DUAL SIMPLEX METHOD

ALGORITHM

Function that generates a matrix of correct size:

Step – 1: First, we'll generate a numpy array with enough rows for each constraint plus the objective function and enough columns for the variables, slack variables, M (max/min) and the corresponding value.

Functions that check if the the current solution is optimal:

Step – 2: Next, we'll check to see if 1+ pivots are required due to a negative element in the furthest right column, excluding the bottom value, of course.

Similarly, we'll check to see if 1+ pivots are required due to a negative element in the bottom row, excluding the final value.

Functions that determines where a pivot element is located:

STEP - 3: And now that we've created functions that return booleans, whether or not additional pivots are required,

we need to determine where these elements are located. We'll start with finding negative elements in the furthest right column.

Similarly, we need to locate negative elements in the bottom row.

Function that pivots about an element:

STEP - 4: We've identified the column and row indexes, respectively, for negative elements in the last column, last row.

But we need to take it one step further and find the pivot element corresponding to these values.

Next, we need to find a pivot element corresponding to a negative element in the bottom row.

Now, pivot about an element to remove the negative entry in the final column or row and return the updated table.

Functions to receive string input and insert float variables into matrix:

STEP - 5: Now we need a means for the user to input a string, which will be converted into float variables.

Our function will receive inputs such as ('1,3,L,5'); this means $1(x1) + 3(x2) \leq 5$. Alternatively, 'G' could be used to mean a $\geq$ inequality.

Functions to maximize and minimize the problem:

STEP - 6: Our function is designed to solve any combination of variables and constraints.

So we need to build a function that will generate only the required number of variables x1, x2,…xn.

It follows that we need a means to check if 1+ constraints can be added to the matrix, meaning there are at least two rows of all 0 elements.

If this condition is not satisfied, our program will not allow the user to add additional constraints.

Now, we need to actually add the constraints to the problem. We can define just the function as is follows.

The function will take the tableau as an argument as well as the equation, which will be converted using the previous function and will be inserted into the tableau, appropriately.

Similar to the constrain() function we built, we also need a function to add the objective function to the tableau, given that it satisfies add_obj().

It's finally time to put all the building blocks together and create the maximization and minimization functions.

These functions will appear very similar, both will use while loops to determine if 1+ pivot is required, locate the pivot element,

pivot about it, and continue the process until all negative elements have been removed from the last column and row. Then variables will be

generated for x1 through xn and assigned values according to their positions in the tableau. Additionally, max will be assigned its appropriate value.

Lastly, the function will return the max and variables in dictionary form.