24MAC175 Group 1 Coursework Report

Matthew Lakin, James Death, Nathan Moore, Saad Tahir

November - December 2024

Contents

1 Algorithm Explanation

The question we recieved isn't able to be solved using the Simplex Method as it is. The Simplex Method is used to solve linear programming problems exclusively in the canonical form. So the first step is to sanitise the question and put it in the correct form.

1.1 Converting to Canonical Form

The question we were asked to solve was as follows:

Minimise $z = 7x_1 + 11x_3 - 10x_4 - x_5 + 26x_6$,
Subject to constraints:

$$x_1 - x_2 + x_3 + x_5 + x_6 = 76 \tag{1.1}$$
$$x_2 - x_3 + x_4 + 3x_6 \leq 18 \tag{1.2}$$
$$x_1 + x_2 - 3x_3 + x_4 + x_5 \leq 12 \tag{1.3}$$
$$x_1 + x_2 + x_6 \geq 50 \tag{1.4}$$

with all variables non-negative: $x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$.

The first step in solving this is to put it into canonical form. Canonical form is where the question is in the form:

$$\text{Minimise } z = \underline{c}^T \cdot \underline{x} \tag{1.5}$$
$$A\underline{x} = \underline{b} \tag{1.6}$$
$$\underline{x} \geq 0 \tag{1.7}$$

For this, we add slack and surplus variables to (1.1), (1.2), and (1.3) to remove the inequality in favour of an equality:

Minimise $z = 7x_1 + 11x_3 - 10x_4 - x_5 + 26x_6$, subject to

$$x_1 - x_2 + x_3 + x_5 + x_6 = 76 \tag{1.8}$$
$$x_2 - x_3 + x_4 + 3x_6 + S_1 = 18 \tag{1.9}$$
$$x_1 + x_2 - 3x_3 + x_4 + x_5 + S_2 = 12 \tag{1.10}$$
$$x_1 + x_2 + x_6 - S_3 = 50 \tag{1.11}$$

with all variables non-negative: $x_1, x_2, x_3, x_4, x_5, x_6, S_1, S_2, S_3 \geq 0$.

However (1.11) violates the non-negativity constraint, so to fix that we need to introduce artificial variables to the LPP. In order to make a basic feasible solution we need to add artificial variables to both (1.8) and (1.11). We will also amend the objective function artificial variable coefficients with a large M value. This will be used in calculations so the artificial variables tend to 0 as iterations progress. So now the problem becomes:

Minimise $z = 7x_1 + 11x_3 - 10x_4 - x_5 + 26x_6 + M(R_1 + R_2)$, subject to

$$x_1 - x_2 + x_3 + x_5 + x_6 + R_1 = 76 \tag{1.12}$$
$$x_2 - x_3 + x_4 + 3x_6 + S_1 = 18 \tag{1.13}$$
$$x_1 + x_2 - 3x_3 + x_4 + x_5 + S_2 = 12 \tag{1.14}$$
$$x_1 + x_2 + x_6 - S_3 + R_2 = 50 \tag{1.15}$$

with all variables non-negative: $x_1, x_2, x_3, x_4, x_5, x_6, S_1, S_2, S_3, R_1, R_2 \geq 0$.

The LPP is now in canonical form and is ready for simplex.

1.2 Revised Simplex Algorithm

The Revised Simplex Algorithm is a variant of the Simplex Algorithm. Instead of using a tableau, it uses a matrix to store values.

The question consists of m constraints and n variables. We use slack, surplus and artificial variables to create a basic feasible solution. Iterating through all possible basic solutions is not a viable method since the amount of possible basic solutions is $\binom{n}{m}$ which grows factorially.
Our implementation of the Revised Simplex Algorithm follows these steps:

1. Partition the matrix into the basis and non-basis variables, B and N respectively.
2. Compute a basic solution.
3. Using the dual matrix, compute the reduced costs.
4. If all reduced costs are non-negative, the solution is optimal and the algorithm terminates.
5. If not, select the most negative reduced cost as the entering variable.
6. A direction vector d_B is calculated by multplying the B matrix with the vector of the entering variable.
7. We find the ratios between the right-hand side and the direction vector. The smallest ratio is the leaving variable.
8. We update the basis and repeat the process until the solution is optimal.

2 Worked Example

On paper, we solved a slightly simpler version of the question we were given. We did this to verify the Simplex Method. The solution is below:

Minimise $Z = -6x_1 + 7x_2 + 4x_3$

Subject to:

$$\begin{aligned} 2x_1 + 5x_2 - x_3 &\leq 18 \quad - \ I \\ x_1 - x_2 - 2x_3 &\leq -14 \quad - \ II \\ 3x_1 + 2x_2 + 2x_3 &= 26 \quad - \ III \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

Above function is equivalent to maximimising:

$$Z' = 6x_1 - 7x_2 - 4x_3$$

The constraints will remain the same, but we can multiply II by -1:

$$\begin{aligned} 2x_1 + 5x_2 - x_3 &\leq 18 \quad - I \\ -x_1 + x_2 + 2x_3 &\geq 14 \quad - II \\ 3x_1 + 2x_2 + 2x_3 &= 26 \quad - III \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

Standard Form

We introduce the artificial / slack variables to remove inequalities:

$$\begin{aligned} I \rightarrow \quad & 2x_1 + 5x_2 - x_3 + s_1 = 18 \\ II \rightarrow \quad & -x_1 + x_2 + 2x_3 - s_2 + a_2 = 14 \\ III \rightarrow \quad & 3x_1 + 2x_2 + 2x_3 + a_3 = 26 \\ \text{where:} \quad & x_1, x_2, x_3, s_1, s_2, a_2, a_3 \geq 0 \end{aligned}$$

Next, we add a large artificial variable penalty M for a_2 and a_3, where M is a large positive number:

$$Z' = 6x_1 - 7x_2 - 4x_3 - Ma_2 - Ma_3$$

Initial Simplex Tableau

Pivot column due to 4M (largest positive value)

	Basis	C_B	x_1 6	x_2 -7	x_3 -4	s_1 0	s_2 0	a_2 -M	a_3 -M	b	Ratio
	s_1	0	2	5	-1	1	0	0	0	18	18/-1 (ignore since negative)
Pivot Row (smallest ratio)	a_2	-M	-1	1	2	0	-1	1	0	14	14/2 = 7
	a_3	-M	3	2	2	0	0	0	1	26	26/2 = 13
	Z_j		-2M	-3M	-4M	0	M	-M	-M	-40M	
	$C_j - Z_j$		6+2M	-7+3M	-4+4M	0	-M	0	0		

Since a_2 is an artificial variable and is the pivot row, we will remove it from the columns and replace it in the row with x_3

Iteration 1

$\text{Row } 2_{old} / 2 = \text{Row } 2_{New}$

$\text{Row } 1_{old} + \text{Row } 2_{New} = \text{Row } 1_{New}$

$\text{Row } 3_{old} - \text{Row } 2_{old} = \text{Row } 3_{New}$

Pivot column

Basis	C_B	x_1 6	x_2 -7	x_3 -4	s_1 0	s_2 0	a_3 -M	b	Ratio
s_1	0	1.5	5.5	0	1	-0.5	0	25	25/1.5 = 50/3
x_3	-4	-0.5	0.5	1	0	-0.5	0	7	(ignore since negative)
a_3	-M	(4) Pivot element	1	0	0	1	1	12	12/4 = 3
Z_j		2-4M	2-M	-4	0	2-M	-M	-28-12M	
$C_j - Z_j$		4+4M	-5+M	0	0	-2+M	0		

Since a_3 is an artificial variable and the pivot row, we will remove it from the columns and replace it in the row with x_1

Iteration 2

Row 3_{old} / 4 = Row 3_{New}
Row 1_{old} $-\frac{3}{2}$ Row 3_{New} = Row 1_{New}
Row 2_{old} $+\frac{1}{2}$ Row 3_{New} = Row 2_{New}

Basis	C_B	x_1 6	x_2 −7	x_3 −4	s_1 0	s_2 0	b	Ratio
s_1	0	0	5.125	0	1	−0.875	20.5	N/A
x_3	−4	0	0.625	1	0	−0.375	8.5	
x_1	6	1	0.25	0	0	0.25	3	
	Z_j	6	−1	−4	0	3	−16	
$C_j - Z_j$		0	−6	0	0	−3	≤ 0	

Since no positive value in the net evaluation row, objective function cannot be optimised further.

Solution

$x_1 = 3$
$x_2 = 0$
$x_3 = 8.5$
$s_1 = 20.5$
$s_2 = 0$
$Z = 16$ $\Rightarrow$ Since we are ultimately looking to minimise Z, we can multiply the objective function by −1

3 Implementation

For this project, we used Python since it has access the NumPy module and MatPlotLib for graphing. NumPy is more efficient than basic Python script since it has been written in C. The code was written in a modular fashion, with each function performing a specific task.

3.1 Representation of the Problem

Before doing any simplex calculations on the question, we had to parse the question into a form that Python could understand. For our representation, we used 5 data structures:

- Nature: This is a binary value that represents whether the problem is a maximisation or minimisation problem. If the problem is a maximisation problem, the value is -1; if it is a minimisation problem, it is 1.
- A: This is a matrix that represents the coefficients of the constraints. Each row represents a constraint, and each column represents a variable.
- b: This is a vector that represents the right-hand side of the constraints. Each row represents a constraint.
- c: This is a vector that represents the coefficients of the objective function. Each column represents a variable.
- Signs: This is a vector that represents the signs of the constraints. Each row represents a constraint. The value -1 represents a less than or equal to constraint, 0 represents an equal to constraint, and 1 represents a greater than or equal to constraint.

As we have code to convert the LPP (Linear Programming Problem) to canonical form, we can accept any LPP in any form.

3.2 Main Method

The main method is the entry point for the program. It is where the question is inputted and the functions are called. When using this project, this is the only method that needs to be interacted with for solutions to LPP's.

3.3 Converting to Canonical Form

The function convertToCanonicalForm in file bfs.py is used to convert the LPP to canonical form. The function takes the LPP in our parsing style as input and returns the LPP in canonical form. We added a check to see if the LPP is valid before running the function as a form of data validation. It takes each constraint and does 1 of 3 things:

- If the constraint is an equality constraint, it adds an artificial variable to the constraint.
- If the constraint is a less than or equal to constraint, it adds a slack variable to the constraint.
- If the constraint is a greater than or equal to constraint, it adds a surplus and an artificial variable to the constraint.

Then the function makes all the constraints into equalities.
The function also ammends the objective function to include the artificial variables with a large M value coefficient if required, and then returns the LPP in canonical form.

3.4 Simplex Method

The function revisedSimplexMethod in file simplex.py is used to solve the LPP in canonical form. This uses the exact same algorithm as described in the Algorithm Explanation section. It uses the optimality test to determine if the current solution is optimal. If it is not, it gives the entering variable and uses the feasibility test to find the leaving variable. It then updates the bases and repeats until the solution is optimal.
It also checks for infeasibility and unboundedness. If the LPP is infeasible, it returns "Solution is Infeasible". If the LPP is unbounded, it returns "Unbounded solution".

3.5 Parsing the LPP

The function renderLPP in file render.py is used to render the LPP. It supports LPP's in both canonical and non-canonical form. The function takes the LPP in our parsing style as input and returns a string representation of the LPP.

3.6 Graphing

The file graphing.py is used to make the graph of the value of the objective function as the b value of each constraint is varied independantly. This file is self-contained and is ran separately from the main code.
The file variableGraphing.py is used to graph the specific variables in the solution as the b value of a constraint is varied. This file is self-contained and is ran separately from the main code.

3.7 Using the Code

To use the code to answer the provided question, you must verify that variables in main.py are set correctly. The variables should be as follows:

```
nature = 1
c = nature * np.array([7, 0, 11, -10, -1, 26])
A = np.array([[1, -1, 1, 0, 1, 1], [0, 1, -1, 1, 0, 3], [1, 1, -3, 1, 1, 0], [1, 1,
    0, 0, 0, 1]])
b = np.array([76, 18, 12, 50])
signs = np.array([0, -1, -1, 1])
```

Once you run main.py, you should see the multiple iterations with each entering and leaving variable and at the end the optimal solution. If the solution is infeasible or unbounded, it will print that instead:

```
Optimal Value:  200
Optimal Solution:  [49.2  0.8 27.6 44.8  0.   0.   0.   0.   0.   0.   0. ]
Status:  Optimal solution found
```

4 Special Cases

4.1 Infeasible Solutions

We performed an analysis which investigates whether the solution remains feasible when the constraints of the LPP are altered by varying the value of b. This was completed using MATLAB's LPP solver. The package systematically adjusted the b values of each constraint independently and gave the number 1 when the solution was feasible and 0 when unfeasible. We concluded when adjusting constraint 1, the solution became infeasible at an extreme b value. This occurred when the b value was decreased by 100 from 76 to -24. We then verified the solution was infeasible using the Python "SciPy" simplex package and it produced the following:

```
The problem is infeasible. (HiGHS Status 8: model_status is Infeasible;
    primal_status is At lower/fixed bound)
```

From this analysis we now have an example of a definitive infeasible problem to test against the implemented algorithm. At first when testing this problem we had to alter the input slightly to accept the negative b value. We did this by multiplying the whole constraint 1 by -1 and inputting it into the code as the following parameters:

```
nature = 1
c = nature * np.array([7, 0, 11, -10, -1, 26])
A = np.array([[-1, 1, -1, 0, -1, -1], [0, 1, -1, 1, 0, 3], [1, 1, -3, 1, 1, 0], [1,
    1, 0, 0, 0, 1]])
b = np.array([24, 18, 12, 50])
signs = np.array([0, -1, -1, 1])
```

This should produce an infeasible solution as the b value is 24 in order to maintain a positive, however to compensate the first constraint in "A" has swapped signs in all values in order to simulate a b value of -24. After an initial test it did produce an optimal value as well as a corresponding solution of variables.

```
Optimal Value:  6682
Optimal Solution:  [ 0. 50. 32.  0.  0.  0.  6.  0. 58.  0.  0.]
Status:  Optimal solution found
```

Not only was the optimal value unusually large compared to previous data, but the optimal solution had an artificial variable in the basis. Since the simplex algorithm was not able to filter out all artificial variables to 0 the problem is indeed infeasible. To counteract this problem so that the algorithm can account for other potential infeasible solutions, we adapted the code to check the artificial variables in the optimal solution. It would conclude a status 'Optimal solution found' only if all artificial variables equated to 0. After rerunning the same problem it produced the following:

```
Solution is Infeasible.
```

4.2 Unbounded Solutions

Additionally we wanted to test the algorithm against an unbounded example. We found an unbounded example online that reads:

$$\text{Maximise } z = 6x + 8y \tag{4.1}$$
$$\text{Subject to } x - y \geq 0, \tag{4.2}$$
$$-x + 3y \leq 3, \tag{4.3}$$
$$x, y \geq 0 \tag{4.4}$$

In order to confirm it was an unbounded LPP we ran the input against the 'SciPy' package as before and it confirmed the LPP had an unbounded solution.

```
nature = -1
c = nature * np.array([6, 8])
A = np.array([[1, -1], [-1, 3], [1, 1]])
b = np.array([0, 3, 0])
signs = np.array([1, -1, 1])
```

Our implemented algorithm checks if the solution is unbounded each iteration, thus if it concludes the solution is unbounded it will change the status to 'Unbounded solution' . Against this example it worked as intended and printed the following, after 4 iterations, rather than an optimal solution:

```
Solution is Unbounded.
```

5 Verification

For verification, we admitted the question:
Minimise $z = 7x_1 + 11x_3 - 10x_4 - x_5 + 26x_6$, subject to

$$x_1 - x_2 + x_3 + x_5 + x_6 = 76 \tag{5.1}$$
$$x_2 - x_3 + x_4 + 3x_6 \leq 18 \tag{5.2}$$
$$x_1 + x_2 - 3x_3 + x_4 + x_5 \leq 12 \tag{5.3}$$
$$x_1 + x_2 + x_6 \geq 50 \tag{5.4}$$

with all variables non-negative: $x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$.
We then used the Python package SciPy to solve the question. The representation of the question in SciPy is as follows:

```
c = np.array([7, 0, 11, -10,-1,26])
Aup = np.array([[0, 1, -1, 1, 0, 3],
                [1, 1, -3, 1, 1, 0],
                [-1, -1, 0, 0, 0, -1]])
bup = np.array([18, 12, -50])
Aeq = np.array([[1, -1, 1, 0, 1, 1]])
beq = np.array([76])
```

Since SciPy's method doesn't support the greater than or equal to constraints, we had to alter (5.4) by multiplying both sides of the equation by -1 and flipping the inequality.

$$-x_1 - x_2 - x_6 \leq -50 \tag{5.5}$$

We then used the SciPy method linprog to solve the question. The results were as follows:

```
res = linprog(c,Aup,bup,Aeq,beq)
print(res)
```

```
message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
success: True
 status: 0
    fun: 200.0
      x: [ 4.920e+01  8.000e-01  2.760e+01  4.480e+01  0.000e+00
           0.000e+00]
    nit: 5
  lower:  residual: [ 4.920e+01  8.000e-01  2.760e+01  4.480e+01
                      0.000e+00  0.000e+00]
         marginals: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
                      1.000e+00  4.500e+01]
  upper:  residual: [        inf         inf         inf         inf
                             inf         inf]
         marginals: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
                      0.000e+00  0.000e+00]
  eqlin:  residual: [ 0.000e+00]
         marginals: [-1.000e+00]
ineqlin:  residual: [ 0.000e+00  0.000e+00  0.000e+00]
         marginals: [-9.000e+00 -1.000e+00 -9.000e+00]
mip_node_count: 0
mip_dual_bound: 0.0
mip_gap: 0.0
```

The solution of the question corresponds with the "fun" value in the dictionary output. The value of the objective function is 200.0, which matches the value we found using the Revised Simplex Method. This verifies that our implementation is correct for this question.

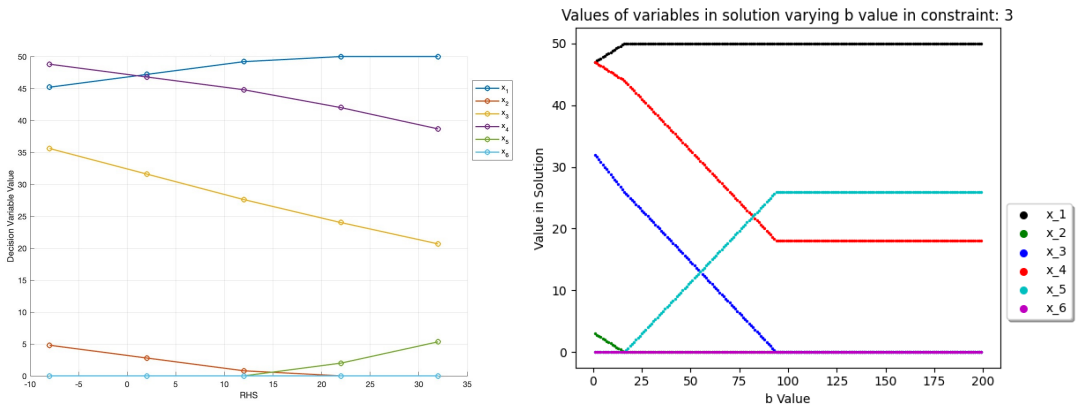


Figure 1: Left: MATLAB Sensitivity Analysis, Right: Python Sensitivity Analysis

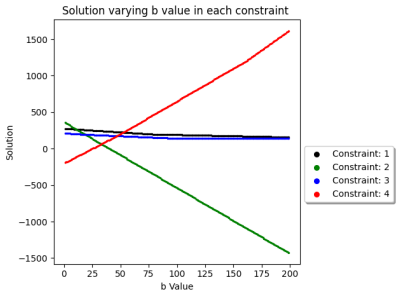
We displayed the variables in the solution changing b in constraint 3 calculated using a MATLAB package, and our implemented algorithm in Python.
We did this to ensure that all solutions match up with the MATLAB package.
Notable similarities can be observed between the graphs:

- x_1 and x_2 have an intersection at b = 1
- At b = 16, x_2 leaves the basis and x_5 enters the basis
- The trajectories of x_3 and x_5 are on course to intersect in the MATLAB graph as shown in the Python graph.

6 Sensitivity Analysis

6.1 Behaviour of the Objective Function

From this graph a clear idea of the strengths of the constraints can be shown. Firstly, constraints 2 and 4 show the greatest strength, this is shown through the smallest of adjustments to their value, massively impacts the solution. This can't be said for constraints 1 and 3, as from the graph it shows that when altered, the solution doesn't change much and stays fairly level. In order to produce a smaller optimal solution, increasing the b value of constraint 2 will minimise the solution the greatest. Constraint 4 is the opposite, where increasing the b will directly hinder the minimisation process by increasing the optimal solution the most.



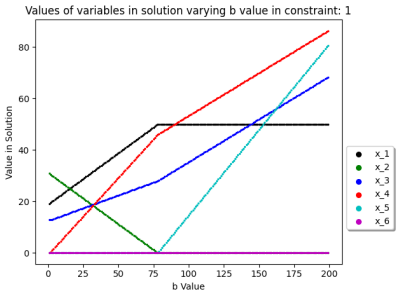
If the constraints were treated as a weight, then we can calculate their worth via the gradient of their respective lines in the graph:

- Constraint 1: at b=1, Solution = 276 and at b = 200, Solution = 157.33, Gradient = -0.596
- Constraint 2: at b=1, Solution = 357.33 and at b = 200, Solution = -1438, Gradient= -9.022
- Constraint 3: at b=1, Solution = 211 and at b = 200, Solution = 144, Gradient = -0.337
- Constraint 4: at b=1, Solution = -193.33333 and at b = 200, Solution = 1626, Gradient = 9.142

Since we are minimising the objective function in this question, the more negative the gradient, the more the constraint is contributing to the minimisation of the objective function.

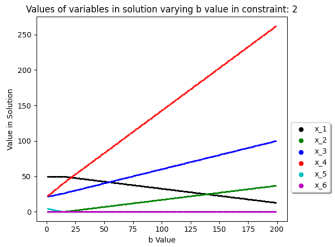
6.2 Constraint 1

It can be seen that both variables x_3 and x_4 steadily increase at near the same rate, as the b value is increased. It can also be observed that there is a shift in the basis variables of x_2 and x_5 which can be highlighted. This is where one variable is contributing, the other one is not involved and is zero. It can be seen that x_1, after the b value of 78, stays consistent from then on.



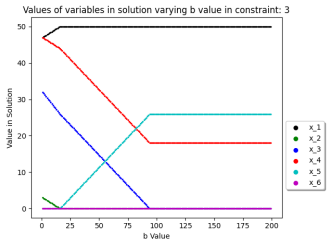
6.3 Constraint 2

We notice variable x_4 consistently increases as the b value is increased, this shows its importance and when relaxing the constraint resources are reallocated to x_4, for an optimal solution. The non-basic variable x_5 becomes active for the extremely low b values, where x_1 is seen to steadily decrease, crossing over with multiple other variables. Both variables x_2 and x_3 follow the same trajectory upwards as the b value is increased.



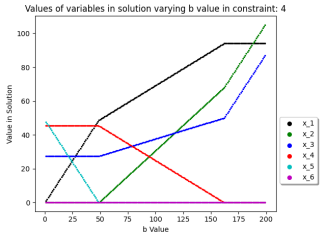
6.4 Constraint 3

Again, x_1 staying the same value from around b value of 17. This takes place at the same time as variable x_5 becomes active again and increases fast. The non-basic variable x_6 remains inactive, as it does in all constraints. Variable x_1 is observed to be very strong in this constraint and is clearly important for optimality.



6.5 Constraint 4

In this constraint the intersection of variables x_1 and x_4 can be seen clearly, where when the b value is increased, x_1 increases whilst x_4 decreases. This relationship suggests x_1 and x_4 are competing variables in this constraint and so, respond inversely to changes to the b value. The same x_2 and x_5 relationship is shown, with x_5 leaving, as x_2 enters. The variable x_3 is seen to steadily increase as the b value increases, when passed b = 50.



7 Conclusion

Through this project, we have produced a functioning implementation of the Revised Simplex Method in Python. We have verified the implementation using SciPy's linprog function and MATLAB's LPP solver.
We have analysed the behaviour of the objective function when varying the constraints of the LPP and explained what it means in the context of the question.

After much testing, we have calculated the result of:
Minimise $z = 7x_1 + 11x_3 - 10x_4 - x_5 + 26x_6$, subject to

$$x_1 - x_2 + x_3 + x_5 + x_6 = 76 \tag{7.1}$$
$$x_2 - x_3 + x_4 + 3x_6 \leq 18 \tag{7.2}$$
$$x_1 + x_2 - 3x_3 + x_4 + x_5 \leq 12 \tag{7.3}$$
$$x_1 + x_2 + x_6 \geq 50 \tag{7.4}$$

with all variables non-negative: $x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$.
is equal to 200 with the optimal solution of:

$$\underline{x} = \begin{bmatrix} 49.2 \\ 0.8 \\ 27.6 \\ 44.8 \\ 0 \\ 0 \end{bmatrix} \tag{7.5}$$

We parsed the question into this form:

```
nature = 1
c = nature * np.array([7, 0, 11, -10, -1, 26])
A = np.array([[1, -1, 1, 0, 1, 1], [0, 1, -1, 1, 0, 3], [1, 1, -3, 1, 1, 0], [1, 1,
    0, 0, 0, 1]])
b = np.array([76, 18, 12, 50])
signs = np.array([0, -1, -1, 1])
```

and produced the output of:

```
Optimal Value:  200
Optimal Solution:  [49.2  0.8 27.6 44.8  0.   0.   0.   0.   0.   0.   0. ]
Status:  Optimal solution found
```

This code can be used to solve other LPP's. It will also check for special cases such as infeasibility and unboundedness.