SOLVING LINEAR PROGRAMMING PROBLEMS USING SIMPLEX METHOD

DENYS TIASKO

Contents

Analysis

Identification of problem

When identifying an area to tackle for my project I was narrowed down to my interests that I am into during and out of school time, particularly mathematics and object-oriented programming. The problem being identified is solving real-life problems as linear programming problems using two-stage simplex method. Nowadays, there should be a straightforward, swift and convenient method for a user to receive an answer for his complex query, which might take him a long time to solve manually. Simplex method was a generally perfect idea to implement since there are a lot of instances in everyday life that this algorithm is the best at, such as managing the business, logistic and production planning, traffic routing and many others. Some simple optimization problems can be solved by a few different methods by sketching the constraints and an objective function on a graph that use different techniques. However, those methods are only valid when the objective function contains only 2 variables. In real life, there would be more features that affect the function value (e.g. cost) which results in using the simplex method. This method is one of the most efficient and useful algorithms ever invented that is easy to learn and is still used by computers to solve optimization problems.

Identification of users

Since my project was established to solve real-life problems, there is a wide range of potential users. First of all, my project may be attractive to A-level Further Mathematicians, who do Decision Mathematics as one of their choices. Students can use the program to verify their solution or follow the steps shown in the console in case they want to find and correct their mistake halfway through. Besides, this project may be suitable to business companies or industries. A good example might be Amazon using integer and linear programming for their vehicle routing or network design problems. In general, simplex method can be used in planning, scheduling, distribution, manufacturing and many others.

Existing systems

There are already existing web platforms that help you solve linear programming problems. For example, Wolfram Alpha has a linear programming solver which gives the most optimal values for variables but does not return the total of the objective function. Although the existing systems on the internet give you the solution, they do not provide you with the step-by-step solution which helps students understand how to solve such problems. My solution gives a good understanding of each step and visualization of the tableau. Although students will not be allowed to run this program in lessons or examinations, it can still be very beneficial to students doing homework or revision.

Research

As a student, I was very lucky to have a chance to talk to a couple of students who study Further Mathematics. After speaking to various people about their preferences for a linear programming solver, I have collected several features that should be implemented within my project that will make it more attractive and easier to use. Some of the things that potential clients established to be essential for the program are:

- A user menu which allows you to add/change objective function and constraints.
- It should have a straightforward user interface.
- It should show each iteration.
- It should return the optimal solution with values for objective function and variables.

Further Mathematics Decision 1 course covers only 2-stage simplex and Big-M method for solving linear programming problems. The syllabus is concise and gives a general idea about these two methods. Therefore, I decided to research new aspects of simplex algorithm on Internet and found a few special cases that I implemented in my code and will describe about it later in other parts of the document.

Functional requirements

1 Mainmenu class
- 1.1 UI is in the form of a simple menu in the console
- 1.2 Instructions must be printed before showing the menu
- 1.3 Menu options should be easy to read and choose
  - 1.3.1 Menu options will include:
    - 1.3.1.1 Display instructions – prints the instructions and explains what counts as the valid input
    - 1.3.1.2 Set an objective function – allows user to enter an objective function

1.3.1.3 Change the objective function – allows user to change an objective function
1.3.1.4 Display the objective function – prints the objective function
1.3.1.5 Add constraints – allows user to add constraints
1.3.1.6 Change a constraint – allows user to change a constraint
1.3.1.7 Delete a constraint – allows user to delete a constraint
1.3.1.8 Display the constraints – prints the constraints
1.3.1.9 Solve the problem – run the simplex method simulation
1.3.1.10 Exit the program – shuts down the program

1.4 Menu must be printed after each menu option execution (except for solve the problem and exit the problem)

2 User input

2.1 Digit
2.1.1 Input for a digit must be in [“0”-“9”]

2.2 Letter
2.2.1 Input for a variable must be in [“x”, “y”, “z”, “m”, “n”]

2.3 Sign
2.3.1 Input for a sign (operator or (in)equality) must be in [“>”, “>=”, “<”, “<=”, “=”, “+”, “-“]
2.3.2 User mustn’t use brackets [“(“, ”)”]
2.3.3 User mustn’t use “*” when trying to indicate the coefficients of a variable

2.4 User must input a digit in [0-9] for the number of constraints

2.5 Prompt input
2.5.1 Input must be one of the choices given by the prompt

2.6 Menu choice
2.6.1 Input must be one of the abbreviations on the left side of the menu

2.7 User mustn’t input any letters than those that are allowed

2.8 User must input integer values

2.9 User mustn’t input float values

2.10 Objective function
2.10.1 Number of coefficients must be equal to number of variables
2.10.2 Input must only consist with coefficients (timed by (-1)) and variables
2.10.3 If the coefficient is 1 or -1, user must enter it before the variable

2.11 Constraints
2.11.1 Number of coefficients must be one greater than the number of variables
2.11.2 Input must contain only one (in)equality sign and a value after the sign
2.11.3 If the coefficient is 1 or -1, user must enter it before the variable
2.11.4 If the coefficient is 0, user must enter it before the variable

2.12 Checking user input
2.12.1 Every time a user input is required, the program must check if the input is valid before proceeding with the request

3 Generate tableau
- 3.1 The program must generate a new tableau every time the user inputs new objective function or constraint
- 3.2 Tableau generating class must always:
  - 3.2.1 Have the following properties:
    - 3.2.1.1 Max_or_min
    - 3.2.1.2 Obj_function
    - 3.2.1.3 Constraints
    - 3.2.1.4 Tableau
    - 3.2.1.5 Artif_num
    - 3.2.1.6 Artif_vars_rows
  - 3.2.2 Be able to return tableau property
  - 3.2.3 Have functions' descriptions for the user to view and understand

4 Simplex method iteration
- 4.1 Program must be able to print every iteration
- 4.2 Program must be able to print a list of non-zero variables after the iteration
- 4.3 Program must be able to tackle special cases, such as degeneracy, unbounded solutions and infeasible(nonexistent) solutions

Non-functional requirements

1 Safety and security
- 1.1 Program is ended safely.
- 1.2 General system design must follow Interface – Model pattern
  - 1.2.1 Code is split into Model (2), UI and utilities files
    - 1.2.1.1 It averts external disruptions on data within the program

2 Usability
- 2.1 Simple user interface
  - 2.1.1 Interface is user-friendly and easy to use
  - 2.1.2 Easy-readable text
  - 2.1.3 Relevant indents
  - 2.1.4 Menu options should be clear and self-describing
  - 2.1.5 Appropriate and simple abbreviations for menu options

3 Speed
- 3.1 Any menu option function should immediately return output
- 3.2 Simplex method iterations should be done and printed within 2 seconds

4 Reliability
- 4.1 Program should not crush at any moment during execution

4.2 User can choose any menu option without failure

Extension objectives (not included in technical solution)

1 Graphic User Interface

1.1 Menu

1.1.1 For easier interaction between user and program, the menu should be graphical. This enables user to choose a menu option by clicking on it rather than input it in the console

1.1.2 In cases where a user has to input an objective function etc., GUI should have an input field for such requests

1.1.3 Simplex method iterations should be printed in GUI instead of console which allows user to track each step more easily

2 Saving and loading from file

2.1 User should have an option to save and load a run of the simulation from the file

2.1.1 Following elements should be saved to a file:

2.1.1.1 Properties

2.1.1.1.1 Max_or_min

2.1.1.1.2 Obj_function

2.1.1.1.3 Constraint

2.1.1.2 Every iteration with a list of non-zero variables

Documented Design

Overall design

Using simplex method, a popular algorithm for linear programming, my program can solve an optimization problem, which involves an objective function and several constraints expressed as inequalities. One of the most efficient way to run simplex algorithm is to create a tableau with coefficients of the function and constraints. Together with the tableau and algorithms of the method, I need multiple diagrams focused on different parts of the project in order to explain the functionality of the simulation.

Graphical representation

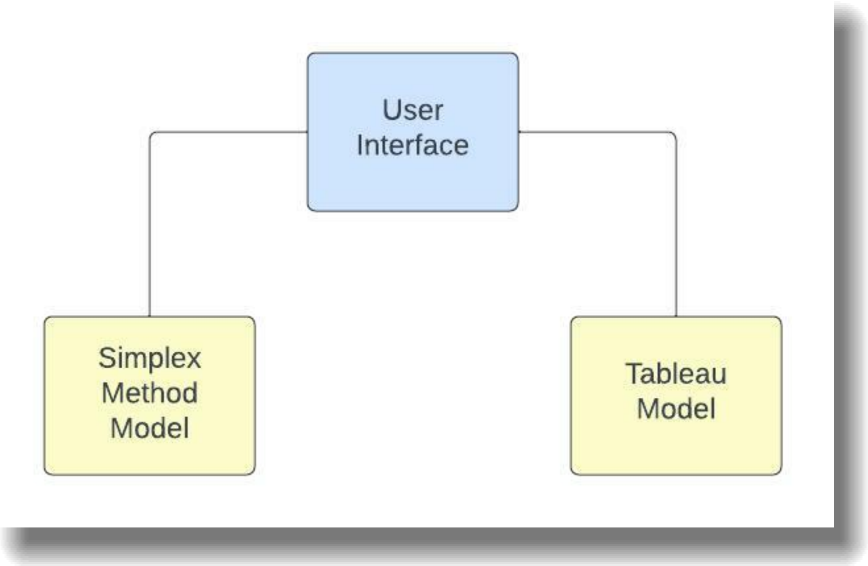


Diagram 1.1

The diagram above is graphical representation of the structure of my code. The program does not require any database or connection to a server. The architectural pattern of the code entails a physical separation of each stage of the code. It can simply be described in two layers:

- Interface layer(blue) – layer that user can control and perform actions in order to see the result of Model layer
- Model layer(yellow) – layer that models and runs the simulation

Every on the diagram is separated into individual file and set up as a class. The fact that they are set up as classes and being arranged in this pattern allows the following things:

- Easy to debug the code
- Easy to test particular pieces of code as the layers are separated
- Destruction of User Interface will destroy Simplex Method and Tableau models
- Easy to make improvements in each layer without affecting other layers

Description of each element in the diagram

User Interface: Top layer of the diagram. The UI is interacting with the user by printing prompts and asking for an input from a user when required. For example, before a player chooses an option to solve the problem, he will be asked to set an objective function and constraints and UI will check if the input is valid. After entering valid input, data will be passed to objects in yellow layer to model and run the simulation.

Tableau Model: bottom layer of the diagram. The user cannot interact with this model and it operates only with data passed to it. After successful run of operations in UI, Tableau Model will get special data and create a tableau for simplex method to operate on. For instance, after getting the coefficients of objective function and constraints, 2-D tableau is created with special order and position of coefficients.

Simplex Method Model: bottom layer of the diagram. Similar to Tableau Model, the user cannot interact with this model and it operates only with data passed to it. After effective use of UI and created tableau in Tableau Model, Simplex Method Model will run operations on the tableau in order to solve linear programming problem. For example, one of the functions of this model is to scale the tableau so that in one column there is only one “1” and other values are “0”. Further explanation and examples will be seen later in the project.

Class diagram

The use of class diagrams allows to visualize an overview of the program without the need-to-know implementation details. The class diagram below describes the proposed architecture in Diagram 1.1. When looking at the diagram, few important features should be noted:

- Diagram notation is in UML form:
  - + stands for a global method/field
  - - stands for a private method/field
  - Shaded rhombus entails composition association and its direction
  - Empty rhombus entails aggregation association and its direction
  - Class definitions(top-to-bottom): name, fields, methods
- There is a aggregation association between tableau and mainmenu classes as an instance of tableau class is created in solve() method
- There is a aggregation association between simplex_method and mainmenu classes as an instance of simplex_method class is created in solve() method
- There is a composition association between tableau and simplex_method classes as an instance of tableau class is created in the constructor of simplex_method class
- Tableau and simplex_method classes get instantiated only once throughout the runtime of the program

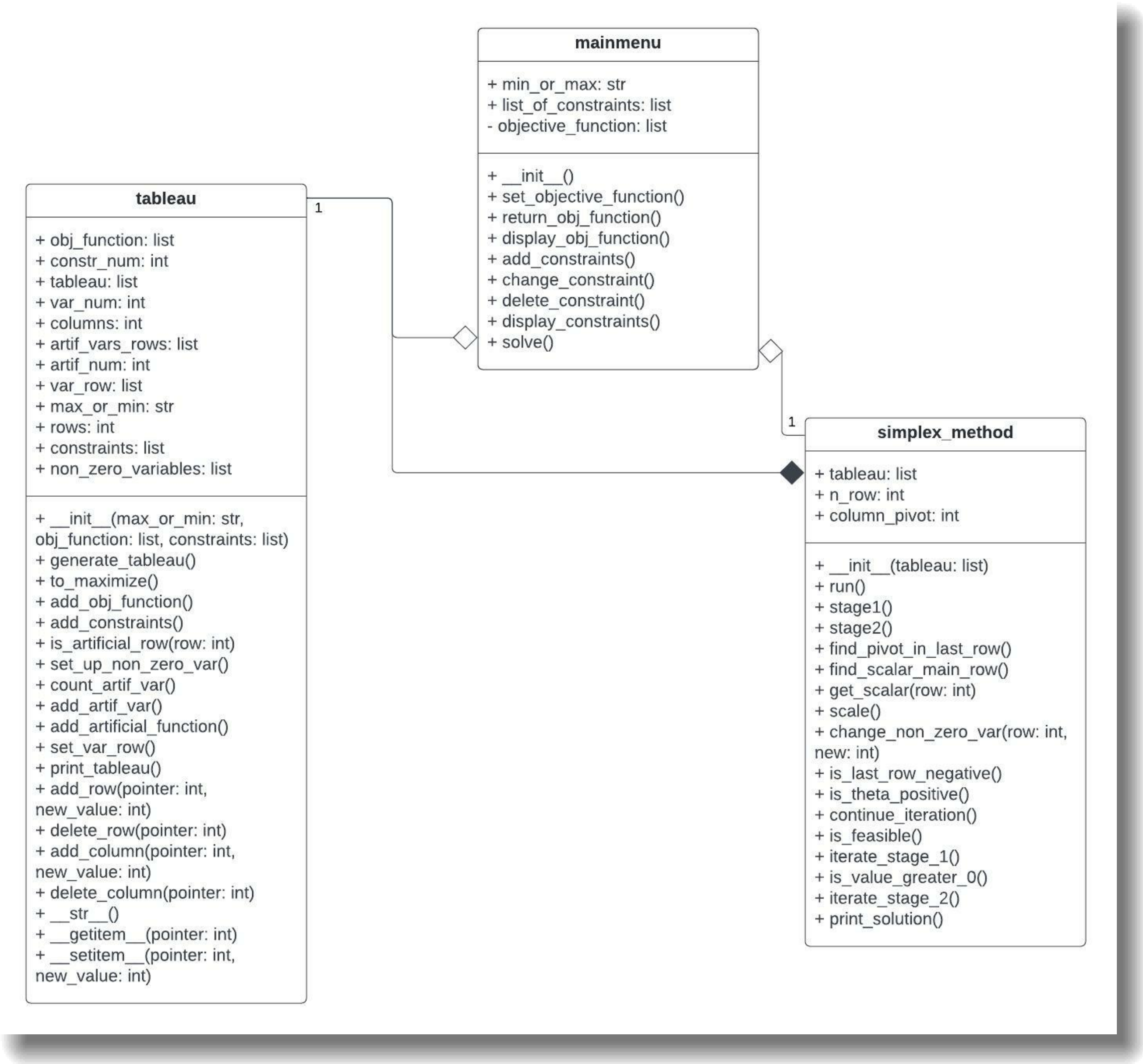
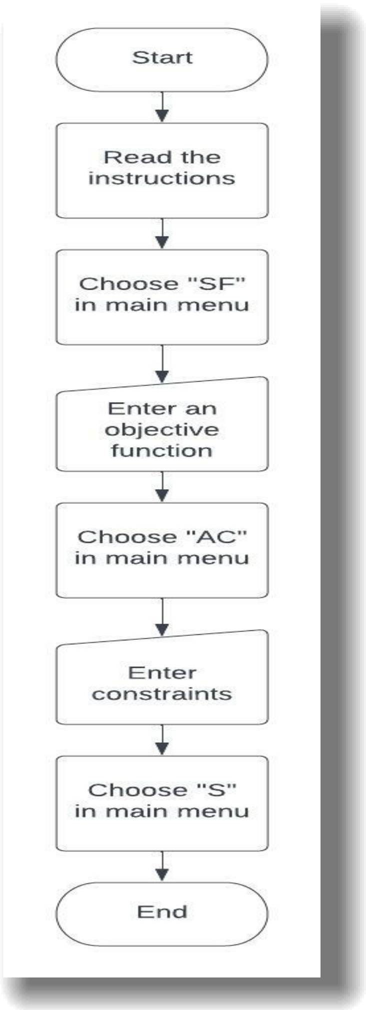


Diagram 1.2

Abstracted flow chart



Flowchart on the left displays the entire program and how it should be used. All inputs are called when the user chooses an option from the main menu. The user should be careful with the input of an objective function and the constraints and check it after choosing “S” or by using other menu options.

Diagram 1.3

User interface flow diagram and description of each process

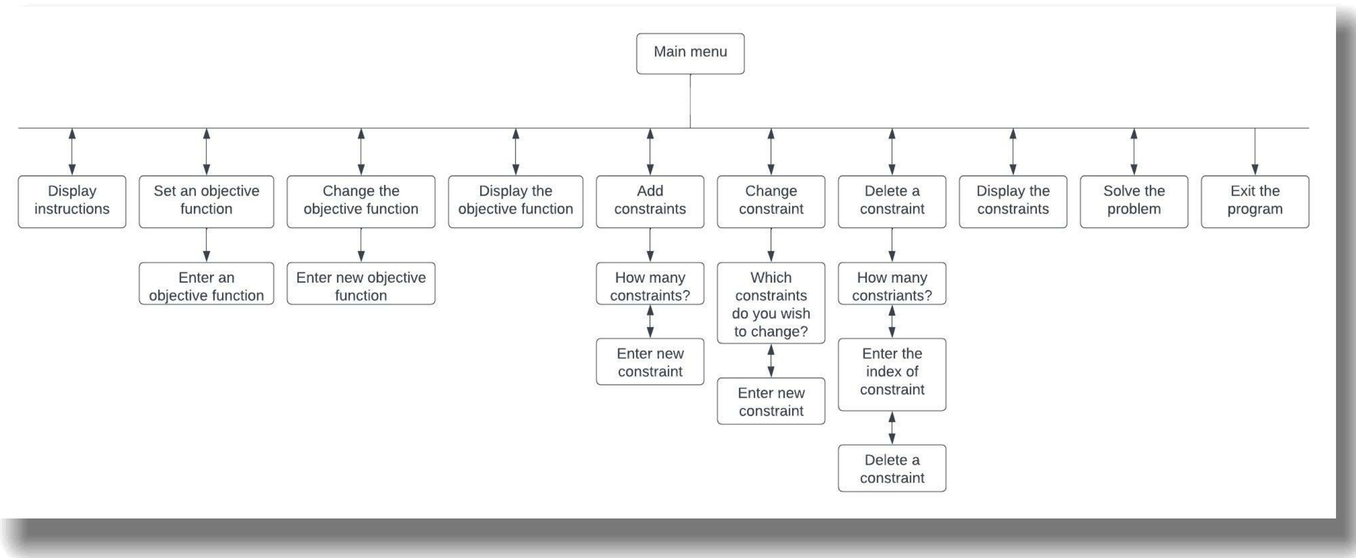


Diagram 1.4

As the program is run, a "welcome" message, the instructions for use of the simulation and the main menu are printed.

Mainmenu

The simulation is started by running menu () file. As the program runs, and "Enter menu option:" and main menu are printed. Main menu is created with a dictionary data structure with abbreviations as keys and explanations as values. The output in python console looks like this:

```
Enter menu option:

---Main Menu---

[I]   :  Display instructions
[SF]  :  set an objective function
[CF]  :  change the objective function
[DF]  :  display the objective function
[AC]  :  add constraints
[CC]  :  change a constraint
[DC]  :  delete a constraint
[SC]  :  display the constraints
[S]   :  solve the problem
[X]   :  exit the program
```

The user is then able to choose one of the options which is abbreviated for each option from the menu with key letters. User's input is checked with a while loop by looking for an input in main menu dictionary's keys:

```
choice = input()

while not choice.upper() in ['SF', 'SC', 'DF', 'DC', 'S', 'X', 'CF', 'I', 'CC', 'AC']:
    choice = input('Enter one of the instructions: \n')
```

Methods, that are called when a choice in main menu is made, entail a large use of defensive programming and error handling, in particular try…except, assertions and while loops.

Display instructions

This option prints out instructions with important information regarding the input of constraints and an objective function. This option is only called when the user forgets the requirements for input as the instructions are printed at the start of the execution.

Set up an objective function

Firstly, it will check whether the user has already set up an objective function. If not, then it will ask for optimization choice: either to maximize or minimize the value of objective function. Program checks if the user's input is valid with a while loop (choice between "max" or "min"). After, it asks to input the objective function. The input is then stripped and converted to a list of coefficients and inequality signs with a function strip(astring) from utils.py

If the user has already set up an objective function, the program will ask user to use another menu's option to change the objective function.

Change an objective function

When user chooses this option, the program will delete the old objective function and set a new one using the same method as "Set up an objective function" option.

Display the objective function

If the user has not set an objective function, it will ask him to choose a menu option to set one. Otherwise, it will print the objective function.

Add constraints

Program asks how many constraints to add and checks if the user's input is valid with a while loop using the function is_digit() from utils.py file. Once it is done, the user has to input constraints one by one. The input is then stripped and converted to a list of coefficients and inequality signs with a function strip(astring) from utils.py. The list is appended to the list of constraints.

Change a constraint

If this option is selected, the program checks if there are any constraints already set. If not, a prompt "There are no constraints. Use [AC] to add new constraints" appears. Otherwise, it will ask how many constraints the user wishes to change and checks if the user's input is valid with a while loop using the function is_digit() from utils.py file.

If the input is larger than the number of constraints already inputted, the program returns to the main menu and print a prompt "Number of constraints: *number*. Add new constraints using [AC] instruction".

If the inputs were valid, the program prints the constraints and asks which one to change. The input is checked as others previously. After this, the user is asked to input a new constraint. The input is then stripped and converted to a list of coefficients and inequality signs with a function strip(astring) from utils.py. The list is inserted at the place of an old constraint with the same index.

Delete a constraint

The program checks if there are any constraints already set. If not, a prompt "There are no constraints. Use [AC] to add new constraints" appears. Otherwise, it asks how many constraints the user wishes to delete and checks if the user's input is valid with a while loop using the function is_digit() from utils.py file.

If the input is larger than the number of constraints already inputted, the program iterates until the user puts in valid input.

If the inputs were valid, the program prints the constraints and asks which one to delete. The input is checked as others previously. After this, that constraint is deleted.

Display the constraints

If the user has not set any constraints, it will ask him to choose a menu option to set one. Otherwise, it will print the constraints.

Solve the problem

When user wants to solve the problem, the program checks if the objective function and constraints are inputted and ready to use.

If everything is valid, a prompt is printed and the program asks whether the objective function and constraints are correct and the user wishes to proceed to see the solution of the problem. Input is checked with a while loop to ensure it is either "yes" or "no".

If the input is yes, the program creates an instance of a tableau class with max_or_min, objective function, constraints parameters. The instance is passed as the argument to simplex_method class to create an instance of the class and work out the solution.

If the input is no, the program returns user back to main menu.

If the user is missing either an objective function or constraints, the program returns the user back to main menu.

Exit

This option is selected only in main menu and it shut down the execution.

Tableau

The major role of this class is to create a table, expressed as linked list (that looks like a matrix), with coefficients of variables to work with. Each list in the tableau linked list is either a function or a constraint with its coefficients of variables. Linked list is printed in the way so it looks like a 2x2 matrix as all lists inside must have the same length. When the project is instantiated in the mainmenu class, the data is passed to tableau's properties which are assigned to variables and then the program processes complex operations on them. As well as creating a tableau, it contains different methods in order to manipulate with the tableau later, such as add or delete row etc.

For instance, if the user's inputs are:

- Min_or_max = 'max'
- Objective function: -3x+1y-2z
- Constraints:
  - 5x+y0+1z<=16
  - 3x+1y+1z<=12
  - 1x-1y+4z>=9

The inputs for objective function and constraints are stripped with a function strip_a_string(astring) from utils.py and returned as a list of coefficients and inequality signs. After this operation, mainmenu fields look like this:

- Min_or_max = 'max'
- Objective function: [-3, 1, -2]
- List_of_constraints: [[5, 0, 1, '<=', 16], [3, 1, 1, '<=', 12], [1, -1, 4, '>=', 9]]

When the program creates an instance of tableau class, those variables are passed to tableau fields.

Before explaining how the tableau is generated, there might be some unknown terms:

- Slack variable is a variable that is added to an inequality constraint to transform it into an equality.
- A surplus variable is the difference between the total value of the true variables and the number on the right-hand side of the equation. Thus, surplus variable coefficient is negative.
- Artificial variable is a variable that is introduced in the linear program model to obtain the initial basic feasible solution.

First step of creating a nested list is to count the number of artificial variables. That is done by looking at inequality signs. If the constraint contains “>” or “>=”, then the number of artificial variables is incremented by 1. Then the coefficients of constraints are added to the lists. If the constraint contains ‘<’ or ‘<=’ slack variable is 1, surplus and artificial are 0. If the constraint contains greater than inequality signs, than surplus variable is -1, artificial – 1, slack – 0.

If there are artificial variables in the constraints, new function should be introduced for minimizing the sum of artificial variables. This function, if exists, is added to the linked list so it is printed on the bottom of the tableau.

After, an array of basic(non-zero) variables is created by finding out if there is an artificial variable in the constraint. Length of the array is the same as the number of constraints. If yes, then artificial variable is inserted into the array with the index equal to the index of that constraint in the linked list. If not, slack variable is inserted.

After manipulating the linked list, the tableau and list of basic variables look this way:

```
['x', 'y', 'z', 's1', 's2', 's3', 'a1', 'Value']
[5.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 16.0]
[3.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 12.0]
[1.0, -1.0, 4.0, 0.0, 0.0, -1.0, 1.0, 9.0]
[-3.0, 1.0, -2.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[-1.0, 1.0, -4.0, 0.0, 0.0, 1.0, 0.0, -9.0]

[s1, s2, a1]
```

Further description how the tableau is created will be seen in Technical Solution.

Simplex algorithm

Explanation and design

As much as the name of the project could tell you, my simulation is based on solving linear programming problems. As mentioned in the Identification of the problem, it is much more preferable because it may involve more than 2 variables.

Often, constraints involve only "=", "<" or "<=" inequality signs which indicates that method starts at the origin in the feasible region and uses slack variables as basic variables to give a basic feasible solution. The solution is then improved upon moving systematically to other vertices of the feasible region until the optimum solution is found. Here, a basic simplex method can find the solution.

However, it all becomes more complex when the constraints contain ">" or ">=" as problems with such constraints have no obvious basic feasible solution since the origin is not in the feasible region. If this is the case, 2 stage simplex method is introduced.

There is also a difference in maximizing and minimizing objective function value. If the purpose is to minimize the value (e.g. cost), objective function is multiplied by (-1) and after all iterations the return value is multiplied by (-1) to get minimum value for the objective function. The purpose of doing this can be explained by example. If the user wants to minimize {P}, it is the same as to maximize {-P}.

The whole process can be designed with a flow chart shown below.

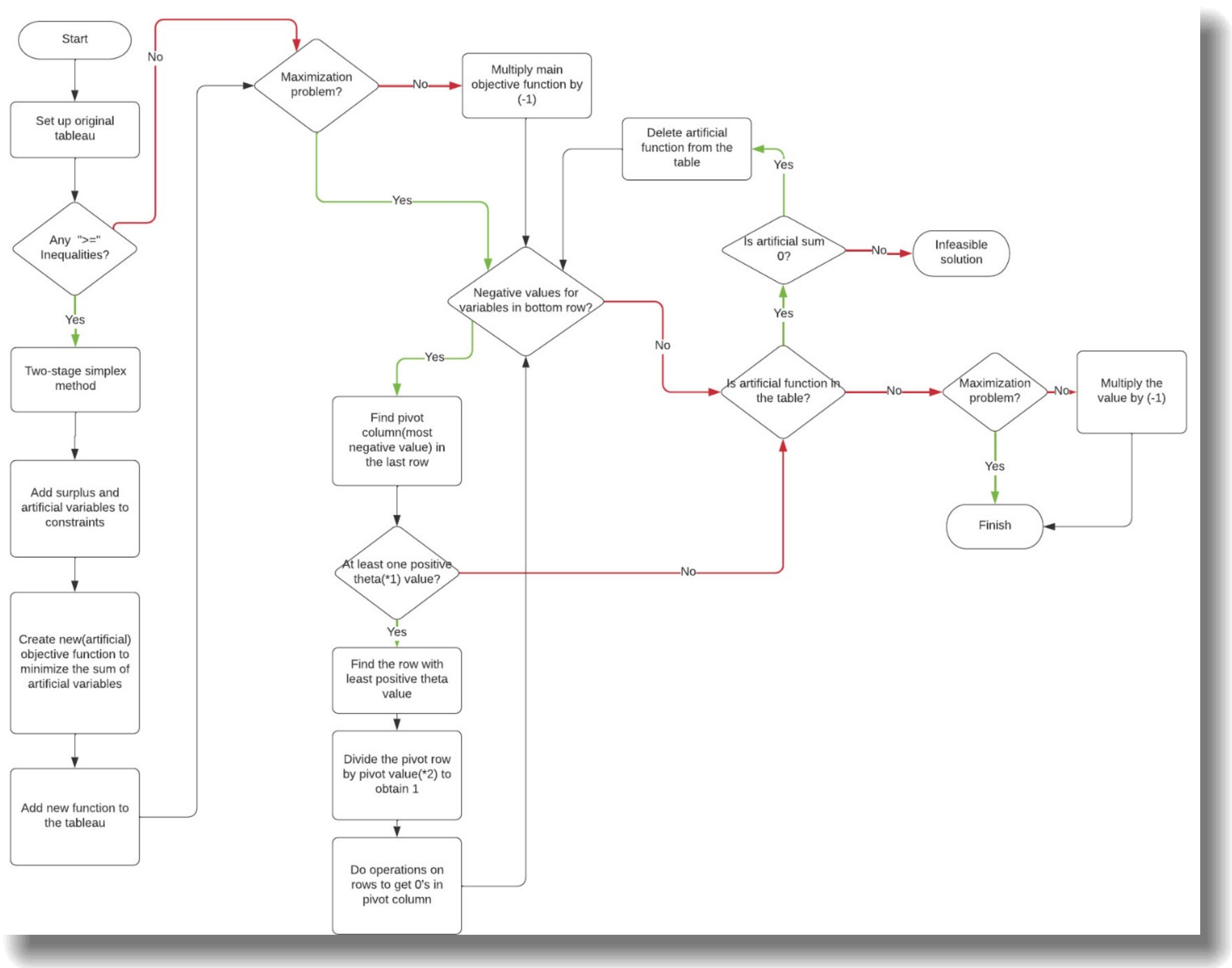


Diagram 1.5

A few key words need explanation:

- Theta value*1 - ratio between last and pivot column values in a row
- Pivot value*2 - pivot column element in a row with least positive theta
- Slack variable – variable used with “<=” constraints. Usually represented as {s}
- Surplus{s} and artificial{a} variables – variables introduced during two-stage simplex method
- Infeasible solution – no solution for the problem. Is either does not exist or a method other than simplex should be used

Implementation

My simplex method simulation works on the principal as introduced in the *Diagram 1.5*. However, my program works with the 2-D list and its coefficients without any other components. For example, inequality sign “>=” is introduced as “-1” for surplus variable and “1” for artificial variable in the tableau; however, inequality sign “<” is introduced as “1” for slack variable.

Otherwise, it follows the flow chart from start to end and works with numbers in the tableau.

After researching about two-stage simplex method on internet, I found different occasions that I have not covered in Edexcel Further Mathematics Decision 1 syllabus. However, I tackled special cases like degeneracy, unbounded and nonexistent solutions.

Feasibility study and justification of the solution

The application of UI system and the proposed architecture can guarantee that the project is feasible. Its purpose is to be able to solve linear programming problems with the input of an objective function and constraints. A heavy use of Object Orientated Programming (OOP) is an extremely positive aspect of the project because it allows to split classes into different files as well as OOP is considered to be a simple and straightforward programming paradigm. In-depth planning of the architecture allows me to easily debug and tackle errors.

Additionally, the project does not require an advanced hardware. The minimum requirement for the hardware is to be able to run high level language and an interpreter. The program does not take up a lot of space which allows to use it in different circumstances.

Last of all, if I ever get stuck with the syntax of python or unresolved error in simplex method, I can always refer to online resources to help me tackle the issue.

Technical solution

Solving a problem using only 1 stage

Iteration involving only 1 stage iteration is a basic simplex method iteration. The steps needed to be taken in order to solve a problem using this method is shown in *Diagram 1.5* in Documented Design. Reminder: only 1 stage is required when all of the constraints contain only “<” or “<=” inequality signs; hence, only slack variables are introduced.

The iteration starts in the feasible region. It starts with checking if we can start/continue an iteration

```
while self.continue_iteration() == True:
    self.iterate_stage_2()
```

Insert 2.1

Which consists of two requirements:

```
def continue_iteration(self):
    """
    Checks whether the program can continue the iteration by 2 factors:
        There are still negative values for variables in the last row
        There is still at least one positive theta value
    :return: True if it can continue. False otherwise
    """
    continue_iter = False
    if self.is_last_row_negative() == True:
        if self.is_theta_positive() == True:
            continue_iter = True
        else:
            print('The solution is unbounded. Another method is required for finding the optimal solution.')
            exit()
    return continue_iter
```

Insert 2.2

Function above covers one of the special cases when the program can iterate infinitely many times. Hence, if is_theta_positive() is False, the solution is unbounded and the program halts.

```
def is_last_row_negative(self):
    """
    Checks if there are negative values in the last row except for 'Value' column
    :return: True if there are negative values in the last row except for 'Value' column. False otherwise
    """
    for i in range(len(self.tableau[self.tableau.rows-1])-1):
        if self.tableau[self.tableau.rows-1][i] < 0:
            return True
    return False
```

Insert 2.3

```
def is_theta_positive(self):
    """
    Checks if there are negative values in the right column
    Implemented with defensive programming when there is division by 0 when looking for theta value
    :return: True if there are negative values in the right column
    """
    for i in range(self.tableau.constr_num):
        try:
            if (self.tableau[i][-1]) / (self.tableau[i][self.find_pivot_in_last_row()]) > 0:
                return True
        except ZeroDivisionError:
            i += 1
    return False
```

Insert 2.4

Note: defensive programming and is used in is_theta_positive() function to avoid crash of the program.

If both functions return True, program calls to iterate_stage_2(). See *Insert 2.1*

```
def iterate_stage_2(self):
    """
    Iteration for stage 2
    Scales the tableau until there are no negative values in the bottom row except for 'Value' column
    """
    self.scale()
```

Insert 2.5

```
def scale(self):
    """
    Scales the tableau
    In a row with least positive theta value each element is divided by the scalar to obtain 1 at index of pivot column
    Other rows are manipulated to get 0 at index of pivot column
    So that the column with pivot value would look like(assuming there are 3 constraints and 1 function and pivot row is random):
    [0]
    [0]
    [1]
    [0]
    """
    scalar = self.find_scalar_main_row()

    for i in range(len(self.tableau[self.n_row])):
        if self.tableau[self.n_row][i] != 0.0:
            self.tableau[self.n_row][i] = self.tableau[self.n_row][i] / scalar

    for i in range(self.tableau.rows):
        if self.tableau[i] != self.tableau[self.n_row]:
            temp = self.get_scalar(i)
            for j in range(len(self.tableau[i])):
                self.tableau[i][j] = self.tableau[i][j] + temp * (-1) * self.tableau[self.n_row][j]

    self.change_non_zero_var(self.n_row,self.tableau.var_row[self.column_pivot])

    print(self.tableau)
```

Insert 2.6

When scale() is called, find_scalar_main_row() function is called first.

```
def find_scalar_main_row(self):
    """
    Finds a value that should divide each element in the row with least positive theta value to obtain 1 at the pivot by doing the following:
    Iterating through every constraint row to find the least theta value(ratio between last element and column_pivot values in the row)
    There is a special case of degeneracy: when the value of basic variable is 0
    If so, theta value is 0 and the row with such case becomes a main row for the next iteration
    :return: Value that should divide each element in the row with least positive theta value to obtain 1 at the pivot
    """
    self.n_row = 0
    self.column_pivot = self.find_pivot_in_last_row()
    theta_value = float('inf')

    for i in range(self.tableau.constr_num):
        #checking for degeneracy
        if self.tableau[i][-1] == 0:
            print("Possible degeneracy, see next iterations \n")
            theta_value = 0
            self.n_row = i
        elif self.tableau[i][self.column_pivot] and self.tableau[i][-1] > 0:
            temp = self.tableau[i][-1] / self.tableau[i][self.column_pivot]
            if temp > 0 and temp < theta_value:
                theta_value = temp
                self.n_row = i

    return self.tableau[self.n_row][self.column_pivot]
```

Insert 2.7

Following it, find_pivot_in_last_row() is called.

```python
def find_pivot_in_last_row(self):
    """
    Finds the most negative value(pivot column) in the bottom
    If there are no negative values in bottom row during stage 1 and sum of artificial vars is not 0:
        No feasible solution
    :return: Index of most negative value in the bottom row of the tableau
    """
    cost_function = self.tableau.rows - 1
    temp = 0
    for i in range(0, self.tableau.columns - 1):
        if self.tableau[cost_function][i] < temp and self.tableau[cost_function][i] < 0:
            temp = self.tableau[cost_function][i]

    if temp == 0:
        print('No feasible solution')
        exit()
    else:
        return self.tableau[cost_function].index(temp)
```

Insert 2.8

After a value is assigned to “scalar” variable (*Insert 2.6*), a few operations are performed and get_scalar() is called which returns a value that is assigned to variable “temp”.

```python
def get_scalar(self,row):
    """
    Finds a scalar in a 'row' with an index of column_pivot variable
    :param row: A row to operate on
    :return: Returns a scalar that is used in scaling the tableau
    """
    return self.tableau[row][self.column_pivot]
```

Insert 2.9

After scaling the 2-D linked list, the program calls a change_non_zero_var (n_row, tableau.var_row[column_pivot]) to change the basic variables in the list.

```
def change_non_zero_var(self,row,new):
    """
    Changes a non-zero variable in a list
    :param row: index in a list
    :param new: new non-zero variable
    """
    self.tableau.non_zero_variables[row] = new
```

Insert 2.10

After running all of preceding functions the program prints out a new tableau to the console after 1 whole iteration. Then the process starts all over again starting from *Insert 2.1*

Solving a problem using 2 stages

2 stages are required when constraints contain ">" or ">=" inequality signs and surplus and artificial variables are introduced. Solving a linear programming problem this way is not harder but requires more iterations in order to get a solution. In such cases, an artificial objective function is introduced and added to the tableau.

```
def add_artificial_function(self):
    """
    Adds artificial function's coefficients to the tableau by doing the following:
    New obj function is I = (-) * sum of artif vars
    Manipulates rows of the tableau to get I
    """
    if self.artif_num > 0:

        self.add_row(self.rows, 0.0)

        for row in self.artif_vars_rows:
            for i in range(len(self.tableau[row])):
                self.tableau[-1][i] = self.tableau[-1][i] + self.tableau[row][i]

        for i in range(len(self.tableau[-1])):
            if self.tableau[-1][i] != -0.0:
                self.tableau[-1][i] = (-1) * self.tableau[-1][i]

        pointer = self.var_num + self.constr_num

        for i in range(self.artif_num):
            self.tableau[-1][pointer] = 0.0
            pointer += 1
```

Insert 3.1

Firstly, assuming there are artificial variables, continue_iteration() called and a value returned.(See *Insert 2.2-2.4* for reference)

```python
if self.tableau.artif_num > 0:

    while self.continue_iteration() == True:
        self.iterate_stage_1()
        if self.is_feasible():
            done = True
        else:
            done = False
    else:
        done = True

    self.tableau.delete_row(-1)

    for i in range(self.tableau.artif_num):
        self.tableau.delete_column(self.tableau.columns - 2)
    self.tableau.artif_num = 0
else:
    done = True

print('---End of stage 1---' + '\n')

return done
```

Insert 3.2

After, iterate_stage_1() is called where is_feasible() is called to check if the sum of the artificial variables is 0.

```python
def iterate_stage_1(self):
    """
    Iterations for stage 1
    Scales the tableau until the sum of artificial variables is 0
    """
    while not self.is_feasible():
        self.scale()
```

Insert 3.3

```python
def is_feasible(self):
    """
    Checks if the sum of artificial variables(value in the bottom right corner) is 0
    :return: True if the sum of artificial variables is 0. False otherwise
    """
    return self.tableau[self.tableau.rows-1][self.tableau.columns-1] == 0
```

Insert 3.4

While the sum is not 0, scale() is called which performs operations on the tableau(*See Insert 2.6-2.10*)

If the sum is 0 then stage 1 ends and program moves to stage 2, which is a basic simplex method described previously.

Setting up the tableau

The tableau is designed as a linked list that looks like a matrix because of its symmetry. Number of rows is dependent on the number of constraints, and type of variables. If there are no artificial variables, number of rows is equal to number of constraints + 1(objective function); however, with artificial variables number of rows is incremented by 1 because of introducing new function.

Firstly, the program counts the number of artificial variables. This allows the program to change the number of rows and columns in the tableau.

```
def count_artif_var(self):
    """
    Counts the number of artificial variables
    :return:  number of artificial variables
    """
    for i in range(self.constr_num):
        for j in self.constraints[i]:
            if j in utils.pos_signs:
                self.artif_num += 1
                self.artif_vars_rows.append(i)

    self.columns += self.artif_num

    return self.artif_num
```

Insert 4.1

Secondly, the constraints are added to the tableau.

```
def add_constraints(self):
    """
    Adds constraints' coefficients to the tableau by doing the following:
    Adds new rows to the tableau
    Inserts constraints' coefficients to each row
    """
    for row in range(self.constr_num):
        self.tableau.append([])

        for column in range(self.var_num):
            self.tableau[row].append(float(self.constraints[row][column]))

        for i in range(self.constr_num+self.artif_num):
            self.tableau[row].append(0.0)

        self.tableau[row].append(float(self.constraints[row][-1]))
```

Insert 4.2

Then, the values for slack, surplus and artificial variables are added to the table.

```python
def add_artif_var(self):
    """
    Adds artificial variables to the tableau by doing the following:
    if there is a artificial variable in a row:
        assign -1 to surplus var
        assign 1 to artificial var
    """
    for row in range(self.constr_num):
            if self.constraints[row][self.var_num] in utils.neg_signs:
                self.tableau[row][self.var_num + row] = 1.0

    if self.artif_num > 0:

        for row in range(self.constr_num):
            if self.constraints[row][self.var_num] in utils.pos_signs:
                self.tableau[row][self.var_num + row] = -1.0

        pointer = self.var_num + self.constr_num
        for i in self.artif_vars_rows:
            self.tableau[i][pointer] = 1.0
            pointer += 1
```

Insert 4.3

Depending on artificial variables, artificial function may be added to the tableau.

```python
def add_obj_function(self):
    """
    Adds objective function coefficients to the tableau by doing the following:
    Creates a new row with 0.0 as every element in a list
    Inserts coefficients of the objective function
    Coefficients are multiplied by (-1) if it is minimization problem
    """

    self.add_row(self.rows, 0.0)
    if self.to_maximize() == True:
        for i in range(self.var_num):
            self.tableau[-1][i] = float(self.obj_function[i])
    else:
        for i in range(self.var_num):
            self.tableau[-1][i] = (-1) * float(self.obj_function[i])
```

Insert 4.4

Finally, a list of basic variables is created by calling set_up_non_zero_var() function that uses is_artificial_row(row) function to check for artificial variables.

```python
def set_up_non_zero_var(self):
    """
    Sets up the first list of non-zero(basic) variables by doing the following:
    Creates a list of basic variables with 0.0 as every element
    Checks each row for a artificial variable
    If exists:
        Insert it to the list
    else:
        Insert the surplus var to the list
    :return: a list of non-zero variables
    """

    temp = []

    for i in range(self.constr_num):
        temp.append(0.0)

    try:
        for row in range(self.artif_num):
            if self.is_artificial_row(self.artif_vars_rows[row]):
                temp[self.artif_vars_rows[row]] = self.set_var_row()[self.var_num + self.constr_num + row]
    except IndexError:
        print('No artificial variables')

    for i in range(len(temp)):
        if type(temp[i]) == float:
            temp[i] = self.set_var_row()[self.var_num + i]

    return temp
```

Insert 4.5

Error handling is used to prevent program failure.

```python
def is_artificial_row(self, row):
    """
    checks if the constraint has an artificial variable
    :param row: a constraint in coefficient form
    :return: True if there is an artificial variable. False otherwise.
    """
    return row in self.artif_vars_rows
```

Insert 4.6

User Interface

```python
import utils
from table import tableau
from simplex import simplex_method

class mainmenu(object):

    def __init__(self):
        """
        Constructor of mainmenu class.
        Creates variables for objective function, list of constraints, minimize or maximize input
        """
        super(mainmenu, self).__init__()
        self.__objective_function = None
        self.list_of_constraints = []
        self.min_or_max = ''

    def set_objective_function(self):
        """
        Sets an objective function by:
        User input is stripped into a list of coefficients and (in)equality signs and assigned to the obj_function variable
        """
        self.__objective_function = utils.strip_a_string(input('Enter your objective function: \n'))

    def return_obj_function(self):
        """
        Returns objective function
        :return: objective function
        """
        return self.__objective_function

    def display_obj_function(self):
        print("Your objective function is " + str(self.return_obj_function()))

    def add_constraints(self):
        """
        Adds constraints to the list of constraints
        Program will not until the user enters a valid input
        """
        try:
            constr_num = input('How many constraints: ')
            assert utils.is_digit(constr_num), 'Input has to be a digit in [0-9]'
            for i in range(int(constr_num)):
                constraint_input = utils.strip_a_string(input('Input your constraint #%i: ' % (i + 1)))
                self.list_of_constraints.append(constraint_input)
        except AssertionError as msg:
            print(msg)

    def change_constraint(self):
        """
        Changes constraints in the list of constraints
        """
        if len(self.list_of_constraints) == 0:
            print('There are no constraints. Use [AC] to add new constraints.')
            pass
        else:
            try:
                how_many_to_change = input('How many constraints do you wish to change? ')
                assert utils.is_digit(how_many_to_change), 'Input has to be a digit in between 1 and %i. Try again. \n' %len(self.list_of_constraints)
                if int(how_many_to_change) > len(self.list_of_constraints):
                    print("Number of constraints: %i. Add new constraints using [AC] instruction" %len(self.list_of_constraints))
```

Page 1 of 5

```
                    pass
                else:
                    for i in range(int(how_many_to_change)):
                        self.display_constraints()
                        index_of_constraint = input('Which constraint do you wish to change? \n')
                        assert utils.is_digit(index_of_constraint), 'Input has to be a digit in between 1 and %i. Try again. \n' %len(self.list_of_constraints)
                        while int(index_of_constraint) > len(self.list_of_constraints):
                            print('Index of constraint is out of range. There are only %i constraints.' %len(self.list_of_constraints))
                            index_of_constraint = input('Out of %i constraints, which one do you wish to change? ' %len(self.list_of_constraints))
                            while not utils.is_digit(index_of_constraint):
                                index_of_constraint = input('Input has to be a digit between 1 and %i. Try again: \n' % len(self.list_of_constraints))
                        new_constraint = utils.strip_a_string(input('Input your new constraint: \n '))
                        self.list_of_constraints[int(index_of_constraint)-1] = new_constraint
            except AssertionError as msg:
                print(msg)

    def delete_constraint(self):
        """
        Deletes a constraint from the list of constraints
        """
        if len(self.list_of_constraints) == 0:
            print('There are no constraints. Use [SC] to add new constraints.')
            pass
        else:
            try:
                how_many_to_delete = input("How many constraints you wish to delete? \n ")
                assert utils.is_digit(how_many_to_delete), 'Input has to be a digit in between 1 and %i. Try again. \n' %len(self.list_of_constraints)

                while int(how_many_to_delete) > len(self.list_of_constraints):
                    how_many_to_delete = input('Input has to be a digit between 1 and %i. Try again: \n' % len(self.list_of_constraints))

                for i in range(int(how_many_to_delete)):
                    self.display_constraints()
                    constraint_index = input('Which constraint do you wish to delete? \n ')
                    while not utils.is_digit(constraint_index):
                        constraint_index = input('Input has to be a digit between 1 and %i. Try again: \n' % len(self.list_of_constraints))
                    while int(constraint_index) > len(self.list_of_constraints):
                        constraint_index = input('Input has to be a digit between 1 and %i. Try again: \n' % len(self.list_of_constraints))
                    self.list_of_constraints.pop(int(constraint_index)-1)
            except AssertionError as msg:
                print(msg)

    def display_constraints(self):
        """
        Displays constraints
        """
        if len(self.list_of_constraints) == 0:
            print("There are no constraints. Please, use [SC] instruction to add new constraints.")
            pass
        else:
            adictionary = {}
            count = 0
```

Page 2 of 5

```
            for i in self.list_of_constraints:
                count += 1
                x = 'constraint #%s: ' %count
                adictionary[x] = i
            utils.display_dictionary(adictionary,'---Constraints---')

    def solve(self):
        """
        Solves the problem using simplex method
        Creates an instance of tableau class and passing variables below to tableau's arguments
        Creates an instance of simplex_method class with instance of the tableau passed as an argument
        After the run of the simulation, the program shuts down
        """
        new_table = tableau(self.min_or_max, self.__objective_function,self.list_of_constraints)
        new_simplex = simplex_method(new_table)
        quit()

def mainmenudict(object):
    """
    Displays main menu dictionary and lets user interact with it
    The program will return back to main menu in most cases when the input is invalid
    It is done by defensive programming techniques and exception handling(see below for evidence)
    After every valid input from the user, a prompt will be printed to show the user the instructions
    It only halts when the user chooses "X" as his input in the main menu. After that, the program instantly shuts down
    :param object: instance of main menu class
    :return: Choice made by the user
    """
    mainmenu_dictionary = \
    {'[I]   : ': 'Display instructions',
        '[SF]  : ': 'set an objective function',
        '[CF]  : ': 'change the objective function',
        '[DF]  : ': 'display the objective function',
        '[AC]  : ': 'add constraints',
        '[CC]  : ': 'change a constraint',
        '[DC]  : ': 'delete a constraint',
        '[SC]  : ': 'display the constraints',
        '[S]   : ': 'solve the problem',
        '[X]   : ': 'exit the program',}

    utils.display_dictionary(mainmenu_dictionary, '---Main Menu--- \n')

    choice = input()

    while not choice.upper() in ['SF', 'SC', 'DF', 'DC', 'S', 'X', 'CF', 'I', 'CC', 'AC']:
        choice = input('Enter one of the instructions: \n')

    choice = choice.upper()

    if choice == 'I':
        print(utils.instructions)
        pass

    elif choice == 'SF':
        if object.return_obj_function() == None:
            max_or_min = ['max','min']
            try:
                object.min_or_max = input('Do you wish to maximize or minimize? Enter "max" or "min": \n')
                assert  object.min_or_max in max_or_min, 'Wrong input. Please, enter "max" or "min". \n'
```

```
                print(object.min_or_max)
                object.set_objective_function()
            except AssertionError as msg:
                print(msg)
        else:
            print('You have already set an objective function. If you wish to change it, use instruction [CF] in the main menu. ')

    elif choice == 'CF':
        try:
            cf_input = input('Do you really wish to change objective function?(Yes/No) \n')
            assert cf_input in ['yes','no'], 'Wrong input. Please, enter yes/no. \n'
            if cf_input.lower() == 'yes':
                if object.return_obj_function() == None:
                    max_or_min = ['max', 'min']
                    object.min_or_max = input('Do you wish to maximize or minimize? Enter "max" or "min": \n')
                    assert object.min_or_max in max_or_min, 'Wrong input. Please, enter "max" or "min". \n'
                    object.set_objective_function()
                object.set_objective_function()
            else:
                pass
        except AssertionError as msg:
            print(msg)

    elif choice == 'DF':
        if object.return_obj_function() == None:
            print("You have not set an objective function. Use [SF] to set one ")
        else:
            print()
            print('Your objective function is: ' + str(object.return_obj_function()))
            print()

    elif choice == 'AC':
        object.add_constraints()

    elif choice == 'CC':
        object.change_constraint()

    elif choice == 'DC':
        object.delete_constraint()

    elif choice == 'SC':
        object.display_constraints()

    elif choice == 'X':
        print('Thank you very much for using my program. See you! ')

    elif choice == 'S':
        if object.return_obj_function() == None:
            print("You have not set an objective function.")
        else:
            object.display_obj_function()

        if len(object.list_of_constraints) == 0:
            print("You have not set constraints yet.")
        else:
            object.display_constraints()
        input_choice = input("Is the input for your objective function and constraints correct? (Yes/No) \n")

        while input_choice.lower() not in ['yes','no']:
            input_choice = input("Wrong input. Please, enter 'yes' or 'no'. \n ")

        input_choice = input_choice.lower()
```

Page 4 of 5

```

        if input_choice == 'yes' and (object.return_obj_function() == None or len(object.list_of_constraints) == 0):
            print("\nCannot use simplex method without setting the objective function or constraints. Please, try again... ")
        elif input_choice == 'yes':
            print(object.solve())
        elif input_choice == 'no':
            print('\n Getting back to the main menu...')

    return choice.upper()

myfunct = mainmenu()

if __name__ == '__main__':
    print(utils.intro_text)
    print(utils.instructions)

    while not mainmenudict(myfunct) == 'X':
        pass

```

Tableau

```
import utils

class tableau(object):

    def __init__(self, max_or_min, obj_function, constraints):
        """
        Constructor if tableau class
        :param max_or_min: a parameter that contains either 'max' or 'min'
        :param obj_function: objective function
        :param constraints: list of constraints
        """
        super(tableau, self).__init__()
        self.max_or_min = max_or_min
        self.obj_function = obj_function
        self.constraints = constraints
        self.generate_tableau()

    def generate_tableau(self):
        """
        Creates a tableau from the objective function and constraints.
        Creates a matrix m*n
        """
        self.tableau = []

        self.var_num = len(self.obj_function)
        self.constr_num = len(self.constraints)

        self.artif_vars_rows = []
        self.artif_num = 0



        self.columns = self.var_num + self.constr_num + 1
        self.rows = self.constr_num

        self.count_artif_var()
        self.add_constraints()
        self.add_artif_var()
        self.add_obj_function()
        self.add_artificial_function()
        self.non_zero_variables = self.set_up_non_zero_var()

    def to_maximize(self):
        """
        Checks if parameter max_or_min is 'max'
        :return: True if parameter is 'max'. False otherwise
        """
        return self.max_or_min == 'max'

    def add_obj_function(self):
        """
        Adds objective function coefficients to the tableau by doing the following:
        Creates a new row with 0.0 as every element in a list
        Inserts coefficients of the objective function
        Coefficients are multiplied by (-1) if it is minimization problem
        """

        self.add_row(self.rows, 0.0)
        if self.to_maximize() == True:
            for i in range(self.var_num):
                self.tableau[-1][i] = float(self.obj_function[i])
        else:
            for i in range(self.var_num):
                self.tableau[-1][i] = (-1) * float(self.obj_function[i])

    def add_constraints(self):
        """
```

```
        Adds constraints' coefficients to the tableau by doing the following:
        Adds new rows to the tableau
        Inserts constraints' coefficients to each row
        """
        for row in range(self.constr_num):
            self.tableau.append([])

            for column in range(self.var_num):
                self.tableau[row].append(float(self.constraints[row][column]))

            for i in range(self.constr_num+self.artif_num):
                self.tableau[row].append(0.0)

            self.tableau[row].append(float(self.constraints[row][-1]))

    def is_artificial_row(self, row):
        """
        checks if the constraint has an artificial variable
        :param row: a constraint in coefficient form
        :return: True if there is an artificial variable. False otherwise.
        """
        return row in self.artif_vars_rows

    def set_up_non_zero_var(self):
        """
        Sets up the first list of non-zero(basic) variables by doing the following:
        Creates a list of basic variables with 0.0 as every element
        Checks each row for a artificial variable
        If exists:
            Insert it to the list
        else:
            Insert the surplus var to the list
        :return: a list of non-zero variables
        """

        temp = []

        for i in range(self.constr_num):
            temp.append(0.0)

        try:
            for row in range(self.artif_num):
                if self.is_artificial_row(self.artif_vars_rows[row]):
                    temp[self.artif_vars_rows[row]] = self.set_var_row()[self.var_num + self.constr_num + row]
        except IndexError:
            print('No artificial variables')

        for i in range(len(temp)):
            if type(temp[i]) == float:
                temp[i] = self.set_var_row()[self.var_num + i]

        return temp

    def count_artif_var(self):
        """
        Counts the number of artificial variables
        :return:  number of artificial variables
        """
        for i in range(self.constr_num):
            for j in self.constraints[i]:
                if j in utils.pos_signs:
                    self.artif_num += 1
                    self.artif_vars_rows.append(i)

        self.columns += self.artif_num
```

```
        return self.artif_num

    def add_artif_var(self):
        """
        Adds artificial variables to the tableau by doing the following:
        if there is a artificial variable in a row:
            assign -1 to surplus var
            assign 1 to artificial var
        """
        for row in range(self.constr_num):
                if self.constraints[row][self.var_num] in utils.neg_signs:
                    self.tableau[row][self.var_num + row] = 1.0

        if self.artif_num > 0:

            for row in range(self.constr_num):
                if self.constraints[row][self.var_num] in utils.pos_signs:
                    self.tableau[row][self.var_num + row] = -1.0

            pointer = self.var_num + self.constr_num
            for i in self.artif_vars_rows:
                self.tableau[i][pointer] = 1.0
                pointer += 1

    def add_artificial_function(self):
        """
        Adds artificial function's coefficients to the tableau by doing the following:
        New obj function is I = (-) * sum of artif vars
        Manipulates rows of the tableau to get I
        """
        if self.artif_num > 0:

            self.add_row(self.rows, 0.0)

            for row in self.artif_vars_rows:
                for i in range(len(self.tableau[row])):
                    self.tableau[-1][i] = self.tableau[-1][i] + self.tableau[row][i]

            for i in range(len(self.tableau[-1])):
                if self.tableau[-1][i] != -0.0:
                    self.tableau[-1][i] = (-1) * self.tableau[-1][i]

            pointer = self.var_num + self.constr_num

            for i in range(self.artif_num):
                self.tableau[-1][pointer] = 0.0
                pointer += 1

    def set_var_row(self):
        """
        Prints variable row above the tableau
        :return: a list of variables
        """
        variables = ['x', 'y', 'z', 'm', 'n']

        self.var_row = []

        for i in range(self.columns):
            self.var_row.append(0.0)

        for i in range(self.var_num):
            self.var_row[i] = variables[i]

        for i in range(self.constr_num):
            self.var_row[self.var_num + i] = 's%i' % (i + 1)

        for i in range(self.artif_num):
```

```python
            self.var_row[self.constr_num + self.var_num + i] = 'a%i' % (i + 1)

        self.var_row[-1] = 'Value'

        return self.var_row

    def print_tableau(self):
        """
        Prints a tableau without the variables row
        :return: printed tableau
        """
        printed_tableau = ''
        printed_tableau += str(self.set_var_row()) + '\n'
        for i in range(len(self.tableau)):
            printed_tableau += str(self.tableau[i]) + '\n'

        return printed_tableau

    def add_row(self,pointer,new_value):
        """
        Adds a row to the tableau
        :param pointer: Index of the row
        :param new_value: New value for each element in row
        """
        self.tableau.insert(pointer, [])
        for i in range(self.columns):
            self.tableau[pointer].append(new_value)
        self.rows += 1

    def delete_row(self,pointer):
        """
        Deletes a row from the tableau
        :param pointer: Index of the row
        """
        self.rows -= 1
        self.tableau.pop(pointer)


    def add_column(self,pointer,new_value):
        """
        Adds a column to the tableau
        :param pointer: Index of the column
        :param new_value: New value for each element in the column
        """
        self.columns += 1
        for i in range(self.rows):
            self.tableau[i].insert(pointer,new_value)

    def delete_column(self,pointer):
        """
        Deletes a column from the tableau
        :param pointer: Index of the column
        """
        self.columns -= 1
        for i in range(self.rows):
            self.tableau[i].pop(pointer)

    def __str__(self):
        """
        Prints a tableau
        :return: a tableau in string type
        """
        return_string = ""

        return_string += self.print_tableau() + '\n'
        return_string += "Basic variables: " + str(self.non_zero_variables) + '\n'

```

```
        return return_string

    def __getitem__(self, pointer):
        """
        Returns an element from the tableau
        :param pointer: an index
        :return: item from the tableau
        """
        return self.tableau[pointer]

    def __setitem__(self, pointer, new_value):
        """
        Sets a value to the element at location specified by 'pointer' in the tableau
        :param pointer: an index
        :param new_value: new value for the element
        """
        self.tableau.insert(pointer, new_value)

```

Simplex_method

```python
from table import tableau

class simplex_method(object):

    def __init__(self, tableau):
        """
        Constructor for simplex_method class
        :param tableau: a tableau to do operations on
        """
        super(simplex_method, self).__init__()
        self.tableau = tableau

        self.run()

    def run(self):
        """
        Runs the simplex algorithm in two stages depending on the step-by-step result
        """
        if self.stage1() == True:
            if self.stage2() == True:
                self.print_solution()
            else:
                print("Infeasible solution")
        else:
            print("Infeasible solution")

        #self.print_solution()

    def stage1(self):
        """
        Stage 1 of simplex method
        Minimizes the sum of the artificial variables
        Iterates the tableau with artificial function as the last row
        If there are no negative values for variables in artificial function or no positive value for a single theta value:
            Checks the artificial sum
            If it is 0:
                Proceed to stage 2
            else:
                Infeasible region. No solutions exist for that problem

        :return: True if the sum of the artificial variables. False otherwise.
        """

        done = False

        print('---Start of stage 1---' + '\n')
        print(self.tableau)

        if self.tableau.artif_num > 0:

            while self.continue_iteration() == True:
                self.iterate_stage_1()
                if self.is_feasible():
                    done = True
                else:
                    done = False
            else:
                done = True

            self.tableau.delete_row(-1)

            for i in range(self.tableau.artif_num):
                self.tableau.delete_column(self.tableau.columns - 2)
            self.tableau.artif_num = 0
        else:
            done = True
```

Page 1 of 5

```python

        print('---End of stage 1---' + '\n')

        return done

    def stage2(self):
        """
        Stage 2 of simplex method
        Works only with constraints and original objective function if the sum of artificial variables is 0 or there are no artificial variables from the start
        Works as the basic simplex method
        See Diagram 1.5 in Documented Design to see how the method works
        Maximizes the value for objective function
        :return:
        """
        done = False

        print('---Start of stage 2---' + '\n')
        print(self.tableau)

        while self.continue_iteration() == True:
            self.iterate_stage_2()
        else:
            if self.is_value_greater_0() == False and self.tableau.max_or_min == 'max':
                done = False
            else:
                done = True
        print('---End of stage 2---' + '\n')

        return done

    def find_pivot_in_last_row(self):
        """
        Finds the most negative value(pivot column) in the bottom
        If there are no negative values in bottom row during stage 1 and sum of artificial vars is not 0:
            No feasible solution
        :return: Index of most negative value in the bottom row of the tableau
        """
        cost_function = self.tableau.rows - 1
        temp = 0
        for i in range(0, self.tableau.columns - 1):
            if self.tableau[cost_function][i] < temp and self.tableau[cost_function][i] < 0:
                temp = self.tableau[cost_function][i]

        if temp == 0:
            print('No feasible solution')
            exit()
        else:
            return self.tableau[cost_function].index(temp)

    def find_scalar_main_row(self):
        """
        Finds a value that should divide each element in the row with least positive theta value to obtain 1 at the pivot by doing the following:
        Iterating through every constraint row to find the least theta value(ratio between last element and column_pivot values in the row)
        There is a special case of degeneracy: when the value of basic variable is 0
        If so, theta value is 0 and the row with such case becomes a main row for the next iteration
        :return: Value that should divide each element in the row with least positive theta value to obtain 1 at the pivot
        """
        self.n_row = 0
        self.column_pivot = self.find_pivot_in_last_row()
```

Page 2 of 5

```
            elif self.tableau[i][self.column_pivot] and self.tableau[i][-1] > 0:
                temp = self.tableau[i][-1] / self.tableau[i][self.column_pivot]
                if temp > 0 and temp < theta_value:
                    theta_value = temp
                    self.n_row = i

        return self.tableau[self.n_row][self.column_pivot]

    def get_scalar(self,row):
        """
        Finds a scalar in a 'row' with an index of column_pivot variable
        :param row: A row to operate on
        :return: Returns a scalar that is used in scaling the tableau
        """
        return self.tableau[row][self.column_pivot]

    def scale(self):
        """
        Scales the tableau
        In a row with least positive theta value each element is divided by the scalar to obtain 1 at index of pivot column
        Other rows are manipulated to get 0 at index of pivot column
        So that the column with pivot value would look like(assuming there are 3 constraints and 1 function and pivot row is random):
        [0]
        [0]
        [1]
        [0]
        """
        scalar = self.find_scalar_main_row()

        for i in range(len(self.tableau[self.n_row])):
            if self.tableau[self.n_row][i] != 0.0:
                self.tableau[self.n_row][i] = self.tableau[self.n_row][i] / scalar

        for i in range(self.tableau.rows):
            if self.tableau[i] != self.tableau[self.n_row]:
                temp = self.get_scalar(i)
                for j in range(len(self.tableau[i])):
                    self.tableau[i][j] = self.tableau[i][j] + temp * (-1) * self.tableau[self.n_row][j]

        self.change_non_zero_var(self.n_row,self.tableau.var_row[self.column_pivot])

        print(self.tableau)

    def change_non_zero_var(self,row,new):
        """
        Changes a non-zero variable in a list
        :param row: index in a list
        :param new: new non-zero variable
        """
        self.tableau.non_zero_variables[row] = new

    def is_last_row_negative(self):
        """
        Checks if there are negative values in the last row except for 'Value' column
        :return: True if there are negative values in the last row except for 'Value' column. False otherwise
        """
        for i in range(len(self.tableau[self.tableau.rows-1])-1):
            if self.tableau[self.tableau.rows-1][i] < 0:
                return True
        return False

    def is_theta_positive(self):
        """
```

```python
        Checks if there are negative values in the right column
        Implemented with defensive programming when there is division by 0 when looking for theta value
        :return: True if there are negative values in the right column
        """
        for i in range(self.tableau.constr_num):
            try:
                if (self.tableau[i][-1]) / (self.tableau[i][self.find_pivot_in_last_row()]) > 0:
                    return True
            except ZeroDivisionError:
                i += 1
        return False

    def continue_iteration(self):
        """
        Checks whether the program can continue the iteration by 2 factors:
            There are still negative values for variables in the last row
            There is still at least one positive theta value
        :return: True if it can continue. False otherwise
        """
        continue_iter = False
        if self.is_last_row_negative() == True:
            if self.is_theta_positive() == True:
                continue_iter = True
            else:
                print('The solution is unbounded. Another method is required for finding the optimal solution.')
                exit()
        return continue_iter

    def is_feasible(self):
        """
        Checks if the sum of artificial variables(value in the bottom right corner) is 0
        :return: True if the sum of artificial variables is 0. False otherwise
        """
        return self.tableau[self.tableau.rows-1][self.tableau.columns-1] == 0

    def iterate_stage_1(self):
        """
        Iterations for stage 1
        Scales the tableau until the sum of artificial variables is 0
        """
        while not self.is_feasible():
            self.scale()

    def is_value_greater_0(self):
        """
        Checks if the value in the bottom right corner is greater than 0
        :return: True if the value in the bottom right corner is greater than 0. False otherwise
        """
        return self.tableau[self.tableau.rows-1][self.tableau.columns-1] > 0

    def iterate_stage_2(self):
        """
        Iteration for stage 2
        Scales the tableau until there are no negative values in the bottom row except for 'Value' column
        """
        self.scale()

    def print_solution(self):
        """
        Prints the value for the optimal solution if exists and for each of the variables(basic and non-basic)
```

```
        """
        if self.tableau.max_or_min == 'max':
            print('Optimal solution for maximization is ' + str(round(self.tableau[-1][-1],3)) + '\n')
        elif self.tableau.max_or_min == 'min':
            print('Optimal solution for minimization is ' + str((-1) * round(self.tableau[-1][-1],3)) + '\n')

        print('Values of variables: ')

        for i in range(len(self.tableau.non_zero_variables)):
            print(self.tableau.non_zero_variables[i] + ' : ' + str(round(self.tableau[i][-1],4)))

        for i in self.tableau.var_row[:-1]:
            if i not in self.tableau.non_zero_variables:
                print(i + ' : 0.0')

```

Utils

```python
class colours:
    white = '\033[97m'
    green = '\033[92m'
    yellow = '\033[93m'
    red = '\033[91m'
    reset = '\033[0m'

digits = ['0','1','2','3','4','5','6','7','8','9']
pos_signs = ['>', '>=']
neg_signs = ['<', '<=']
signs = ['<', '>','>=','<=', '=','-']
operands = ['+','-']
variables = ['x','y','z','m','n']
intro_text = ' \nWelcome to Simplex World! Using this program you can optimize your solution by using simplex method. Below you can see the main menu. \nBefore proceeding, reading the instructions is recommended for quick and easy use.\n '

instructions = f'{colours.green}      ---INSTRUCTIONS---{colours.reset}' \
               f'\n{colours.white}Only following variables are allowed in that order as an input: x, y, z, m, n.{colours.reset}' \
               f'\n{colours.white}To input an objective function, ignore the variable you are trying to maximize/minimize and input what goes after "=" sign with coefficients timed by (-1){colours.reset}' \
               f'\n{colours.yellow}  example: If you want to maximize P=3x-2y, your input must be "-3x+2y".{colours.reset}'\
               f'\n{colours.white}If one of the variables is in objective function but not in constraint, put 0 before the variable in the constraint. If the coefficient is 1 or -1, write 1/-1 before the variable.{colours.reset}' \
               f'\n{colours.yellow}  example: Objective function: "x+5y-3z". "y" is not in one of the constraints, so your input must be: "-4x+0y+1z>33".{colours.reset}' \
               f'\n{colours.red}Not following the instructions will be followed with a wrong result. {colours.reset}'

def is_digit(adigit):
    """
    Checks if the parameter is a digit
    :param adigit: input that is checked for type
    :return: True if the parameter is a digit. False otherwise.
    """
    return adigit in digits

def is_sign(achar):
    """
    Checks if the parameter is in signs list
    :param achar: input that is checked
    :return: True if the parameter is a sign specified in the list 'signs'. False otherwise.
    """
    return achar in signs

def is_operand(achar):
    """
    Checks if the parameter is '+' or '-'
    :param achar: input that is checked
    :return: True if the parameter is '+' or '-'. False otherwise
    """
    return achar in operands

def is_variable(achar):
    """
    Checks if the parameter is in variables list
    :param achar: input that is checked
    :return: True if the parameter is in variables list. False otherwise.
    """
    return achar in variables

def display_dictionary(dictionary, title):
```

Page 1 of 3

```
    """
    Displays a dictionary with its title
    :param dictionary: a dictionary
    :param title: a title
    """
    print('\nEnter menu option: \n')
    print(title)
    for key, value in dictionary.items():
        print(key, value)
    print()

def number_of_negatives(alist):
    """
    Counts the number of negative coefficients in 'alist'
    :param alist: a list to iterate through
    :return: the number of negative coefficients in 'alist'
    """
    neg_num = 0

    for i in alist:
        if i == '-':
            neg_num += 1

    return neg_num

def number_of_signs(alist):
    """
    Counts the number of signs in 'alist'
    :param alist: a list to iterate through
    :return: the number of signs in 'alist'
    """
    signs_num = 0
    ineq_signs = ['>','<','=']
    for i in alist:
        if i in ineq_signs:
            signs_num += 1

    return signs_num

def strip_a_string(astring):
    """
    Strips a string and returns the coefficients of variables and inequality signs
    :param astring: a string to strip
    :return: list of coefficients and inequality signs
    """
    #checks if the user inputted blank spaces by mistake and deletes them if True

    for i in astring:
        if i == ' ':
            astring = astring.replace(i,'')

    last_element = 0
    list_of_elements = []

    #concatenates multiple digits in a row to form one number if applicable

    for i in range(len(astring)):
        if is_operand(astring[i]) or is_sign(astring[i]) or is_variable(astring[i]):
            list_of_elements.append(astring[last_element:i])
            last_element = i + 1
            list_of_elements.append(astring[i])

        if i == len(astring) - 1:
            list_of_elements.append(astring[last_element:])

    #deletes empty spaces

```

```python
    for i in list_of_elements:
        if i == '':
            list_of_elements.remove(i)

    signs_num = number_of_signs(list_of_elements)

    #concatenates two parts of inequality sign if applicable

    if signs_num > 0:
        for i in range(len(list_of_elements)-signs_num):
            if (list_of_elements[i] == '>' or '<') and list_of_elements[i+1] == '=':
                list_of_elements[i+1] = list_of_elements[i] + list_of_elements[i+1]
                list_of_elements.remove(list_of_elements[i])
            else:
                pass

    coef_list = []

    #converts a string to integer if it is int type

    for i in list_of_elements:
        if is_sign(i):
            coef_list.append(i)
        elif i.isdigit():
            coef_list.append(int(i))

    num_neg = number_of_negatives(coef_list)

    #concatenate '-' with a number

    for i in range(0,len(coef_list)-num_neg):
        if coef_list[i] == '-' and type(coef_list[i+1]) == int:
            coef_list[i+1] = int('-' + str(coef_list[i+1]))
            coef_list.remove(coef_list[i])

    return coef_list

```

Testing

Since my project is written in Interface-Model architecture and using OOP as the project paradigm, I was successful at separating classes into different files. This allowed me to test each element separately without affecting any others. When coding the User Interface, I attempted to tackle the majority of unexpected issues with the input. I used while loops and defensive programming (assertion, try…except) to ensure that the input is valid from every prompt request from the program. Because the user has access only to Interface layer, the user interacts with the program and has no access to simplex_method or tableau. Hence, data type inputted will be tested only in User Interface section, as the program would fail there and would not proceed to creating instances.

Main menu

According to functional requirement 2, I tested main menu with valid and invalid data input to see if it raises any errors. User has to choose one of the keys in the dictionary which are all of a string type. Valid input was expected to carry out the request successfully. Erroneous data was expected to be fixed by error handling and user would get into loop with a prompt with instructions.

```
Enter menu option:

---Main Menu---

[I]   :  Display instructions
[SF]  :  set an objective function
[CF]  :  change the objective function
[DF]  :  display the objective function
[AC]  :  add constraints
[CC]  :  change a constraint
[DC]  :  delete a constraint
[SC]  :  display the constraints
[S]   :  solve the problem
[X]   :  exit the program

sf
Do you wish to maximize or minimize? Enter "max" or "min":
```

```
Enter menu option:

---Main Menu---

[I]   :  Display instructions
[SF]  :  set an objective function
[CF]  :  change the objective function
[DF]  :  display the objective function
[AC]  :  add constraints
[CC]  :  change a constraint
[DC]  :  delete a constraint
[SC]  :  display the constraints
[S]   :  solve the problem
[X]   :  exit the program

2
Enter one of the instructions:
13.2
Enter one of the instructions:
False
Enter one of the instructions:
DC
There are no constraints. Use [SC] to add new constraints.
```

Test runs met the expectations for menu’s options.

Display instructions

When choosing the instruction [I], the program successfully prints out the instructions and returns back to main menu.

```
i
     ---INSTRUCTIONS---
Only following variables are allowed in that order  as an input: x, y, z, m, n.
To input an objective function, ignore the variable you are trying to maximize/minimize and input what goes after "=" sign with coefficients timed by (-1)
  example: If you want to maximize P=3x-2y, your input must be "-3x+2y".
If one of the variables is in objective function but not in constraint, put 0 before the variable in the constraint. If the coefficient is 1 or -1, write 1/-1 before the variable.
  example: Objective function: "x+5y-3z". "y" is not in one of the constraints, so your input must be: "-4x+0y+1z>33".
Not following the instructions will be followed with a wrong result.

Enter menu option:

---Main Menu---

[I]   :  Display instructions
[SF]  :  set an objective function
[CF]  :  change the objective function
[DF]  :  display the objective function
[AC]  :  add constraints
[CC]  :  change a constraint
[DC]  :  delete a constraint
[SC]  :  display the constraints
[S]   :  solve the problem
[X]   :  exit the program
```

Set an objective function

When this option is chosen, user has to input one of the options provided by the prompt. If the input is invalid, defensive programming(assertion) helps the program tackle the error and returns the user back to main menu.

```
sf
Do you wish to maximize or minimize? Enter "max" or "min":
a
Wrong input. Please, enter "max" or "min".


Enter menu option:
```

If the input for the first prompt is correct and considering functional requirement 2 for objective function:

```
sf
Do you wish to maximize or minimize? Enter "max" or "min":
min
min
Enter your objective function:
-3x+2y-11z

Enter menu option:
```

The program carries out the request successfully and returns back to menu.

Change the objective function

When this option is chosen, the following prompt is printed:

```
cf
Do you really wish to change objective function?(Yes/No)
a
Wrong input. Please, enter yes/no.


Enter menu option:
```

If the input is invalid, defensive programming(assertion) helps the program tackle the error and returns the user back to main menu.

```
cf
Do you really wish to change objective function?(Yes/No)
no

Enter menu option:
```

Choosing “no” returns user back to main menu.

```
cf
Do you really wish to change objective function?(Yes/No)
yes
Enter your objective function:
-2x+1y

Enter menu option:
```

Choosing “yes” allows user to set a new objective function and returns the user back to the main menu.

Display the objective function

This option displays a list of coefficients of stripped string inputted by the user. (You can see that this objective function is from the try-run in the previous section)

```
df

Your objective function is: [-2, 1]


Enter menu option:
```

Add constraints

If user chooses this option, he is asked the following:

```
ac
How many constraints: a
Input has to be a digit in [0-9]

Enter menu option:
```

```
ac
How many constraints: 234
Input has to be a digit in [0-9]

Enter menu option:
```

```
ac
How many constraints: 10
Input has to be a digit in [0-9]

Enter menu option:
```

```
ac
How many constraints: 0

Enter menu option:
```

```
ac
How many constraints: 9
Input your constraint #1:
```

```
ac
How many constraints: 3
Input your constraint #1: 7x+10y+34z<120
Input your constraint #2: 0x-1y+3z<=14
Input your constraint #3: 5x+3y+1z>=3

Enter menu option:
```

If the input is abnormal, program returns a prompt with correct input and returns user back to main menu. However, normal, extreme, boundary inputs are accepted.

After the valid input, program asks user to input the constraints one by one. User should respect the functional requirement 2 when inputting constraints as well as the objective function.

Change a constraint

If the user chooses this option straight after starting the program, he will see this prompt.

```
cc
There are no constraints. Use [AC] to add new constraints.
```

After using constraints from the example with 3 constraints in the previous section, next prompt is:

```
cc
How many constraints do you wish to change? five
Input has to be a digit in between 1 and 3. Try again.
```

User returns back to menu.

If the input is a digit but larger than the number of constraints already set, next prompt is printed:

```
cc
How many constraints do you wish to change? 4
Number of constraints: 3. Add new constraints using [AC] instruction
```

However, if the input is in the range:

```
cc
How many constraints do you wish to change? 1

Enter menu option:

---Constraints---
constraint #1:  [7, 10, 34, '<', 120]
constraint #2:  [0, -1, 3, '<=', 14]
constraint #3:  [5, 3, 1, '>=', 3]

Which constraint do you wish to change?
4
Index of constraint is out of range. There are only 3 constraints.
Out of 3 constraints, which one do you wish to change? 2
Input your new constraint:
 -3x+0y+5z>=71
```

The program tackles all invalid inputs and keeps running without any crashes. If the input for "how many constraints to change" prompt is N (greater than 1), the program would allow user to change a constraint N times.

A perfect run for change of constraint with already set constraints would look like the following:

```
cc
How many constraints do you wish to change? 1

Enter menu option:

---Constraints---
constraint #1:  [7, 10, 34, '<', 120]
constraint #2:  [0, -1, 3, '<=', 14]
constraint #3:  [5, 3, 1, '>=', 3]

Which constraint do you wish to change?
2
Input your new constraint:
 -5x+0y-3z>21
A constraint is changed successfully
```

Delete a constraint

Similar to previous section, the output would look the following if the constraints are not set yet.

```
dc
There are no constraints. Use [SC] to add new constraints.
```

Assuming the user has set 2 constraints he would get the next prompt:

```
dc
How many constraints you wish to delete?
 a
Input has to be a digit in between 1 and 2. Try again.
```

Using defensive programming, in particular try…except, while loops and assertions I managed to tackle all possible errors and the successful run with invalid inputs is shown:

```
dc
How many constraints you wish to delete?
 3
Input has to be a digit between 1 and 2. Try again.
4
Input has to be a digit between 1 and 2. Try again.
1

Enter menu option:

---Constraints---
constraint #1:  [-3, 1, '<', 54]
constraint #2:  [4, 6, '>', 12]

Which constraint do you wish to delete?
 second
Input has to be a digit between 1 and 2. Try again:
3
Input has to be a digit between 1 and 2. Try again:
1
The constraint is successfully deleted
```

Display the constraints

If this option is chosen, the program prints the constraints and user returns back to main menu.

```
sc

Enter menu option:

---Constraints---
constraint #1:  [4, 6, '>', 12]
```

Exit the program

If user wishes to leave and chooses “X” as his input, the program prints a prompt and stops running.

```
x
Thank you very much for using my program. See you!

Process finished with exit code 0
```

Solve the problem

If the user has not inputted anything but chose this option, the console would return this prompt:

```
s
You have not set an objective function.
You have not set constraints yet.
Is the input for your objective function and constraints correct? (Yes/No)
yes

Cannot use simplex method without setting the objective function or constraints. Please, try again...
```

If the user has constraints and no objective function:

```
s
You have not set an objective function.
---Constraints---
constraint #1:  [7, 10, 34, '<', 120]
constraint #2:  [5, 3, 1, '>=', 3]

Is the input for your objective function and constraints correct? (Yes/No)
yes

Cannot use simplex method without setting the objective function or constraints. Please, try again...
```

If the user has objective function but no constraints:

```
s
Your objective function is [-3, 2]
You have not set constraints yet.
Is the input for your objective function and constraints correct? (Yes/No)
ys
Wrong input. Please, enter 'yes' or 'no'.
 yes

Cannot use simplex method without setting the objective function or constraints. Please, try again...
```

If the user has set both:

```
s
Your objective function is [-3, 2]
---Constraints---
constraint #1:  [-2, 1, '>', 3]
constraint #2:  [5, -3, '<', 1]

Is the input for your objective function and constraints correct? (Yes/No)
yes
---Start of stage 1---

['x', 'y', 's1', 's2', 'a1', 'Value']
[-2.0, 1.0, -1.0, 0.0, 1.0, 3.0]
[5.0, -3.0, 0.0, 1.0, 0.0, 1.0]
[-3.0, 2.0, 0.0, 0.0, 0.0, 0.0]
[2.0, -1.0, 1.0, 0.0, 0.0, -3.0]

Basic variables: ['a1', 's2']
```

The program creates an instance of the tableau class and simplex_method class that iterates a table and prints out the result. Later in Testing section all iterations will be seen.

General testing

I have not implemented a function that checks if the input for the objective function or the constraints. My expectations are that the problem will not crash but will give wrong final result.

In this run, I tried to enter invalid input for the objective function and constraints. The following is the result:

```
---Main Menu---

[I]   :  Display instructions
[SF]  :  set an objective function
[CF]  :  change the objective function
[DF]  :  display the objective function
[AC]  :  add constraints
[CC]  :  change a constraint
[DC]  :  delete a constraint
[SC]  :  display the constraints
[S]   :  solve the problem
[X]   :  exit the program

sf
Do you wish to maximize or minimize? Enter "max" or "min":
max
max
Enter your objective function:
7functions
---Main Menu---

[I]   :  Display instructions
[SF]  :  set an objective function
[CF]  :  change the objective function
[DF]  :  display the objective function
[AC]  :  add constraints
[CC]  :  change a constraint
[DC]  :  delete a constraint
[SC]  :  display the constraints
[S]   :  solve the problem
[X]   :  exit the program

df

Your objective function is: []
```

```
---Main Menu---

[I]   :  Display instructions
[SF]  :  set an objective function
[CF]  :  change the objective function
[DF]  :  display the objective function
[AC]  :  add constraints
[CC]  :  change a constraint
[DC]  :  delete a constraint
[SC]  :  display the constraints
[S]   :  solve the problem
[X]   :  exit the program

ac
How many constraints: 2
Input your constraint #1: -3q+2m>5
Input your constraint #2: 5k-2w<10
---Main Menu---

[I]   :  Display instructions
[SF]  :  set an objective function
[CF]  :  change the objective function
[DF]  :  display the objective function
[AC]  :  add constraints
[CC]  :  change a constraint
[DC]  :  delete a constraint
[SC]  :  display the constraints
[S]   :  solve the problem
[X]   :  exit the program

s
Your objective function is []
---Constraints---
constraint #1:  [-2, '>', 5]
constraint #2:  ['-', '<', 10]

Is the input for your objective function and constraints correct? (Yes/No)
yes
```

```
---Start of stage 1---

['s1', 's2', 'a1', 'Value']
[0.0, 0.0, 1.0, 5.0]
[0.0, 0.0, 0.0, 10.0]
[0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, -5.0]

Basic variables: ['a1', 's2']

---End of stage 1---

---Start of stage 2---

['s1', 's2', 'Value']
[0.0, 0.0, 5.0]
[0.0, 0.0, 10.0]
[0.0, 0.0, 0.0]

Basic variables: ['a1', 's2']

---End of stage 2---

Infeasible solution
```

Although it gave correct final solution, which is infeasibilty, the input for objective function and constraints is not checked and resulted in a useless run.

Tableau

To compare my generated tableau with the expected one, I decided to use examples from the Edexcel A-level Further Mathematics Decision 1.

Because of my design of architecture, I can access the tableau class in a file and work with it without affecting simplex_method and mainmenu classes.

Trial run 1

D1, Page 200, Example 15

Example 15

Maximise $P = 3x - 2y + z$

subject to:

$$x + y + 2z \leqslant 10$$
$$2x - 3y + z \geqslant 5$$
$$x + y \geqslant 8$$
$$x, y, z \geqslant 0$$

The input to the menu would be similar, but for objective function the user would have to multiply the coefficients by (-1). Hence, user's input would be -3x+2y-z which would be stripped to

[-3, 2, -1]. The entry for constraints coefficients would be [1,1,2,"<=",10], [2,-3,1,">=",5], [1,1,0,">=",8].

The problem is to maximize the function, so creating an instance of the tableau will look like this:

```
max_or_min = 'max'
obj_function = [-3,2,-1]
constraints = [[1,1,2,'<',10], [2,-3,1,'>=',5], [1,1,0,'>=',8]]

example_15 = tableau(max_or_min, obj_function, constraints)
```

From the book:

The initial tableau is now given as:

Basic variable	x	y	z	s_1	s_2	s_3	a_1	a_2	Value
s_1	1	1	2	1	0	0	0	0	10
a_1	2	−3	1	0	−1	0	1	0	5
a_2	1	1	0	0	0	−1	0	1	8
P	−3	2	−1	0	0	0	0	0	0
I	−3	2	−1	0	1	1	0	0	−13

My initial tableau:

```
['x', 'y', 'z', 's1', 's2', 's3', 'a1', 'a2', 'Value']
[1.0, 1.0, 2.0, 1.0, 0.0, 0.0, 0.0, 0.0, 10.0]
[2.0, -3.0, 1.0, 0.0, -1.0, 0.0, 1.0, 0.0, 5.0]
[1.0, 1.0, 0.0, 0.0, 0.0, -1.0, 0.0, 1.0, 8.0]
[-3.0, 2.0, -1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[-3.0, 2.0, -1.0, 0.0, 1.0, 1.0, 0.0, 0.0, -13.0]

Basic variables: ['s1', 'a1', 'a2']
```

Since all coefficients are the same, trial run 1 was successful.

Trial run 2

D1, Page 204, Exercise 7D, Question 1b

b Minimise $C = x - 3y + z$

subject to:
$$x + 2y \leqslant 12$$
$$2x - y - z \geqslant 10$$
$$x + y + z \geqslant 6$$
$$x, y, z \geqslant 0$$

Arguments passed to the instance of a class:

```
max_or_min = 'min'
obj_function = [-1,+3,-1]
constraints = [[1,2,0,'<=',12],[2,-1,-1,'>=',10], [1,1,1,'>=',6]]

exercise1 = tableau(max_or_min, obj_function, constraints)
```

Book's first tableau:

Basic variable	x	y	z	s_1	s_2	s_3	a_1	a_2	Value	θ values
s_1	1	2	0	1	0	0	0	0	12	12
a_1	2	–1	–1	0	–1	0	1	0	10	5
a_2	1	1	1	0	0	–1	0	1	6	6
P	1	–3	1	0	0	0	0	0	0	
I	–3	0	0	0	1	1	0	0	–16	

My first tableau:

```
['x', 'y', 'z', 's1', 's2', 's3', 'a1', 'a2', 'Value']
[1.0, 2.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 12.0]
[2.0, -1.0, -1.0, 0.0, -1.0, 0.0, 1.0, 0.0, 10.0]
[1.0, 1.0, 1.0, 0.0, 0.0, -1.0, 0.0, 1.0, 6.0]
[1.0, -3.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[-3.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, -16.0]

Basic variables: ['s1', 'a1', 'a2']
```

Since all coefficients are the same, trial run 2 was successful.

Trial run 3

D1, Page 204, Exercise 7D, Question 1c

c Maximise $P = 3x - y + 2z$

subject to:
$$5x + z \leqslant 16$$
$$3x + y + z \leqslant 12$$
$$x - y + 4z \geqslant 9$$
$$x, y, z \geqslant 0$$

Arguments passed to the instance of a class:

```
max_or_min = 'max'
obj_function = [-3,1,-2]
constraints = [[5,0,1,'<=',16],[3,1,1,'<=',12],[1,-1,4,'>=',9]]

exercise1 = tableau(max_or_min, obj_function, constraints)
```

Book's first tableau:

Basic variable	x	y	z	s_1	s_2	s_3	a_1	Value	θ values
s_1	5	0	1	1	0	0	0	16	16
s_2	3	1	1	0	1	0	0	12	12
a_1	1	–1	4	0	0	–1	1	9	$\frac{9}{4}$
P	–3	1	–2	0	0	0	0	0	
I	–1	1	–4	0	0	1	0	–9	

My first tableau:

```
['x', 'y', 'z', 's1', 's2', 's3', 'a1', 'Value']
[5.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 16.0]
[3.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 12.0]
[1.0, -1.0, 4.0, 0.0, 0.0, -1.0, 1.0, 9.0]
[-3.0, 1.0, -2.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[-1.0, 1.0, -4.0, 0.0, 0.0, 1.0, 0.0, -9.0]

Basic variables: ['s1', 's2', 'a1']
```

Since all coefficients are the same, trial run 3 was successful.

Simplex_method

Because of my design of architecture, I can access the simplex_method class in a file and work with it without affecting tableau and mainmenu classes.

The trial run below is run with the tableau from Trial run 3 in Tableau section

Trial run

In simplex.py file, instantiation of the classes will look like the following:

```
max_or_min = 'max'
obj_function = [-3,1,-2]
constraints = [[5,0,1,'<=',16],[3,1,1,'<=',12],[1,-1,4,'>=',9]]

new_table = tableau(max_or_min,obj_function,constraints)
new_simulation = simplex_method(new_table)
```

After running the file, the following iterations are printed to the console and the book:

```
---Start of stage 1---

['x', 'y', 'z', 's1', 's2', 's3', 'a1', 'Value']
[5.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 16.0]
[3.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 12.0]
[1.0, -1.0, 4.0, 0.0, 0.0, -1.0, 1.0, 9.0]
[-3.0, 1.0, -2.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[-1.0, 1.0, -4.0, 0.0, 0.0, 1.0, 0.0, -9.0]

Basic variables: ['s1', 's2', 'a1']
```

Basic variable	x	y	z	s_1	s_2	s_3	a_1	Value	θ values
s_1	5	0	1	1	0	0	0	16	16
s_2	3	1	1	0	1	0	0	12	12
a_1	1	–1	4	0	0	–1	1	9	$\frac{9}{4}$
P	–3	1	–2	0	0	0	0	0	
I	–1	1	–4	0	0	1	0	–9	

```
['x', 'y', 'z', 's1', 's2', 's3', 'a1', 'Value']
[4.75, 0.25, 0.0, 1.0, 0.0, 0.25, -0.25, 13.75]
[2.75, 1.25, 0.0, 0.0, 1.0, 0.25, -0.25, 9.75]
[0.25, -0.25, 1.0, 0.0, 0.0, -0.25, 0.25, 2.25]
[-2.5, 0.5, 0.0, 0.0, 0.0, -0.5, 0.5, 4.5]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0]

Basic variables: ['s1', 's2', 'z']

---End of stage 1---
```

Basic variable	x	y	z	s_1	s_2	s_3	a_1	Value	Row operations
s_1	$\frac{19}{4}$	$\frac{1}{4}$	0	1	0	$\frac{1}{4}$	$-\frac{1}{4}$	$\frac{55}{4}$	R1 − R3
s_2	$\frac{11}{4}$	$\frac{5}{4}$	0	0	1	$\frac{1}{4}$	$-\frac{1}{4}$	$\frac{39}{4}$	R2 − R3
z	$\frac{1}{4}$	$-\frac{1}{4}$	1	0	0	$-\frac{1}{4}$	$\frac{1}{4}$	$\frac{9}{4}$	R3 ÷ 4
P	$-\frac{5}{2}$	$\frac{1}{2}$	0	0	0	$-\frac{1}{2}$	$\frac{1}{2}$	$\frac{9}{2}$	R4 + 2R3
I	0	0	0	0	0	0	1	0	R5 + 4R3

$I = 0$, so we have found a basic feasible solution. Proceed to the second stage.

```
---Start of stage 2---

['x', 'y', 'z', 's1', 's2', 's3', 'Value']
[4.75, 0.25, 0.0, 1.0, 0.0, 0.25, 13.75]
[2.75, 1.25, 0.0, 0.0, 1.0, 0.25, 9.75]
[0.25, -0.25, 1.0, 0.0, 0.0, -0.25, 2.25]
[-2.5, 0.5, 0.0, 0.0, 0.0, -0.5, 4.5]

Basic variables: ['s1', 's2', 'z']
```

Basic variable	x	y	z	s_1	s_2	s_3	Value	θ values
s_1	$\frac{19}{4}$	$\frac{1}{4}$	0	1	0	$\frac{1}{4}$	$\frac{55}{4}$	$\frac{55}{19}$
s_2	$\frac{11}{4}$	$\frac{5}{4}$	0	0	1	$\frac{1}{4}$	$\frac{39}{4}$	$\frac{39}{11}$
z	$\frac{1}{4}$	$-\frac{1}{4}$	1	0	0	$-\frac{1}{4}$	$\frac{9}{4}$	9
P	$-\frac{5}{2}$	$\frac{1}{2}$	0	0	0	$-\frac{1}{2}$	$\frac{9}{2}$	

```
['x', 'y', 'z', 's1', 's2', 's3', 'Value']
[1.0, 0.05263157894736842, 0.0, 0.21052631578947367, 0.0, 0.05263157894736842, 2.8947368421052633]
[0.0, 1.1052631578947367, 0.0, -0.5789473684210527, 1.0, 0.10526315789473684, 1.7894736842105257]
[0.0, -0.2631578947368421, 1.0, -0.05263157894736842, 0.0, -0.2631578947368421, 1.526315789473684]
[0.0, 0.631578947368421, 0.0, 0.5263157894736842, 0.0, -0.368421052631579, 11.736842105263158]

Basic variables: ['x', 's2', 'z']
```

Basic variable	x	y	z	s_1	s_2	s_3	Value	Row operations
x	1	$\frac{1}{19}$	0	$\frac{4}{19}$	0	$\frac{1}{19}$	$\frac{55}{19}$	$\frac{4}{19}\text{R1}$
s_2	0	$\frac{21}{19}$	0	$-\frac{11}{19}$	1	$\frac{2}{19}$	$\frac{34}{19}$	$\text{R2}-\frac{11}{4}\text{R1}$
z	0	$-\frac{5}{19}$	1	$-\frac{1}{19}$	0	$-\frac{5}{19}$	$\frac{29}{19}$	$\text{R3}-\frac{1}{4}\text{R1}$
P	0	$\frac{12}{19}$	0	$\frac{10}{19}$	0	$-\frac{7}{19}$	$\frac{223}{19}$	$\text{R4}+\frac{5}{2}\text{R1}$

```
['x', 'y', 'z', 's1', 's2', 's3', 'Value']
[1.0, -0.49999999999999994, 0.0, 0.5, -0.5, 0.0, 2.0000000000000004]
[0.0, 10.5, 0.0, -5.500000000000001, 9.5, 1.0, 16.999999999999996]
[0.0, 2.5, 1.0, -1.5, 2.5, 0.0, 5.999999999999999]
[0.0, 4.5, 0.0, -1.5000000000000004, 3.5000000000000004, 0.0, 18.0]

Basic variables: ['x', 's3', 'z']
```

Basic variable	x	y	z	s_1	s_2	s_3	Value	Row operations
x	1	$-\frac{1}{2}$	0	$\frac{1}{2}$	$-\frac{1}{2}$	0	2	$R1-\frac{1}{19}R2$
s_2	0	$\frac{21}{2}$	0	$-\frac{11}{2}$	$\frac{19}{2}$	1	17	$\frac{19}{2}R2$
z	0	$\frac{5}{2}$	1	$-\frac{3}{2}$	$\frac{5}{2}$	0	6	$R3+\frac{5}{19}R2$
P	0	$\frac{9}{2}$	0	$-\frac{3}{2}$	$\frac{7}{2}$	0	18	$R3+\frac{7}{19}R2$

```
['x', 'y', 'z', 's1', 's2', 's3', 'Value']
[2.0, -0.9999999999999999, 0.0, 1.0, -1.0, 0.0, 4.000000000000001]
[11.000000000000002, 5.0, 0.0, 0.0, 3.999999999999999, 1.0, 39.0]
[3.0, 1.0000000000000002, 1.0, 0.0, 1.0, 0.0, 12.0]
[3.000000000000001, 3.0, 0.0, 0.0, 2.0, 0.0, 24.000000000000004]

Basic variables: ['s1', 's3', 'z']

---End of stage 2---
```

Basic variable	x	y	z	s_1	s_2	s_3	Value	Row operations
s_1	2	–1	0	1	–1	0	4	2R1
s_3	11	5	0	0	4	1	39	$\text{R2}+\frac{11}{2}\text{R1}$
z	3	1	1	0	1	0	12	$\text{R3}+\frac{3}{2}\text{R1}$
P	3	3	0	0	2	0	24	$\text{R4}+\frac{3}{2}\text{R1}$

```
Optimal solution for maximization is 24.0

Values of variables:
s1 : 4.0
s3 : 39.0
z : 12.0
x : 0.0
y : 0.0
s2 : 0.0
```

Optimal $P = 24$ when $x = 0, y = 0, z = 12$

The values for function and variables are the same; therefore, I can confirm that my simplex method ran successfully.

Degenerative

The following objective function and constraint result in degenerative solution. The example is taken from https://www.brainkart.com/article/Special-Cases-in-the-Simplex-Method_11207/

```
max_or_min = 'max'
obj_function = [-3,-9]
constraints = [[1,4,'<=',8],[1,2,'<',4]]

new_table = tableau(max_or_min,obj_function,constraints)
new_simulation = simplex_method(new_table)
```

Running the simulation:

```
---Start of stage 1---

['x', 'y', 's1', 's2', 'Value']
[1.0, 4.0, 1.0, 0.0, 8.0]
[1.0, 2.0, 0.0, 1.0, 4.0]
[-3.0, -9.0, 0.0, 0.0, 0.0]

Basic variables: ['s1', 's2']

---End of stage 1---

---Start of stage 2---

['x', 'y', 's1', 's2', 'Value']
[1.0, 4.0, 1.0, 0.0, 8.0]
[1.0, 2.0, 0.0, 1.0, 4.0]
[-3.0, -9.0, 0.0, 0.0, 0.0]

Basic variables: ['s1', 's2']

['x', 'y', 's1', 's2', 'Value']
[0.25, 1.0, 0.25, 0.0, 2.0]
[0.5, 0.0, -0.5, 1.0, 0.0]
[-0.75, 0.0, 2.25, 0.0, 18.0]

Basic variables: ['y', 's2']

Possible degeneracy, see next iterations

['x', 'y', 's1', 's2', 'Value']
[0.0, 1.0, 0.5, -0.5, 2.0]
[1.0, 0.0, -1.0, 2.0, 0.0]
[0.0, 0.0, 1.5, 1.5, 18.0]

Basic variables: ['y', 'x']

---End of stage 2---
```

```
Optimal solution for maximization is 18.0

Values of variables:
y : 2.0
x : 0.0
s1 : 0.0
s2 : 0.0
```

After checking with the website solution, the result it the same.

Degenerative trial run is successful.

Unbounded solution

The example for a problem with an unbounded solution is taken from the same website as degenerative [https://www.brainkart.com/article/Special-Cases-in-the-Simplex-Method_11207/]

Set up:

```
max_or_min = 'max'
obj_function = [-2,-1]
constraints = [[1,-1,'<=',10],[2,0,'<=',40]]

new_table = tableau(max_or_min,obj_function,constraints)
new_simulation = simplex_method(new_table)
```

Running the simulation:

```
---Start of stage 1---

['x', 'y', 's1', 's2', 'Value']
[1.0, -1.0, 1.0, 0.0, 10.0]
[2.0, 0.0, 0.0, 1.0, 40.0]
[-2.0, -1.0, 0.0, 0.0, 0.0]

Basic variables: ['s1', 's2']

---End of stage 1---

---Start of stage 2---

['x', 'y', 's1', 's2', 'Value']
[1.0, -1.0, 1.0, 0.0, 10.0]
[2.0, 0.0, 0.0, 1.0, 40.0]
[-2.0, -1.0, 0.0, 0.0, 0.0]

Basic variables: ['s1', 's2']

['x', 'y', 's1', 's2', 'Value']
[1.0, -1.0, 1.0, 0.0, 10.0]
[0.0, 2.0, -2.0, 1.0, 20.0]
[0.0, -3.0, 2.0, 0.0, 20.0]

Basic variables: ['x', 's2']

['x', 'y', 's1', 's2', 'Value']
[1.0, 0.0, 0.0, 0.5, 20.0]
[0.0, 1.0, -1.0, 0.5, 10.0]
[0.0, 0.0, -1.0, 1.5, 50.0]

Basic variables: ['x', 'y']

The solution is unbounded. Another method is required for finding the optimal solution.
```

Infeasible solution

The example for a problem with an infeasible solution is taken from the same website as degenerative and unbounded solution sections https://www.brainkart.com/article/Special-Cases-in-the-Simplex-Method_11207/].

Set up:

```
max_or_min = 'max'
obj_function = [-3,-2]
constraints = [[2,1,'<=',2],[3,4,'>=',12]]

new_table = tableau(max_or_min,obj_function,constraints)
new_simulation = simplex_method(new_table)
```

Running the simulation:

```
---Start of stage 1---

['x', 'y', 's1', 's2', 'a1', 'Value']
[2.0, 1.0, 1.0, 0.0, 0.0, 2.0]
[3.0, 4.0, 0.0, -1.0, 1.0, 12.0]
[-3.0, -2.0, 0.0, 0.0, 0.0, 0.0]
[-3.0, -4.0, 0.0, 1.0, 0.0, -12.0]

Basic variables: ['s1', 'a1']

['x', 'y', 's1', 's2', 'a1', 'Value']
[2.0, 1.0, 1.0, 0.0, 0.0, 2.0]
[-5.0, 0.0, -4.0, -1.0, 1.0, 4.0]
[1.0, 0.0, 2.0, 0.0, 0.0, 4.0]
[5.0, 0.0, 4.0, 1.0, 0.0, -4.0]

Basic variables: ['y', 'a1']

No feasible solution
```

Full program test

```
Welcome to Simplex World! Using this program you can optimize your solution by using simplex method. Below you can see the main menu.
Before proceeding, reading the instructions is recommended for quick and easy use.

    ---INSTRUCTIONS---
Only following variables are allowed in that order  as an input: x, y, z, m, n.
To input an objective function, ignore the variable you are trying to maximize/minimize and input what goes after "=" sign with coefficients timed by (-1)
  example: If you want to maximize P=3x-2y, your input must be "-3x+2y".
If one of the variables is in objective function but not in constraint, put 0 before the variable in the constraint. If the coefficient is 1 or -1, write 1/-1 before the variable.
  example: Objective function: "x+5y-3z". "y" is not in one of the constraints, so your input must be: "-4x+0y+1z>33".
Not following the instructions will be followed with a wrong result.
---Main Menu---
```

```
[I]    :  Display instructions
[SF]   :  set an objective function
[CF]   :  change the objective function
[DF]   :  display the objective function
[AC]   :  add constraints
[CC]   :  change a constraint
[DC]   :  delete a constraint
[SC]   :  display the constraints
[S]    :  solve the problem
[X]    :  exit the program

sf
Do you wish to maximize or minimize? Enter "max" or "min":
max
max
Enter your objective function:
-2x-1y
```

```
---Main Menu---

[I]   :  Display instructions
[SF]  :  set an objective function
[CF]  :  change the objective function
[DF]  :  display the objective function
[AC]  :  add constraints
[CC]  :  change a constraint
[DC]  :  delete a constraint
[SC]  :  display the constraints
[S]   :  solve the problem
[X]   :  exit the program

ac
How many constraints: 2
Input your constraint #1: 1x+3y<=15
Input your constraint #2: 1x-1x>=11
---Main Menu---

[I]   :  Display instructions
[SF]  :  set an objective function
[CF]  :  change the objective function
[DF]  :  display the objective function
[AC]  :  add constraints
[CC]  :  change a constraint
[DC]  :  delete a constraint
[SC]  :  display the constraints
[S]   :  solve the problem
[X]   :  exit the program

s
Your objective function is [-2, -1]
---Constraints---
constraint #1:  [1, 3, '<=', 15]
constraint #2:  [1, -1, '>=', 11]

Is the input for your objective function and constraints correct? (Yes/No)
yes
```

```
---Start of stage 1---

['x', 'y', 's1', 's2', 'a1', 'Value']
[1.0, 3.0, 1.0, 0.0, 0.0, 15.0]
[1.0, -1.0, 0.0, -1.0, 1.0, 11.0]
[-2.0, -1.0, 0.0, 0.0, 0.0, 0.0]
[-1.0, 1.0, 0.0, 1.0, 0.0, -11.0]

Basic variables: ['s1', 'a1']

['x', 'y', 's1', 's2', 'a1', 'Value']
[0.0, 4.0, 1.0, 1.0, -1.0, 4.0]
[1.0, -1.0, 0.0, -1.0, 1.0, 11.0]
[0.0, -3.0, 0.0, -2.0, 2.0, 22.0]
[0.0, 0.0, 0.0, 0.0, 1.0, 0.0]

Basic variables: ['s1', 'x']

---End of stage 1---

---Start of stage 2---

['x', 'y', 's1', 's2', 'Value']
[0.0, 4.0, 1.0, 1.0, 4.0]
[1.0, -1.0, 0.0, -1.0, 11.0]
[0.0, -3.0, 0.0, -2.0, 22.0]

Basic variables: ['s1', 'x']

['x', 'y', 's1', 's2', 'Value']
[0.0, 1.0, 0.25, 0.25, 1.0]
[1.0, 0.0, 0.25, -0.75, 12.0]
[0.0, 0.0, 0.75, -1.25, 25.0]

Basic variables: ['y', 'x']
```

```
['x', 'y', 's1', 's2', 'Value']
[0.0, 4.0, 1.0, 1.0, 4.0]
[1.0, 3.0, 1.0, 0.0, 15.0]
[0.0, 5.0, 2.0, 0.0, 30.0]

Basic variables: ['s2', 'x']

---End of stage 2---

Optimal solution for maximization is 30.0

Values of variables:
s2 : 4.0
x : 15.0
y : 0.0
s1 : 0.0
```

The program successfully finished the run and presented no errors or terminals along with a correct result.

Evaluation

Requirements met

In overall, I believe my implementation of the project met almost all of the functional and non-functional requirements. Main menu, generating tableau and simplex method iteration work perfectly as it was shown in Testing section. There were no errors or crashes throughout the whole process and the program never terminated. A heavy use of defensive programming allowed my program to continue running and user to have an enjoyable experience with the simulation. Although the general test gave correct solution, the functional requirement 2, in particular 2.10-2.11 was not met. The input for objective function and constraints can be checked with new functions that count the number and type of variables and coefficients inputted and raise any errors if the input is invalid.

Based on the feedback I received from my colleagues that take Decision course as one of their options, all users enjoyed using the simulation and highlighted the positive aspects and areas to improve. They mentioned that the simulation processes information really fast and efficient.

Improvements

Despite all of the met requirements for the project and propositions from clients in "Dialogue with the clients" section, there are areas where the improvements should be made:

- Fix the issue found in general testing by introducing new functions
- When deleting a constraint, how the deleted constraint and the ones left
- When printing a tableau, add a row on the left of the tableau with basic variables

- Allow users to use "*" to show the multiplication between a coefficient and a variable
- Allow users to miss 1 and -1 as coefficients
- Allow users to not input 0 next to a variable if it is not in the constraint

Potential extensions

Potential extensions to the simulation are highlighted in "Extension Objectives" section of the Analysis section. Nevertheless, after receiving feedback from the users some of the areas to improve were mentioned:

- Save/load optimization purpose, objective function and constraints to a file
- Graphical User Interface
- In GUI, a wide range of buttons offered (digits, variables and signs) to choose from to build a function or a constraint instead of entering it as text
- After the simulation finished iterating, offer an option to show step-by-step explanation of how the simplex method works with an example
- Menu option for simplex method flow chart or description of how simplex method works

Bibliography

Class diagram and other charts creator tool - https://www.lucidchart.com/pages/

Edexcel A-Level Further Mathematics Decision 1 Book - https://www.pearsonactivelearn.com/app/home

Source of special cases examples - https://www.brainkart.com/article/Special-Cases-in-the-Simplex-Method_11207/