Status     : Status of the solution. Status = 0 if the solution is
%                    optimal. Status = -1 if no optimal solution exits.
%
[mConstr,ndim]=size(A);
Solution=[];
BasicVar=[];
Status=-1;
if numel(b)~= mConstr
    disp('basicsimplex: Sizes of matrix A and vector b are inconsistent.');
    return;
end
if numel(BasicVar0)~= mConstr
    disp('basicsimplex: The number of basic variable must be equal to the
number of constraints.');
    return;
end
if numel(unique(BasicVar0))~= mConstr
    disp('basicsimplex: Indices of basic variables must not repeat.');
    return;
end
if numel(c)  ~= ndim
    disp('basicsimplex: Dimensions of matrix A and vector c are
inconsistent.');
    return;
end
if min(b)<0
    disp('basicsimplex: b has nonnegative entries.');
    return;
end
```

```matlab
b=reshape(b,mConstr,1);
c=reshape(c,ndim,1);
%
% Initialize the simplex table
%
A_basic=A(:,BasicVar0);
tf=isequal(A_basic,eye(mConstr));
if tf == 0
    disp('basicsimplex: Basic columns are not the identity matrix.');
    return;
end
A=inv(A_basic)*A;
b=inv(A_basic)*b;
if min(b)<0
    disp('basicsimplex: Components of the initial basic solution must be 
all non negative.');
    Solution=b;
    BasicVar=BasicVar0;
    Status=-1;
    return;
end
%
%  Basic simplex method
%
Opt_Flag=-1;
NonBasicVar=[1:ndim];
NonBasicVar(BasicVar0)=-1;
NonBasicVar=find(NonBasicVar>0);
ReduceCost=zeros(ndim,1);
BasicVar=BasicVar0;
while Opt_Flag == -1
    %
    % Compute the reduced cost coefficients.
    %
    ReduceCost(BasicVar)=inf;
    ReduceCost(NonBasicVar)=c(NonBasicVar)-A(:,NonBasicVar)'*c(BasicVar);
    if min(ReduceCost)>=0
        Opt_Flag=1;
        Status=0;
        Solution=zeros(ndim,1);
        Solution(BasicVar)=b;
        return;
    end
    %
    % Select non-basic variable to enter basis.
    %
    [v,indCandidate]=min(ReduceCost);
    Slack=inf(mConstr,1);
    ind=find(A(:,indCandidate)>0);
    if isempty(ind)
        Opt_Flag=1;
        Status=-1;
        Solution=zeros(ndim,1);
        Solution(BasicVar)=b;
        return;
    end
    Slack(ind)=b(ind)./A(ind,indCandidate);
    [v,indOut]=min(Slack);
    b(indOut)=b(indOut)/A(indOut,indCandidate);
    A(indOut,:)=A(indOut,:)/A(indOut,indCandidate);
    indRest=find([1:mConstr]~=indOut);
    b(indRest)=b(indRest)-A(indRest,indCandidate)*b(indOut);
    A(indRest,:)=A(indRest,:)-A(indRest,indCandidate)*A(indOut,:);
    NonBasicVar(NonBasicVar==indCandidate)=BasicVar(indOut);
    BasicVar(indOut)=indCandidate;
end
return
end
```

Code for general linear programming algorithm:

```
function [Solution,Status]=generalLinearProgramming(A1,A2,A3,b1,b2,b3,c)

A=[A1;A2;A3];
sz=size(A);
column=sz(2);
size1=size(A1);
size2=size(A2);
size3=size(A3);
row1=size1(1);
row2=size2(1);
row3=size3(1);

A(row1+1:row1+row2,column+1:column+row2)=eye(row2);
A(row1+row2+1:row1+row2+row3,column+row2+1:column+row2+row3)=-eye(row3);

b1=reshape(b1,row1,1);
b2=reshape(b2,row2,1);
b3=reshape(b3,row3,1);
b0=[b1;b2;b3];

index=find(b0<0);
A(index,:)=-A(index,:);
b0(index)=-b0(index);

A(1:row1+row2+row3,column+row2+row3+1:column+row2+row3+row1+row2+row3)=eye(
row1+row2+row3);

c1=zeros(1,column+row2+row3);
c2=ones(1,row1+row2+row3);
c0=[c1 c2];

BasicVar0=[column+row2+row3+1:column+row2+row3+row1+row2+row3];

[A,b,Solution,BasicVar,Status]=basicsimplex(A,b0,c0,BasicVar0);
if Status == -1
    disp("Problem is not solvable.");
    return
end
if c0*Solution ~= 0
    disp("Problem is not feasible.");
    Status=-1;
    return
end

A=A(:,1:column+row2+row3);

zero=zeros(1,row2+row3);
```

```
c=[c zero];

[A,b,Solution,BasicVar,Status]=basicsimplex(A,b,c,BasicVar);
if Status == -1
    disp("Problem is not solvable.");
    return
end

return
end
```

Code for test cases:

```
% Basic simplex method test case 1
A=[1,0,1,0,0;0,1,0,1,0;1,1,0,0,1];
b=[4;6;8];
c=[-2,-5,0,0,0];
BasicVar0=[3,4,5];
[A,b,Solution,BasicVar,Status]=basicsimplex(A,b,c,BasicVar0);

% Basic simplex method test case 2
A=[2,1,1,0;1,4,0,1];
b=[3;4];
c=[-7,-6,0,0];
BasicVar0=[3,4];
[A,b,Solution,BasicVar,Status]=basicsimplex(A,b,c,BasicVar0);

% General LP problem test case 1
A1=[];
A2=[];
A3=[4,2;1,4];
c=[2,3];
b1=[];
b2=[];
b3=[12;6];
[Solution,Status]=generalLinearProgramming(A1,A2,A3,b1,b2,b3,c);

% General LP problem test case 2
A1=[];
A2=[1,1];
A3=[5,3];
c=[-3,-5];
b1=[];
b2=[4];
b3=[8];
[Solution,Status]=generalLinearProgramming(A1,A2,A3,b1,b2,b3,c);
```

Code for comparison of L1 and L2 regression:

```
x=randi(500,100,1);
y=3*x+100;
noise=randi([-100 100],100,1);
y=y+noise;
y(3)=1500;
y(5)=7600;
y(12)=5000;
y(13)=5600;
y(15)=1300;
y(34)=2700;
```

```
y(41)=6500;
y(52)=250;
y(62)=2000;
y(70)=6900;
y(73)=8000;
y(89)=1600;
y(95)=120;
y(96)=3400;
y(100)=5000;

sz=size(x);
k=sz(1);
W=zeros(k,2*k);
for i = 1:k
    W(i,2*i-1)=1;
    W(i,2*i)=-1;
end

X=reshape(x,k,1);
B=ones(k,1);
A=[W X -X B -B];
b0=reshape(y,k,1);
c=ones(1,2*k);
c0=[0,0,0,0];
c=[c c0];

A1=A;
A2=[];
A3=[];
b1=b0;
b2=[];
b3=[];

[Solution,Status]=generalLinearProgramming(A1,A2,A3,b1,b2,b3,c);
if Solution(2*k+1)==0
    a=-Solution(2*k+2);
else
    a=Solution(2*k+1);
end
if Solution(2*k+3)==0
    b=-Solution(2*k+4);
else
    b=Solution(2*k+3);
end

scatter(x,y);
hold on;
plot(x,a*x+b);
xlabel('x');
ylabel('y');
title('L1 Regression');

hold off;
sz=size(x);
n=sz(1);
x_bar=mean(x);
y_bar=mean(y);
xy=x.*y;
x_square=x.^2;
```

```
table=[x y xy x_square];

a2=(sum(xy)-n*x_bar*y_bar)/(sum(x_square)-n*x_bar^2);
b2=y_bar-a*x_bar;

scatter(x,y);
hold on;
plot(x,a2*x+b2);
xlabel('x');
ylabel('y');
title('L2 Regression')
```