Jose Canela
MATH467
12/3/18

# PROJECT 2

## INTRODUCTION:

This project consisted of two procedures:

- Implementing the simplex method to solve several general linear programming (GLP) problems
- Studying the similarities and differences between $L^2$ and $L^1$approximation.

## PROBLEM 1 - IMPLEMENTATION OF THE BASIC SIMPLEX ALGORITHM:

Problem 1 consisted of implementing the simplex method we learned from chapter 16 in the textbook. In particular, we were asked to implement the `basicsimplex` function (refer to **APPENDIX 1.1**) to solve the LP

$$\begin{gathered} \underset{x \in R^n}{Minimize}\ c^T x \\ subject\ to\ Ax = b \in R^m \\ x \geq 0 \end{gathered}$$

However, in order to perform the call this function, one needed to know the initial indices for the basic solution. This vector of indices was called `BasicVar0`. My function that implemented the `basicsimplex` function was called `simplex_imp` (Refer to **APPENDIX 1.2** for my `simplex_imp` code).

For my `simplex_imp` code, I solved the textbook problem:

$$\begin{gathered} \underset{x \in R^n}{Minimize}\ -2x_1 - 5x_2 \\ subject\ to\ x_1 \leq 4 \\ x_2 \leq 6 \\ x_1 + x_2 \geq 8 \\ x_1, x_2 \geq 0 \end{gathered}$$

The optimal solution to this problem should be x = [2,6,2,0,0]$^T$. In this case, after plugging in

$$\begin{aligned} A &= \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix} \\ b &= [4 \quad 6 \quad 8]^T \\ c &= [-2 \quad -5 \quad 0 \quad 0 \quad 0]^T \end{aligned}$$

into the `simplex_imp` function, one gets

Solution =

2
6
2
0
0

BasicVar =

1 2 3

which is the exact optimal solution. The code works!

**PROBLEM 2 - IMPLEMENTATION OF THE GENERAL LP ALG USING THE SIMPLEX METHOD:**

Problem 2 consisted of implementing creating a function (I called it `GEN_simplex_imp`) that solved the GLP

$$\begin{aligned} \underset{x \in R^n}{Minimize}\ c^T x& \\ subject\ to\ Ax\ &=\ b \in R^m \\ \hat{A}x\ &\leq\ \hat{b} \in R^{\hat{m}} \\ \tilde{A}x\ &\geq\ \tilde{b} \in R^{\tilde{m}} \\ x\ &\geq\ 0 \end{aligned}$$

Where the vectors $b$, $\hat{b}$, $\tilde{b}$ are assumed to have nonnegative components.

using the `simplex_imp` function. To do this I needed to turn this GLP problem into an LP standard vector form that looks similar to the LP in Problem 1. In order to do this, I created:

- A transformed matrix $A_{trans}$ consisting of the matrices $I$, $-I$, $A$, $\hat{A}$ and $\tilde{A}$

$$A_{trans} = \begin{bmatrix} [A] & [\mathbf{0}] & [\mathbf{0}] \\ [\hat{A}] & [I] & [\mathbf{0}] \\ [\tilde{A}] & [\mathbf{0}] & [-I] \end{bmatrix} \in\ R^{(m+\hat{m}+\tilde{m})*(n+\hat{m}+\tilde{m})}$$

- A transformed **b** vector, $b_{trans}$, consisting of $b$, $\widehat{b}$, and $\tilde{b}$

$$b_{trans}\ =\ \left[b, \hat{b}, \tilde{b}\right]^T\ \in R^{n+\hat{m}+\tilde{m}}$$

- A transformed coefficient vector called $c_{trans}$.
- $c_{trans}\ =\ [c, \mathbf{0}]\ \in R^{n+\hat{m}+\tilde{m}}$

WHERE

$$x_{trans} = [x_1, \dots, x_n, \dots, x_{n+\hat{m}+\tilde{m}}]^T\ \in R^{n+\hat{m}+\tilde{m}}$$

AND $x^*_{trans}$=$[x_1, \dots, x_n]$.

For my `GEN_simplex_imp` code (refer to **APPENDIX 2**), I solved the textbook problem:

$$\underset{x \in R^n}{Minimize}\ -3x_1 - 5x_2$$

$$\begin{aligned} subject\ to\ x_1 + 3x_2 &= 12 \\ x_1 + x_2 &\leq 4 \\ 5\,x_1 + 3x_2 &\geq 8 \\ x_1, x_2 &\geq 0 \end{aligned}$$

The optimal solution to this problem should be x = [0, 4,0,0]$^T$. In this case, after plugging in

$$\begin{aligned} A &= [1 \quad 3] \\ b &= 12 \\ c &= [-3 \quad -5]^T \\ \hat{A} &= [1 \quad 1] \\ \hat{b} &= 4 \\ \tilde{A} &= [5 \quad 3] \\ \hat{b} &= 8 \end{aligned}$$

into my function `GEN_simplex_imp`, one gets

```
A_trans =

     1     3     0     0
     1     1     1     0
     5     3     0    -1


b_trans =

    12
     4
     8


c_trans =

    -3
    -5
     0
     0

Solution =

   -0.0000
    4.0000
    4.0000


BasicVar =
```

1 2 4

which is the exact solution. My general implementation was successful!

**PROBLEM 3 – STUDY OF $L^2$ VS $L^1$APPROXIMATION:**

Problem 3 consisted of comparing the differences between $L^2$ approximation vs $L^1$approximation.

- **PROBLEM 3-DATA FOR REGRESSION ANALYSIS:**
  - In order to compare the regressions methods, I used 10-year US Treasure interest (TNX) and Dow Jones stock index price (TNX) data from 11/05/15 to 12/3/18. My x values represented TNX, and my y values represented DJI.

| Date | TNX | DJI |
|---|---|---|
| 11/5/18 | 3.201 | 25461.6992 |
| 11/6/18 | 3.214 | 25635.0098 |
| 11/7/18 | 3.213 | 26180.3008 |
| 11/8/18 | 3.234 | 26191.2207 |
| 11/9/18 | 3.189 | 25989.3008 |
| 11/12/18 | 3.186 | 25387.1797 |
| 11/13/18 | 3.145 | 25286.4902 |
| 11/14/18 | 3.12 | 25080.5 |
| 11/15/18 | 3.118 | 25289.2695 |
| 11/16/18 | 3.074 | 25413.2207 |
| 11/19/18 | 3.057 | 25017.4395 |
| 11/20/18 | 3.048 | 24465.6406 |
| 11/21/18 | 3.061 | 24464.6895 |
| 11/23/18 | 3.054 | 24285.9492 |
| 11/26/18 | 3.072 | 24640.2402 |
| 11/27/18 | 3.055 | 24748.7305 |
| 11/28/18 | 3.044 | 25366.4297 |
| 11/29/18 | 3.035 | 25338.8398 |
| 11/30/18 | 3.013 | 25538.4609 |
| 12/3/18 | 2.992 | 25826.4297 |

- $L^2$ approximation solution (refer to **APPENDIX 3.1**) was definitely faster to perform given that it had an analytical. I used a function I called `L2`. $L^2$ approximation is essentially minimizing the quadratic function:

$$f(a,b) = \frac{1}{2}\begin{bmatrix} a \\ b \end{bmatrix}^T \begin{bmatrix} 2\sum_{k=1}^{N} x_k{}^2 & 2\sum_{k=1}^{N} x_k \\ 2\sum_{k=1}^{N} x_k & 2N \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} - \begin{bmatrix} 2\sum_{k=1}^{N} x_k y_k \\ 2\sum_{k=1}^{N} y_k \end{bmatrix}^T \begin{bmatrix} a \\ b \end{bmatrix} + 2\sum_{k=1}^{N} (y_k{}^2)$$

by setting the gradient $\nabla f(a,b) = 0$ solving for (a,b), one gets a ***unique*** analytic solution:

$$\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 2\sum_{k=1}^{N} {x_k}^2 & 2\sum_{k=1}^{N} x_k \\ 2\sum_{k=1}^{N} x_k & 2N \end{bmatrix}^{-1} \begin{bmatrix} 2\sum_{k=1}^{N} x_k y_k \\ 2\sum_{k=1}^{N} y_k \end{bmatrix}.$$

Hence, I got the following results:

Slope Approx

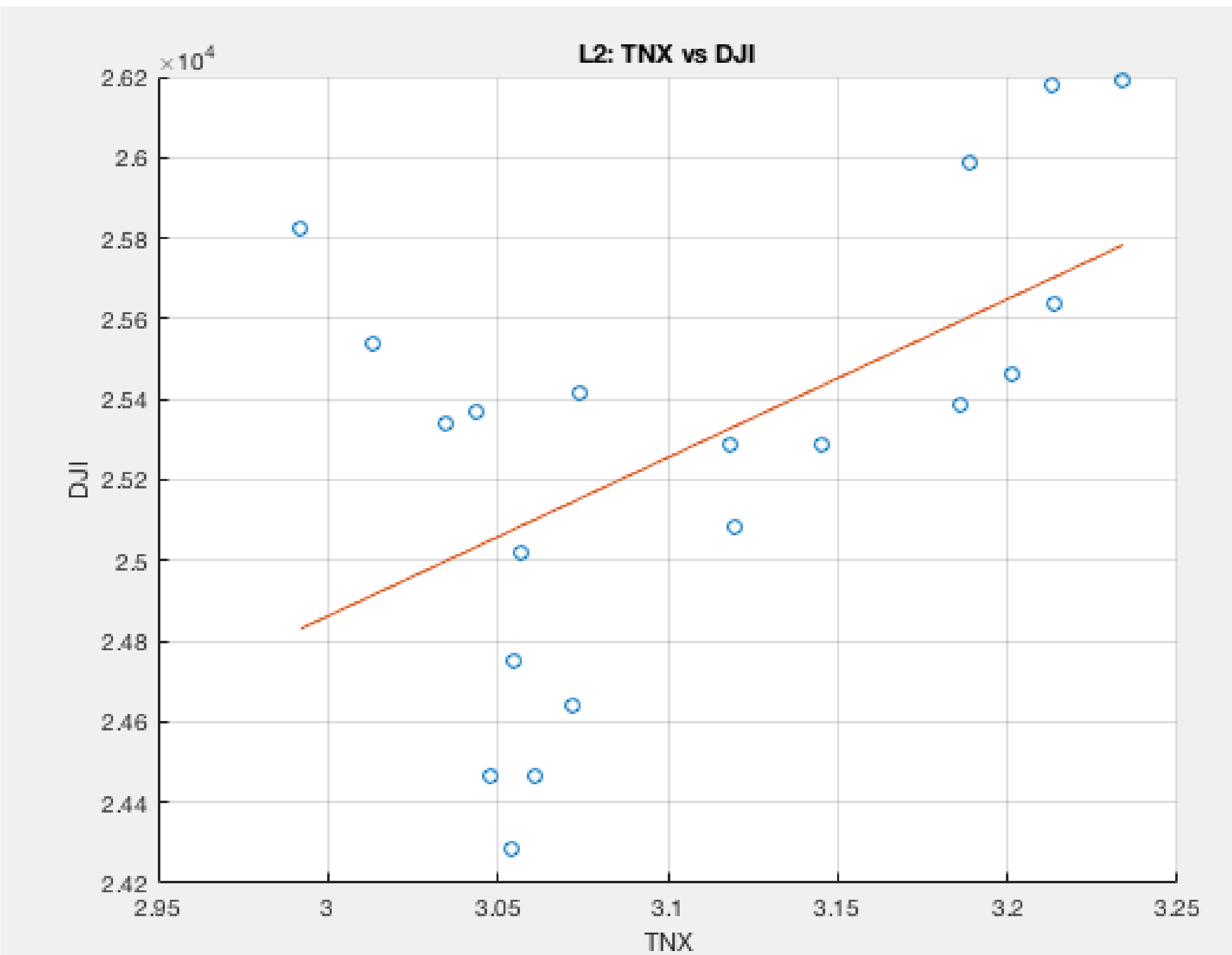
a =

3.9350e+03

Intercept Approx

b =

1.3057e+04



- L[1] approximation (refer to **APPENDIX 3.2**) takes longer but it is more robust and less prone to round off error if N is massive. I used a function I called `L1`. However, first, as the professor alluded to in the prompt, if we let $w_k^+, w_k^- \geq 0 \; for \; k = 1, \ldots, N$, such that $w_k^+ - w_k^- + ax_k + b = y_k$, L[1] approximation becomes the LP problem:

$$Minimize \sum_{k=1}^{N} (w_k^+ + w_k^-)$$
$$subject \; to \; w_k^+ - w_k^- + ax_k + b = y_k \;\; for \; k = 1, \ldots, N$$
$$w_k^+, w_k^- \geq 0 \;\; for \; k = 1, \ldots, N$$

WHERE $w_k^+ w_k^- = 0 \; for \; k = 1, \ldots, N$.

Now we can turn this problem into vector standard form:

$$Minimize \; c^T z$$
$$subject \; to \; Az = Y$$
$$w_k^+, w_k^- \geq 0 \;\; for \; k = 1, \ldots, N$$
$$WHERE \quad A = \begin{bmatrix} [\mathbf{1}, \mathbf{-1}] & \cdots & \mathbf{0} & \vdots & \vdots \\ \vdots & \ddots & \vdots & \boldsymbol{x} & \mathbf{1} \\ \mathbf{0} & \cdots & [\mathbf{1}, \mathbf{-1}] & \vdots & \vdots \end{bmatrix}$$
$$Y = [y_1 \;\; y_2 \ldots \; y_N]^T \in R^N$$
$$c = [1 \quad 1 \quad \ldots \quad 1 \quad 1 \quad 0 \quad 0]^T \in R^{N+N+2}$$
$$z = [w_1^+ \;\; w_1^- \ldots \; w_N^+ \;\; w_N^- \quad a \quad b]^T \in R^{N+N+2}$$

After calling `L1`, I got the following results:

Slope Approx

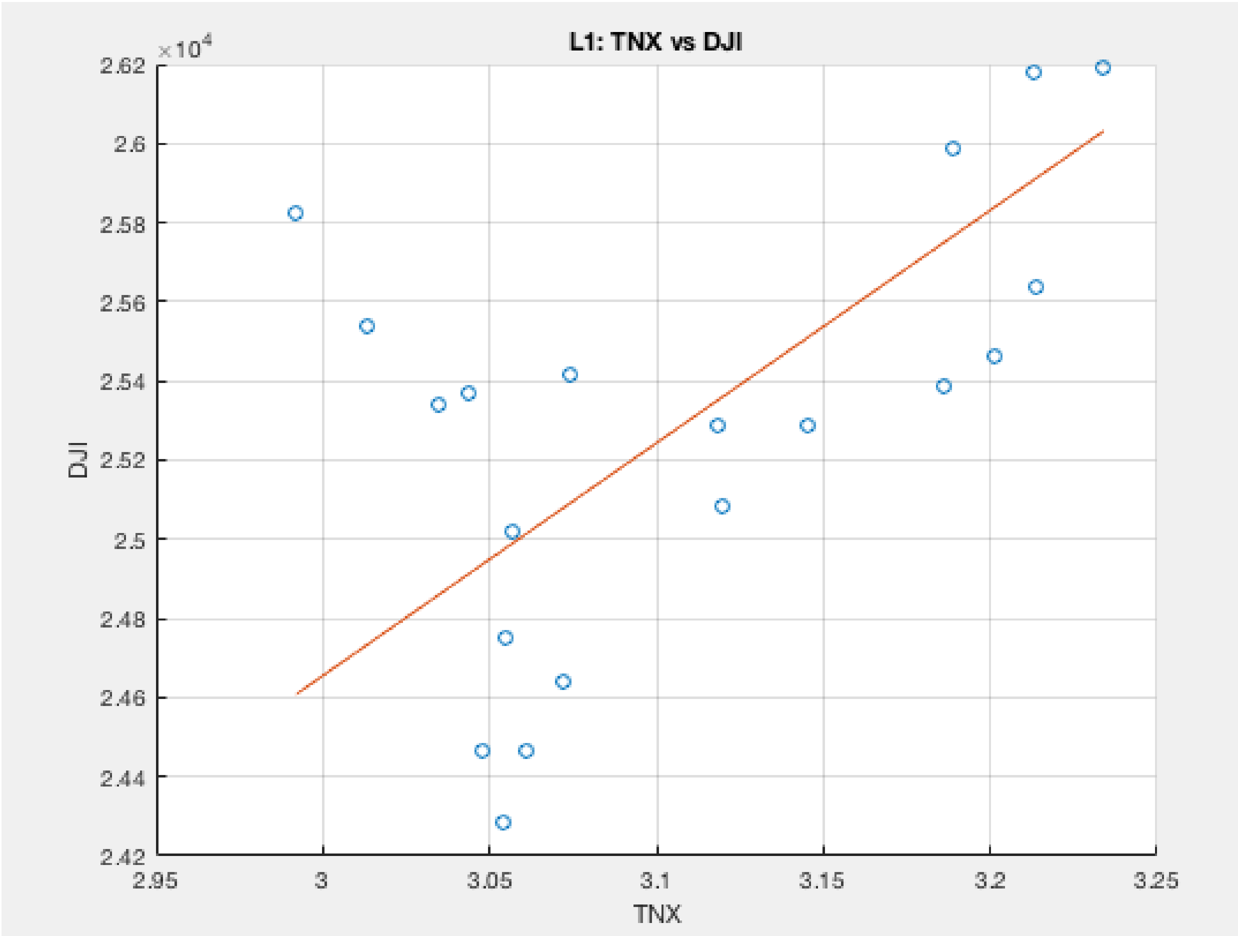
a =

5.8793e+03

Intercept Approx

b =

7.0177e+03




**CONCLUSIONS:**

I was able to successfully implement the both a basic simplex algorithm and a simplex algorithm that solved a general linear programming problem. With this, I was able to perform $L^1$ approximation. In the case of $L^2$ approximation, the solution to find (a,b) was purely analytical. Now, when comparing $L^2$ approximation to $L^1$ approximation, $L^2$ tends to accumulate more round off error given that it is performing an operation of form $x = Qb^{-1}$ which is essentially just doing Gaussian Elimination which is not very robust too. On the other hand, $L^1$ approximation doesn't accrue much error! Still, $L^1$ approximation does have weakness in that the solution one gets may not be unique. This means that the result isn't easily calculated like $L^2$ approximation. $L^2$ approximation does have a unique though which is partly why it is widely used. Still, one can get very similar solutions using either method.

## APPENDIX:

### APPENDIX 1.1

```
function [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:
%       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
BasicVar0=unique(BasicVar0);
if numel(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
b=reshape(b,mConstr,1);
c=reshape(c,ndim,1);
%
% Initialize the simplex table
%
A_basic=A(:,BasicVar0);
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
```

## APPENDIX 1.2

```
%% Problem 1: Implementation of the basic simplex algorithm
function [Solution,BasicVar,Status] = simplex_imp(A,b,c)

% Usage: [Solution,BasicVar,Status]=simplex_imp(A,b,c)
%   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.
%
%   Outputs:
%       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.

% Line 20 - 28 are code from professor's basic simplex alg
[mConstr,ndim]=size(A);

% Initial indices of basis variables for the solution.
BasicVar0 = [1:mConstr];
% Basis of A
A_basic = A(:,BasicVar0);

%According to matlab A\b is more accurate than inv(A)*b
x_A_basic = A_basic\b';

% Index denoting the rows that we are searching through for the matrix on
% line 36.
i = 1;

%Indices of columns
v = 1:ndim;

% Matrix of different combinations of the indices for BasicVar0
B = nchoosek(v,mConstr); %According to Matlab Documentation,
                              %"C = nchoosek(v,k) returns a matrix
                              %containing all possible combinations of
                              %the elements of vector v taken k at a time.
                              %Matrix C has k columns and n!/((n?k)! k!)
                              %rows, where n is length(v)."

% This section a mirror image of the type of operation the professor did.
%Essentially we are trying to determine what will be the BasicVar0 we use.
while min(x_A_basic)<0
    BasicVar0 = B(i,:);
    A_basic = A(:,BasicVar0);
    x_A_basic = A_basic\b;
    i = i+1;
end
```

```
%Implementing professor's basic simplex algorithm!
[Solution,BasicVar,Status]=basicsimplex(A,b,c,BasicVar0);

end
```

**APPENDIX 2**

```
%% Problem 2:Implementation of general LP alg.
function [Solution,BasicVar,Status] =
GEN_simplex_imp(A,b,c,A_hat,b_hat,A_tild,b_tild)
%
% Usage: [Solution,BasicVar,Status]=
GEN_simplex_imp(A,b,c,A_hat,b_hat,A_tild,b_tild)
%   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.
%
%       A_hat       : Array of dimension m_hat by n for the inequality
constraint
%                    Ax<=b.
%       b_hat       : Vector of dimension m_hat for the right hand side of the
%                    inequality constraint Ax<=b.
%       A_tild      : Array of dimension m_tild by n for the inequality
constraint
%                    Ax>=b.
%       b_tild      : Vector of dimension m_tild for the right hand side of
the
%                    inequality constraint Ax>=b.
%   Outputs:
%       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);
m_hat = length(b_hat);
m_tild = length(b_tild);
c_trans = zeros((ndim+m_hat+m_tild),1);

A_trans = zeros((mConstr+m_hat+m_tild),(ndim+m_hat+m_tild));
b_trans = zeros((mConstr+m_hat+m_tild),1);

% Transformation of A's
A_trans = transformingA(mConstr,ndim,A_trans,A) +...
transformingA_hat(mConstr,m_hat,ndim,A_trans,A_hat) +...
transformingA_tild(mConstr,m_hat,m_tild,ndim,A_trans,A_tild) +...
creatingI_hatI_tild(mConstr,m_hat,m_tild,ndim,A_trans)

% Transformation of b's
b_trans = transformingb_hat(mConstr,m_hat,b_trans,b_hat) +...
    transformingb_tild(mConstr,m_hat,m_tild,b_trans,b_tild) +...
    transformingb(mConstr,b_trans,b)

% Transformation of c
```

```
c_trans(1:ndim) = c

% Get Solution x*
[Solution,BasicVar,Status] = simplex_imp(A_trans,b_trans,c_trans);
end
```

**APPENDIX 2a**

```
%% Inserting the Constraint A into A_trans
function [A_trans] = transformingA(mConstr,ndim,A_trans,A)
A_trans(1:mConstr,1:ndim) = A;
end
```

**APPENDIX 2b**

```
%% Inserting A_hat Constraint into A_trans
function [A_trans] = transformingA_hat(mConstr,m_hat,ndim,A_trans,A_hat)
A_trans((mConstr+1):(mConstr+m_hat),1:ndim) = A_hat;
end
```

**APPENDIX 2c**

```
%% Inserting A_tild Constraint into A_trans
function [A_trans] =
transformingA_tild(mConstr,m_hat,m_tild,ndim,A_trans,A_tild)
A_trans((mConstr+m_hat+1):(mConstr+m_hat+m_tild),1:ndim) = A_tild;
end
```

**APPENDIX 2d**

```
%% Inserting I corresponding to m_hat and -I corresponding to m_tild into
A_trans
function [A_trans] = creatingI_hatI_tild(mConstr,m_hat,m_tild,ndim,A_trans)
A_trans((mConstr+1):(mConstr+m_hat),(ndim+1):(ndim+m_hat)) = eye(m_hat);
A_trans((mConstr+m_hat+1):(mConstr+m_hat+m_tild),(ndim+m_hat+1):(ndim+m_hat+m
_tild)) = -eye(m_tild);
end
```

**APPENDIX 2e**

```
%% Inserting b Constraint into b_trans
function [b_trans] = transformingb(mConstr,b_trans,b)
b_trans(1:mConstr) = b;
end
```

**APPENDIX 2f**

```
%% %% Inserting b_tild Constraint into b_trans
function [b_trans] = transformingb_tild(mConstr,m_hat,m_tild,b_trans,b_tild)
b_trans((mConstr+m_hat+1):(mConstr+m_hat+m_tild)) = b_tild;
end
```

**APPENDIX 2g**

```
%% Inserting b_hat Constraint into b_trans
function [b_trans] = transformingb_hat(mConstr,m_hat,b_trans,b_hat)
b_trans((mConstr+1):(mConstr+m_hat)) = b_hat;
end
```

**APPENDIX 3.1**

```
%% L2 Approximation for y= ax+b
function p = L2(TNX,DJI)
N = length(TNX);

Q = [sum(2*dot(TNX,TNX)) sum(2*TNX); sum(2*TNX) 2*N];
b = [sum(2*dot(TNX,DJI)); sum(2*DJI)];

% Vector p =[a,b]
p = Q\b;

% Slope approx
disp('Slope Approx')
a = p(1)

%Intercept approx
disp('Intercept Approx');
b = p(2)

hold on
grid on
scatter(TNX,DJI);
f = plot(TNX,a*TNX+b)
title('L2: TNX vs DJI')
xlabel('TNX')
ylabel('DJI')
hold off

end
```

**APPENDIX 3.2**

```
%% L1 Approximation for y = ax+b
function [a,b] = L1(TNX,DJI,linear_relationship,intercept_position)

% Initializing Parameter values for computation speed
mConstr = length(TNX);
BasicVar0 = 1:mConstr;
A_L1 = zeros(mConstr,(2*mConstr+2)); %matrix A
c = ones((2*mConstr+2),1); %coefficients are full of 1s

% creatingMatrix A
A_L1 =
creatingmatrixA_L1(mConstr,A_L1,TNX,linear_relationship,intercept_position);
```

```
%Implements professor's simplex method to find a and b
[Solution,BasicVar,Status] = basicsimplex(A_L1,DJI,c,BasicVar0);

% Slope approx
disp('Slope Approx')
a = linear_relationship*Solution(2*mConstr+1)

%Intercept approx
disp('Intercept Approx');
b = intercept_position*Solution(2*mConstr+2)

hold on
grid on
scatter(TNX,DJI);
g = plot(TNX,a*TNX+b)
title('L1: TNX vs DJI')
xlabel('TNX')
ylabel('DJI')
hold off

end
```

**APPENDIX 3.2a**

```
%% Creating matrix A for L1 Reg
function [A_L1] =
creatingmatrixA_L1(mConstr,A_L1,x,slope_direction,intercept_position)
for i = 1:mConstr
        A_L1(i,i) = 1;
        A_L1(i,(i+1)) = -1;
        if slope_direction<0
        A_L1(i,(2*mConstr+1)) = -x(i);
        else
            A_L1(i,(2*mConstr+1)) = x(i);
        end
        if intercept_position<0
        A_L1(i,(2*mConstr+2)) = -1;
        else
            A_L1(i,(2*mConstr+2)) = 1;
        end
end
end
```

**REFERENCES:**

Chong, Edwin K. P., and Stanislaw H. Zak. An Introduction to Optimization, John Wiley & Sons, Incorporated, 2014. ProQuest Ebook Central, https://ebookcentral.proquest.com/lib/socal/detail.action?docID=1124000.