# 24MAC175 Group 1 Coursework Report

Matthew Lakin, James Death, Nathan Moore, Saad Tahir

November - December 2024

# Contents

# 1 Worked Example

# 2 Algorithm Explanation

The question we recieved to solve isn't able to be solved using the Simplex Method as it is. The Simplex Method is used to solve linear programming problems in the canonical form. So the first step is to clean the question for the Simplex method.

## 2.1 Converting to Canonical Form

The question we were asked to solve was as follows:

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

$$x_1 - x_2 + x_3 + x_5 + x_6 = 76 \tag{2.1}$$
$$x_2 - x_3 + x_4 + 3x_6 \leq 18 \tag{2.2}$$
$$x_1 + x_2 - 3x_3 + x_4 + x_5 \leq 12 \tag{2.3}$$
$$x_1 + x_2 + x_6 \geq 50 \tag{2.4}$$

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

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

$$\text{Minimise } z = \underline{c}^T \cdot \underline{x} \tag{2.5}$$
$$A\underline{x} = \underline{b} \tag{2.6}$$
$$\underline{x} \geq 0 \tag{2.7}$$

For this, we add slack and surplus variables to (2.1), (2.2), and (2.3) to remove the inequality in favour of an equality:

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

$$x_1 - x_2 + x_3 + x_5 + x_6 = 76 \tag{2.8}$$
$$x_2 - x_3 + x_4 + 3x_6 + S_1 = 18 \tag{2.9}$$
$$x_1 + x_2 - 3x_3 + x_4 + x_5 + S_2 = 12 \tag{2.10}$$
$$x_1 + x_2 + x_6 - S_3 = 50 \tag{2.11}$$

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

However (2.11) violates the non-negativity constraint, so to fix that we need to introduce artificial variables to the LPP. In order to make a basic feasible solution we need to add artificial variables to both (2.8) and (2.11). We will also amend the objective function with a large M value. This will be used in calculations so the artificial variables tend to 0 as iterations progress. So now the problem becomes:

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

$$x_1 - x_2 + x_3 + x_5 + x_6 + R_1 = 76 \tag{2.12}$$
$$x_2 - x_3 + x_4 + 3x_6 + S_1 = 18 \tag{2.13}$$
$$x_1 + x_2 - 3x_3 + x_4 + x_5 + S_2 = 12 \tag{2.14}$$
$$x_1 + x_2 + x_6 - S_3 + R_2 = 50 \tag{2.15}$$

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

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

## 2.2 Revised Simplex Algorithm

The Revised Simplex Algorithm is a variant of the Simplex Algorithm that is used to solve linear programming problems. It uses inverse matrices to simplifiy the calculations for computers.

The question consists of m constraints and n variables. The matrix A is $m \times n$ and is defined and defined as follows:

$$A = \begin{bmatrix} | & | & & | \\ \underline{P_1} & \underline{P_2} & \cdots & \underline{P_n} \\ | & | & & | \end{bmatrix} \tag{2.16}$$

The matrix A is partitioned into two matrices, $B$ and $N$, where $B$ is a $m \times m$ matrix and $N$ is a $m \times (n - m)$ matrix. The matrix $B$ is the basis matrix and the matrix $N$ is the non-basis matrix. The matrix $B$ is initially the identity matrix due to choosing slack and artificial variables. The matrix $N$ is the matrix of non-basic variables. The matrix $A$ is then defined as:

$$A = \begin{bmatrix} B & N \end{bmatrix} \tag{2.17}$$

We then see if the current solution is valid. Since we chose the basis using the slack and artificial variables, we know that it is valid. The amount of basic solutions is equal to $\binom{n}{m}$. This grows exponentially so it is impractical to enumerate all possible solutions.

### 2.2.1 Optimality Test

The optimality test is used to determine if the current solution is optimal. The optimality test is defined as:

$$\underline{c_N} - \underline{c_B^T} B^{-1} N \tag{2.18}$$

where $c_B$ is the cost vector of the basic variables, $c_N$ is the cost vector of the non-basic variables, and $B^{-1}$ is the inverse of the basis matrix. There are three possibilities for the optimality test:

- If all values in the optimality test positive, then any $x_B$ will increase $z$ so we're at the minimum.
- If all values are positive or zero, then the current solution is a non unique optima.
- If there is a negative value, then the current solution is not optimal. In this case, continue the algorithm.

We choose the most negative value as the entering variable. This is the variable that will enter the basis.

### 2.2.2 Feasibility Test

As we have found an entering variable, we find the associated column of N and call it $a_E$. We then find the ratios of the right hand side to the column of the entering variable. If all values are negative, then the problem is unbounded. If there is a positive value, then we choose the smallest value as the corrensponding leaving variable. This is the variable that will leave the basis.

Finally, we update the basis with the entering column from N, remove the leaving column from B and update the coefficients of the objective function. We then repeat until we reach the optimal solution.

# 3 Implementation

For this project, we used Python for access to the NumPy module and MatPlotLib for graphing. The code was written in a modular fashion, with each function performing a specific task.

## 3.1 Representation of the Problem

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

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

As we have code to convert the LPP to canonical form, we can accept any LPP in any form.

## 3.2 Main Method

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

## 3.3 convertToCanonicalForm

Documentation for convertToCanonicalForm is as follows:

```
Converts the LPP to canonical form.

    Args:
        c (numpy.ndarray): Coefficients of the objective function.
        A (numpy.ndarray): Coefficient matrix of the constraints.
        b (numpy.ndarray): Right-hand side of the constraints.
        signs (numpy.ndarray): List of indices of the basic variables.

    Returns:
        c (numpy.ndarray): Canonical Coefficients of the objective function.
        A (numpy.ndarray): Canonical Coefficient matrix of the constraints.
        b (numpy.ndarray): Canonical Right-hand side of the constraints.
        signs (list): List of {0} of length of constraints.
        basicIndices (list): List of indices of the basic variables.
        artificalIndices (list): List of indices of the artificial variables.
```

## 3.4 revisedSimplexMethod

Documentation for revisedSimplexMethod is as follows:

```
Performs the revised simplex method to solve a linear programming problem.

    Args:
        c (numpy.ndarray): Coefficients of the objective function.
        A (numpy.ndarray): Coefficient matrix of the constraints.
        b (numpy.ndarray): Right-hand side of the constraints.
        basic_vars (list): List of indices of the basic variables.

    Returns:
        dict: A dictionary containing the optimal value, the optimal solution,
            and the status of the solution.
```

## 3.5 renderLPP

## 3.6 graphing

# 4 Verification

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

$$x_1 - x_2 + x_3 + x_5 + x_6 = 76 \tag{4.1}$$
$$x_2 - x_3 + x_4 + 3x_6 \leq 18 \tag{4.2}$$
$$x_1 + x_2 - 3x_3 + x_4 + x_5 \leq 12 \tag{4.3}$$
$$x_1 + x_2 + x_6 \geq 50 \tag{4.4}$$

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

```
import numpy as np
from scipy.optimize import linprog

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

Since SciPy's method doesn't support the greater than or equal to constraints, we had to alter (4.4) by multiplying both sides of the equation by -1 and flipping the inequality.

$$-x_1 - x_2 - x_6 \leq -50 \tag{4.5}$$

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

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

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

The solution of the question corresponds with the "fun" value in the dictionary output. The value of the objective function is 200.0, which matches the value we found using the Revised Simplex Method. It also shows it took 5 iterations in the "nit" value which also aligns itself with our calculations. This verifies that our implementation is correct for this question.

# 5 Sensitivity Analysis

## 5.1 Behaviour of the Objective Function

## 5.2 Constraint 1

## 5.3 Constraint 2

## 5.4 Constraint 3

## 5.5 Constraint 4

# 6 Conclusion