Math 360 Portfolio

Alexander Kloska

Fall 2018, C. Wells

Contents

Part I

Self-Analysis and Discussion of Course Grade

Industry, Initiative, and Independence For Industry, Initiative, and Independence I would assign myself an A. Through much of my portfolio, I took the initiative to explore methods not discussed in class such as Mixed Integer Linear Programming (Section 2.9.3, Section 4.1.7), Big M method (Section 2.7), Lagrange multipliers method for nonlinear programming (Section 2.10.3), and the Bisection and Golden Section Line Search Methods (Section 2.10.4 and Section 2.10.5). In addition to this exploration, I have also written many functions for a package in R that can be used for linear optimization (Section 4.1).

Curiosity, Conjecturing, and Connections For Curiosity, Conjecturing, and Connections I would assign myself a A. This class has been extremely thought provoking for me and I find myself frequently reading about new methods in published papers. While the math in these papers is often over my head, I find that I am pushing myself harder to truly understand mathematical optimization. Moving into nonlinear programming, I immediately noticed some similarities from the affine scaling algorithm involving step sizes and overstepping boundaries.

Communication In Communication I would assign myself a A. I have attached a new section at the end of the portfolio called Client Reports (Section 5). In this section I have included four executive reports in layman terminology with references to detailed walkthroughs. Aside from client reporting, I focused on writing much of the portfolio in a style that helps to break down the individual steps of the ideas and algorithms used throughout the semester.

Reflection and Growth Mindset For Reflection and Growth Mindset I would assign myself a B+. I did not have a chance to reflect much more on many of the problems we've done simply due to not having enough time available. However, I still see myself coming back in the future to review these problems, potentially with new methods. Even as this is my final portfolio, I still have many questions and a strong desire to learn more.

Part II

Supporting Documentation

1 In-Class Assessments

1.1 Assessment 1

1. Consider the following Linear Program (LP):

$$\begin{array}{rrcrcrcr}
\text{maximize} & z = 2x_1 & & & + & 9x_3 & & \\
\text{subject to:} & 3x_1 & + & 2x_2 & - & x_3 & \geq & 10 \\
 & x_1 & - & x_2 & & & \leq & 0 \\
 & x_1 & + & 3x_2 & + & x_3 & \leq & 11 \\
 & & & & & x_1, x_2, x_3 & \geq & 0
\end{array}$$

   (a) Put the LP into standard inequality ($\leq$) form.

   (b) For each of the following solutions, determine whether the solution is feasible or not, and explain why.

   i. $x_1 = 0, x_2 = 0, x_3 = 0$

   ii. $x_1 = 0, x_2 = 0, x_3 = 11$

   iii. $x_1 = 0, x_2 = 0, x_3 = -10$

   iv. $x_1 = 2, x_2 = 2, x_3 = 0$

   (c) For each of the following solutions, evaluate the objective function value at the solution.

i. $x_1 = 0, x_2 = 0, x_3 = 0$

ii. $x_1 = 0, x_2 = 0, x_3 = 11$

iii. $x_1 = 0, x_2 = 0, x_3 = -10$

iv. $x_1 = 2, x_2 = 2, x_3 = 0$

2. Consider the following LP (where the objective function is unspecified):

$$\begin{array}{lrl} \text{maximize} & z = & f(x_1, x_2) \\ \text{subject to} & -x_1 + & x_2 \leq 2 \\ & x_1 - & 2x_2 \leq 1 \\ & & x_1, x_2 \geq 0 \end{array}$$

(a) Sketch and shade the feasible region. You may use the included grid.

(b) Is the *feasible region* bounded or unbounded? How can you tell?

(c) Assuming that the objective for the LP is

$$\text{maximize} \quad z = x_1 - 3x_2$$

i. Draw and label (with the objective value) several contour lines for this objective function;

ii. Explain, based on the feasible region and your contours, whether the LP:

- is unbounded,
- is infeasible, or
- has optimal solution(s);

Stretch Questions

(d) The following parts involve **the same constraints** (and thus the same feasible region), but require you to

- **find an appropriate (linear) objective function**
  or
- **explain why it is impossible to find such a (linear) objective function.**

(Assume in each case that you are *maximizing* your objective function.)

Hint: Start with the geometry, then design contour lines, and then develop the function.

i. Find an objective function so that the LP has no optimal solution.

ii. Find an objective function so that the LP is unbounded.

iii. Find an objective function so that the LP has a unique optimal solution.

iv. Find an objective function so that the LP has exactly two optimal solutions.

v. Find an objective function so that the LP has infinitely many optimal solutions.

vi. Find an objective function so that the LP is infeasible.

Alex Kloska

Progress Assessment

1. a) Already in standard inequality form

b) i) Not feasible. Does not satisfy first constraint $3x_1 + 2x_2 - x_3 \geq 10$ ✓

ii) Not feasible. Does not satisfy first constraint $3x_1 + 2x_2 - x_3 \geq 10$ ✓

iii) Not feasible. Does not satisfy negativity constraint $x_3 \geq 0$ ✓

iv) Feasible. Satisfies all constraints. ✓

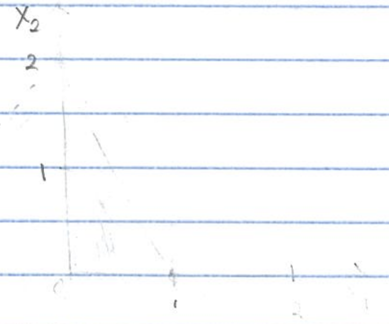
c) i) $z = 0$ ✓

ii) $z = 99$ ✓

iii) $z = -90$ ✓

iv) $z = 4$ ✓

2. a)



b) The feasible region is unbounded because it is a cone shape that extends infinitely ✓

c) ~~At $(0.25, 0.25)$, $z = -0.50$~~

~~$-0.50 = x_1 - 3x_2$~~

~~$x_1 = 0$, $x_2 = 1/6$~~

~~$x_2 = 0$, $x_1 = -0.5$~~

~~At $(0.5, 0.5)$, $z = -1$~~

~~$-1 = x_1 - 3x_2$~~

~~$x_1 = 0$, $x_2 = 1/3$~~

$x_2 = 0$, $x_1 = -1$

(i) Optimal solution at $(1, 0)$

Alex Kloska

Progress Assessment 1 **Math 360, C. Wells** **September 12, 2018**

1.1.1 Original (Scanned)

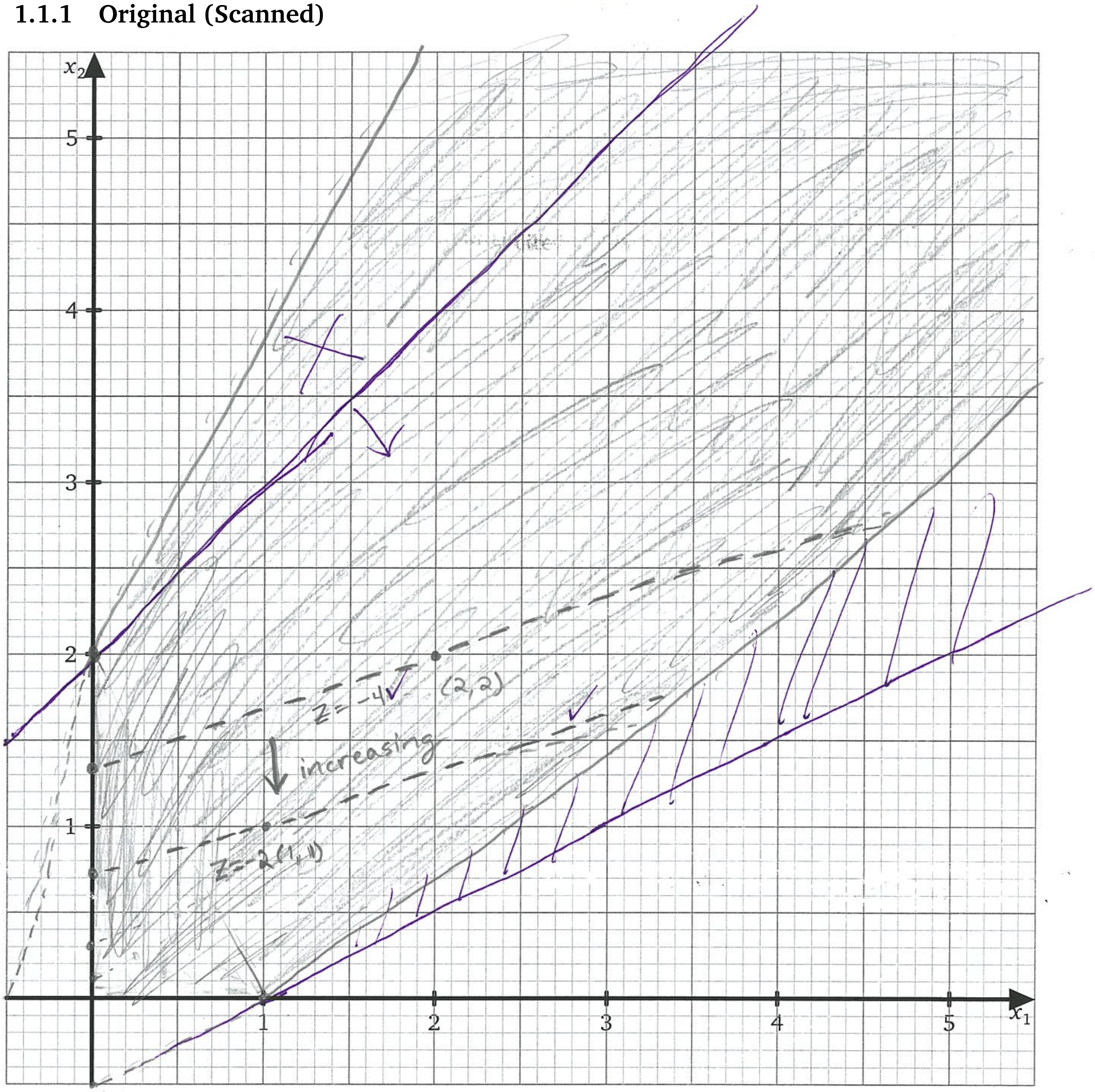


Figure 1: Sketch and Shade the Feasible Region.

t (2,2), Z = -4

4 = X_1 - 3X_2

X_1 = 0, X_2 = 4/3

X_2 = 0, X_1 = -4

At (1,1), Z = -2

-2 = X_1 - 3X_2

X_1 = 0, X_2 = 2/3

$$
\begin{array}{ll}
\text{maximize} & z = f(x_1, x_2) \\
\text{subject to} & -x_1 + \quad x_2 \leq 2 \\
& \quad x_1 - \ 2x_2 \leq 1 \\
& \quad x_1, x_2 \geq 0
\end{array}
$$

1.1.2 Reflection

1.2 Assessment 2

1.2.1 In-Class Portion

1. Consider the linear program

$$\begin{array}{ll} \text{maximize} & z = 2x_1 + 3x_2 \\ \text{subject to} & \quad 4x_1 + 3x_2 \leq 15 \\ & \quad\;\; x_1 + \;\; x_2 \leq \;\; 4 \\ & \qquad\; x_1, x_2 \geq 0 \end{array}$$

Classify each of the following solutions as one of:

- infeasible solution,
- feasible boundary solution, or
- feasible interior solution.

(a) $\vec{x} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$ **Infeasible**

(b) $\vec{x} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$ **Infeasible**

(c) $\vec{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ **Feasible Interior**

(d) $\vec{x} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ **Feasible Boundary**

(a) and (b) are infeasible because they violate constraints. (c) is a feasible interior solution because it is within all of the constraints and not touching a boundary. (d) is a feasible boundary solution because it is at the boundary.

*See scanned document at end for responses to questions 2 and 3.

2. Consider each of the following tableaux.

   (a) Does the tableau represents a valid basic solution? If so, what are the basic variables and what is the basic solution?

   (b) Does the tableau represents a basic feasible solution?

   (c) Does the tableau represents an optimal solution?

   (d) If the tableau represents a basic feasible solution that is not optimal, what would be your next step in finding an optimal basic feasible solution (using the simplex algorithm)?

   (e) If time allows, carry out your step.

I)

z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	s_5	RHS
1	$78/19$	0	0	0	$20/19$	0	$13/19$	0	$834/19$
0	$33/19$	1	0	0	$7/19$	0	$-4/19$	0	$138/19$
0	$-8/19$	0	1	0	$-4/19$	0	$5/19$	0	$-30/19$
0	$-172/19$	0	0	1	$-48/19$	0	$22/19$	0	$-645/19$
0	$17/19$	0	0	0	$-20/19$	1	$6/19$	0	$-131/19$
0	$-137/19$	0	0	0	$-59/19$	0	$31/19$	1	$-775/19$

II)

z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	s_5	RHS
1	$104/11$	0	0	$-13/22$	$28/11$	0	0	0	$1407/22$
0	$18/11$	0	1	$-5/22$	$4/11$	0	0	0	$135/22$
0	$1/11$	1	0	$2/11$	$-1/11$	0	0	0	$12/11$
0	$37/11$	0	0	$-3/11$	$-4/11$	1	0	0	$26/11$
0	$-86/11$	0	0	$19/22$	$-24/11$	0	1	0	$-645/22$
0	$61/11$	0	0	$-31/22$	$5/11$	0	0	1	$155/22$

III)

z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	s_5	RHS
1	0	$-32/7$	$-51/7$	0	0	$6/7$	0	0	$114/7$
0	0	$40/7$	$6/7$	1	0	$-4/7$	0	0	$71/7$
0	0	1	2	0	1	-1	0	0	11
0	1	$4/7$	$2/7$	0	0	$1/7$	0	0	$19/7$
0	0	$12/7$	$41/7$	0	0	$-4/7$	1	0	$50/7$
0	0	$31/7$	$-9/7$	0	0	$-8/7$	0	1	$9/7$

IV)

z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	s_5	RHS
1	$1/3$	0	0	$5/12$	0	0	$7/6$	0	$119/4$
0	$5/12$	1	0	$7/48$	0	0	$-1/24$	0	$37/16$
0	$1/3$	0	1	$-1/12$	0	0	$1/6$	0	$5/4$
0	$43/12$	0	0	$-19/48$	1	0	$-11/24$	0	$215/16$
0	$14/3$	0	0	$-5/12$	0	1	$-1/6$	0	$29/4$
0	$47/12$	0	0	$-59/48$	0	0	$5/24$	1	$15/16$

3. Below are listed several simplex tableaux. Each represents a maximization LP at a basic feasible solution.

   Your goal is to analyze the tableaux thoroughly. Some suggestions are the following:

   (a) Determine which variables are basic and which are non-basic.

   (b) List the current basic feasible solution (including all variables and their values).

   (c) i. Perform an appropriate simplex pivot, explaining why you chose the entering and leaving variables as you did, or

ii. Explain why no pivot is possible/appropriate and what that says about the LP.

(d) If you can, determine whether the LP, for example, is unbounded, is infeasible, is degenerate, is at optimum, has alternate optima, *etc*.

Tableau I

z	x_1	x_2	x_3	s_1	s_2	s_3	RHS
1	0	−2	0	1	0	0	10
0	0	−1	0	1	1	5	2
0	1	−4	0	4	0	8	6
0	0	0	1	4	0	−1	8

Tableau II

z	x_1	x_2	x_3	s_1	s_2	s_3	RHS
1	−2	0	1	0	2	0	6
0	2	1	6	0	5	0	5
0	4	0	8	1	12	0	8
0	3	0	9	0	−2	1	9

Tableau III

z	x_1	x_2	x_3	s_1	s_2	s_3	RHS
1	0	1	0	4	0	0	8
0	0	0	1	1	1	0	3
0	0	−1	−3	1	0	1	4
0	1	1	1	−2	0	0	5

1.2.2 Take Home Portion

1. Solve the following LP using the simplex algorithm.

$$\begin{array}{rl}
\text{Maximize:} & 6x_1 + 8x_2 + 9x_3 \\
\text{Subject to:} & 4x_1 + 8x_2 + 2x_3 \leq 21 \\
& 7x_1 + 5x_2 + 4x_3 \leq 30 \\
& 7x_1 + 4x_2 + 2x_3 \leq 19 \\
& 4x_1 + 4x_2 + 7x_3 \leq 18 \\
& 8x_1 + 9x_2 + x_3 \leq 23 \\
& x_1, x_2, x_3 \geq 0
\end{array}$$

Answer: Setting up the tableau:

z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	s_5	*RHS*
1	−6	−8	−9	0	0	0	0	0	0
0	4	8	2	1	0	0	0	0	21
0	7	5	4	0	1	0	0	0	30
0	7	4	2	0	0	1	0	0	19
0	5	4	$\boxed{7}$	0	0	0	1	0	18
0	8	9	1	0	0	0	0	1	23

Iteration 1:
Entering variable: x_3

Leaving variable: s_4
Pivoting at row 5, column 3 returns:

z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	s_5	RHS
1	−0.86	−2.86	0	0	0	0	1.29	0	23.14
1	2.86	$\boxed{6.86}$	0	1	0	0	−0.29	0	15.86
0	4.71	2.71	0	0	1	0	−0.57	0	19.71
0	5.86	2.86	0	0	0	1	−0.29	0	13.86
0	0.57	0.57	1	0	0	0	0.14	0	2.57
0	7.43	8.43	0	0	0	0	−0.14	1	20.43

Iteration 2:
Entering variable: x_2
Leaving variable: s_1
Pivoting at row 2, column 2 returns:

z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	s_5	RHS
1	0.33	0	0	0.42	0	0	1.17	0	29.75
0	0.42	1	0	0.15	0	0	−0.04	0	2.31
0	3.58	0	0	−0.40	1	0	−0.46	0	13.44
0	4.67	0	0	−0.42	0	1	−0.17	0	7.25
0	0.33	0	1	−0.08	0	0	0.17	0	1.25
0	3.92	0	0	−1.23	0	0	0.21	1	0.94

This is the optimal solution because there are no more negatives in the objective row. The solution is at $x_1 = 0$, $x_2 = 2.31$, $x_3 = 1.25$.

2. Consider the following LP:

$$\begin{array}{ll} \text{maximize:} & x_1 + x_2 + x_3 \\ \text{subject to:} & x_1 + 2x_2 - x_3 \leq 5 \\ & 3x_1 + x_2 + x_3 \leq 8 \\ & -x_1 + x_2 \leq 1 \\ & x_1, x_2, x_3 \geq 0 \end{array}$$

This LP has an optimum at $x_1 = 0, x_2 = 1, x_3 = 7$. Use this information to construct the final tableau and determine if the LP has any alternate optima.

3. Find the dual of the following linear program.

$$\begin{array}{ll} \text{maximize:} & 6x_1 + 8x_2 + 9x_3 \\ \text{subject to:} & 4x_1 + 8x_2 + 2x_3 \leq 21 \\ & 7x_1 + 5x_2 + 4x_3 \leq 30 \\ & 7x_1 + 4x_2 + 2x_3 \leq 19 \\ & 4x_1 + 4x_2 + 7x_3 \leq 18 \\ & 8x_1 + 9x_2 + x_3 \leq 23 \\ & x_1, x_2, x_3 \geq 0 \end{array}$$

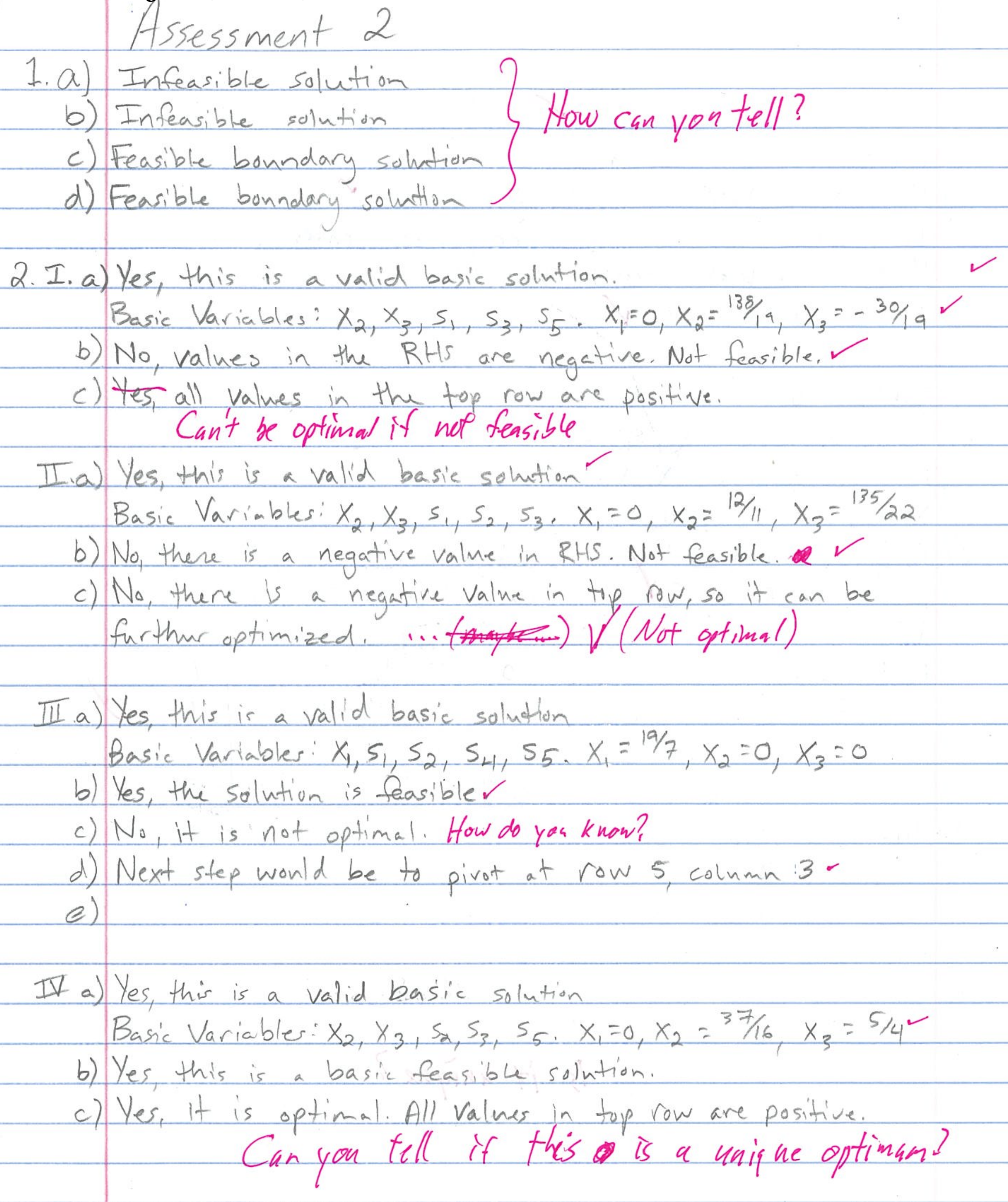
Answer: The dual linear program is:

$$\begin{array}{rl} \text{Minimize:} & 21x_1 + 30x_2 + 19x_3 + 18x_4 + 23x_5 \\ \text{subject to:} & 4x_1 + 7x_2 + 7x_3 + 4x_4 + 8x_5 \geq 6 \\ & 8x_1 + 5x_2 + 4x_3 + 4x_4 + 9x_5 \geq 8 \\ & 2x_1 + 4x_2 + 2x_3 + 7x_4 + x_5 \geq 9 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0 \end{array}$$

1.2.3 Reflection

Assessment 2 was a relatively easy assessment to do with the except of number 2 on the Take Home portion. Based on my performance for assessment 2, I am confident in my understanding of feasibility, basic solutions, and primal-dual relationships. Elementary matrices are still a section of linear algebra that I need to work on understanding better.

1.2.4 Original (Scanned)



Alex Kloska
MTH 360

Assessment 2

1. a) Infeasible solution
b) Infeasible solution
c) Feasible boundary solution
d) Feasible boundary solution

How can you tell?

2. I. a) Yes, this is a valid basic solution. ✓
Basic Variables: X_2, X_3, S_1, S_3, S_5. $X_1=0$, $X_2 = 138/19$, $X_3 = -30/19$ ✓
b) No, values in the RHS are negative. Not feasible. ✓
c) ~~Yes,~~ all values in the top row are positive.
Can't be optimal if not feasible

II. a) Yes, this is a valid basic solution ✓
Basic Variables: X_2, X_3, S_1, S_2, S_3. $X_1=0$, $X_2 = 12/11$, $X_3 = 135/22$
b) No, there is a negative value in RHS. Not feasible. ✓
c) No, there is a negative value in top row, so it can be furthur optimized. ...(~~maybe~~) ✓ (Not optimal)

III a) Yes, this is a valid basic solution
Basic Variables: X_1, S_1, S_2, S_4, S_5. $X_1 = 19/7$, $X_2 = 0$, $X_3 = 0$
b) Yes, the solution is feasible ✓
c) No, it is not optimal. How do you know?
d) Next step would be to pivot at row 5, column 3 ✓
e)

IV a) Yes, this is a valid basic solution
Basic Variables: X_2, X_3, S_2, S_3, S_5. $X_1=0$, $X_2 = 37/16$, $X_3 = 5/4$ ✓
b) Yes, this is a basic feasible solution.
c) Yes, it is optimal. All values in top row are positive.
Can you tell if this is a unique optimum?

3. Tableau I

a) Basic Variables: x_1, x_3, s_2 ✓

Non-Basic Variables: x_2, s_1, s_3

b) $x_1 = 6, x_2 = 0, x_3 = 8, s_1 = 0, s_2 = 2, s_3 = 0$ ✓

c) No pivot is possible. x_2 is only viable entering variable, but there is no viable leaving variable.

d) This is unbounded. ✓

Tableau II

a) Basic Variables: x_2, s_1, s_3

Non-Basic Variables: x_1, x_3, s_2

b) $x_1 = 0, x_2 = 5, x_3 = 0, s_1 = 8, s_2 = 0, s_3 = 9$

c) Entering Variable: x_1 ✓ (only negative in row 1)

Leaving Variable: s_2 ✓ (lowest value from ratio test)

$R3 \leftarrow R3 \div 4$

$R1 \leftarrow R1 + 2R3$

$R2 \leftarrow R2 - 2R3$

$R4 \leftarrow R4 - 3R3$

Z	x_1	x_2	x_3	s_1	s_2	s_3	RHS
1	0	0	5	1/2	8	0	10
0	0	1	2	-1/2	-1	0	1
0	1	0	2	1/4	3	0	2
0	0	0	3	-3/4	-11	1	3

✓

d) This LP is now at optimum

Tableau III

a) Basic Variables: x_1, s_2, s_3

Non-Basic Variables: x_2, x_3, s_1

b) $x_1 = 5, x_2 = 0, x_3 = 0, s_1 = 0, s_2 = 3, s_3 = 4$

c) No pivot possible. No viable entering variable.

d) This LP is at optimum Is it unique?

1.3 Assessment 3

*See scanned document after the assessment for responses to questions 1 and 2.

1. Explain the Two-Phase Simplex Algorithm[1]. In particular,
   - Explain the purpose of the Two Phase Simplex Algorithm,
   - Explain the implementation of the Two Phase Simplex Algorithm, and
   - Explain the various possible outcomes and their consequences.
2. An LP has objective function maximize $3x_1 + 2x_2 - x_3$, slack variables s_1, s_2, s_3, and s_4, four main constraints, and all variables non-negative.
   (a) At any basic feasible solution, how many variables are basic and how many are non-basic?
   (b) The current point is $(x_1, x_2, x_3, s_1, s_2, s_3, s_4) = (0, 0, 6, 0, 8, 3, 5)$. Is this a basic point or not, and how can you tell?
   (c) You have the following options for moving from the current point,

$$\begin{aligned}\vec{d}_1 &= \begin{pmatrix}1 & 0 & 4 & 0 & -3 & 2 & 1\end{pmatrix}\\ \vec{d}_2 &= \begin{pmatrix}0 & 1 & 1 & 0 & 1 & 1 & 1\end{pmatrix}\\ \vec{d}_3 &= \begin{pmatrix}0 & 0 & 2 & 1 & -1 & 1 & -2\end{pmatrix}.\end{aligned}$$

      i. Which direction(s) are directions of improvement, and how can you tell?
      ii. Explain how you can tell that the LP is unbounded.

3. Professor Emmet (Doc) Brown is writing a grant proposal that involves implementing a new algorithm he has developed. He estimates that it would require at least 1000 hours of programming by a professional consultant, but he can substitute undergraduate programmers or graduate programmers. The costs for an hour of time for undergraduates, graduates, and consultants are, respectively, $4, $10, and $25.

Doc's experience is that an hour of undergraduate time is approximately equivalent to 1/5 of an hour of professional time, and graduate student time is roughly equivalent to 3/10 hours of professional time.

Programmers also require oversight. Doc estimates that he will need to invest 1/5, 1/10, and 1/20 hour of his own time for each hour of programming from undergrads, grads, and professionals; he only has 164 hours to devote to the programming aspects of the project, however.

Finally, due to the number of graduate students available, Doc can use at most 500 hours of graduate student time.

[1] You may choose to explain the "Big M" method if you prefer.

(a) Set up and solve a linear program for this situation.

(b) Based on your final tableau, how much money could be saved if Doc could find more of his own time to contribute to the project?

4. Let **P** be the linear program

$$\begin{array}{ll} \text{maximize} & z = 2x_1 + 3x_2 \\ \text{subject to} & 4x_1 + 3x_2 \leq 15 \\ & x_1 + x_2 \leq 4 \\ & x_1, x_2 \geq 0 \end{array}$$

Classify each of the following solutions as one of:

- infeasible solution,
- feasible boundary solution, or
- feasible interior solution.

(a) $\vec{x} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$ **Infeasible Solution**

(b) $\vec{x} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$ **Infeasible Solution**

(c) $\vec{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ **Feasible Interior**

(d) $\vec{x} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ **Feasible Boundary**

5. Farmer John and Farmer Jane want to plant some or all of their 45 acres with wheat and/or corn. They estimate that each acre of wheat will yield \$300 profit, while each acre of corn will yield \$200. In their experience, labor (workers per acre) and fertilizer (tons per acre) required are as follows:

	wheat	corn	available
labor	3	2	100
fertilizer	2	4	120

This yields the following linear program:

$$\begin{array}{ll} \text{maximize} & 200w + 300c \\ \text{subject to} & 3w + 2c \leq 100 \\ & 2w + 4c \leq 120 \\ & w + c \leq 45 \\ & c, w \geq 0 \end{array}$$

Jane adds slack variables and uses the simplex algorithm to arrive at the following optimal tableau: (NOTE: You do not need to re-solve this if you do not wish to do so. It is correct!)

z	x_1	x_2	s_1	s_2	s_3	RHS
1	0	0	25	$125/2$	0	$10,000$
0	1	0	$1/2$	$-1/4$	0	20
0	0	1	$-1/4$	$3/8$	0	20
0	0	0	$-1/4$	$-1/8$	1	5

Answer the following questions based on the above tableau:

(a) If John and Jane have the opportunity to rent more land for planting at \$2000 per acre for the growing season, should they? Explain.
They should not rent more land for \$2000 per acre because each acre of land added will only increase revenue by \$25.

(b) John and Jane find a fertilizer distributor who can sell them fertilizer at \$55 per ton. Should they purchase more?
Yes, they should purchase more fertilizer from this distribution because it will increase their profits by \$7.50 for each ton of fertilizer they purchase (\$62.50 - \$55).

(c) How much additional profit (if any) could John and Jane expect if they had an additional 10 workers?
They should not expect any additional profit because as it is, they already have 5 unused workers, so adding more workers will only decrease profits unless they are free labor.

(d) For parts 2a and 2b, what is the maximum John and Jane could rent/purchase and still have the same optimal basis?
For Part 1, they could rent up to 20 more acres of land without changing the basis. If they rented 20 acres of land, they would fully utilize all of their workers. For Part 2, they could purchase up to 40 additional tons of fertilizer while remaining in the same basis. This will use up the remainder of the unused workers.

(e) Look again at part 2c. If John and Jane have 10 extra workers, how much of their land should they plant with corn versus wheat?
If they had 10 additional workers, they should still plant 20 acres of corn and 20 acres of wheat. Looking at the column of the shadow price, adding additional workers will not change any of the other variables RHS values.

6. Consider the sketch of the feasible region and solutions (points) shown below, where the arrows indicate the feasible side for each constraint:

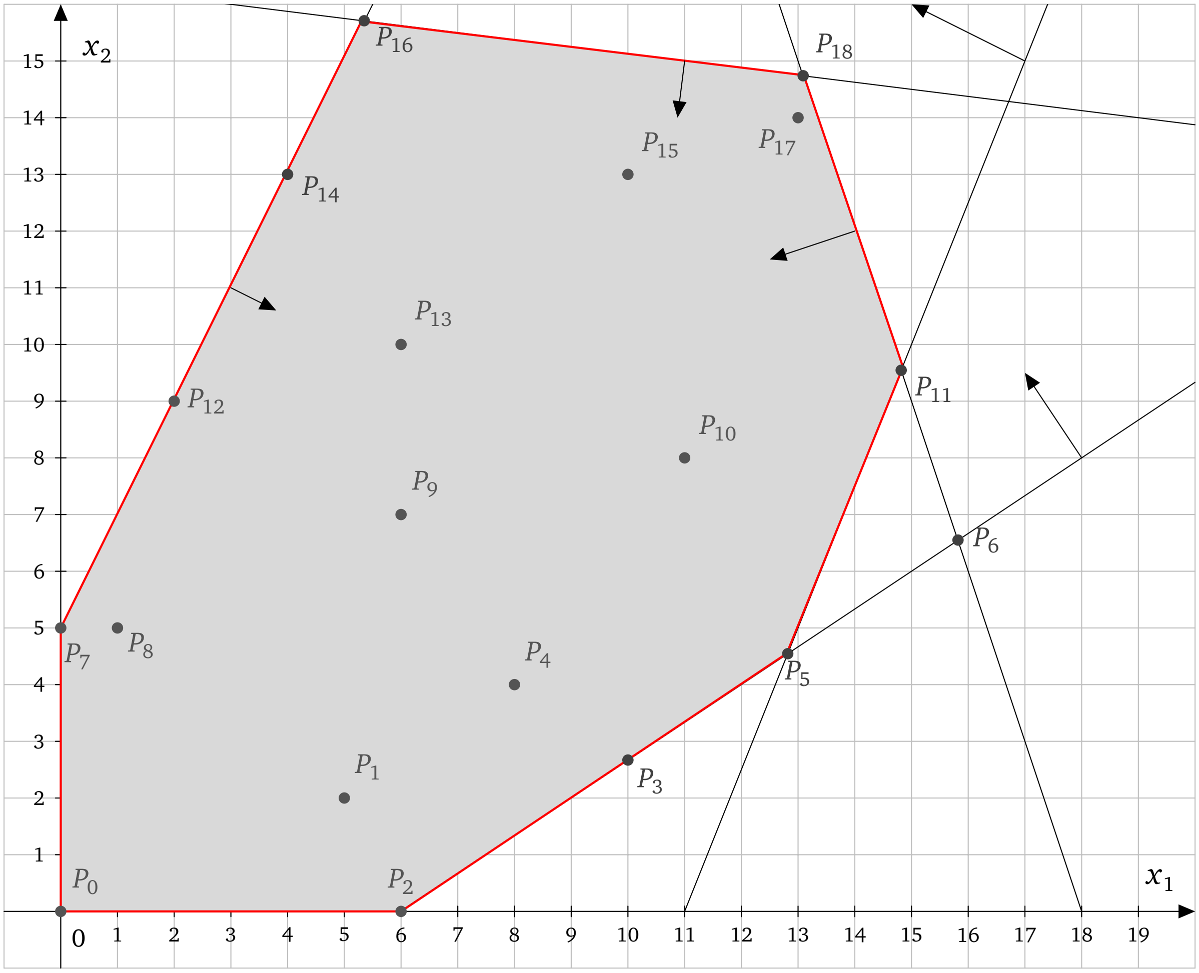


Figure 1: Feasible Region and Points

(a) Shade the feasible region and outline the boundary of the region.

(b) Which of the following successions of solutions could have arisen from: (1) the simplex algorithm, (2) an interior point algorithm, or (3) neither? Be sure to explain how you determined your answer.

i. $P_0, P_2, P_3, P_5, P_{11}$
This could have arisen from the simplex algorithm because it starts at the origin and each solution is at a corner.

ii. $P_1, P_9, P_{13}, P_{15}, P_{17}$
This could have arisen from an interior point algorithm because none of them are at a boundary.

iii. $P_8, P_9, P_{13}, P_{14}, P_{16}$
This could have arisen from an interior point algorithm because it starts at an interior point and you can follow the steps as they move in the optimal direction.

iv. P_0, P_7, P_{16}, P_{18}
This could have arisen from the simplex algorithm because it starts at the origin and moves to corners.

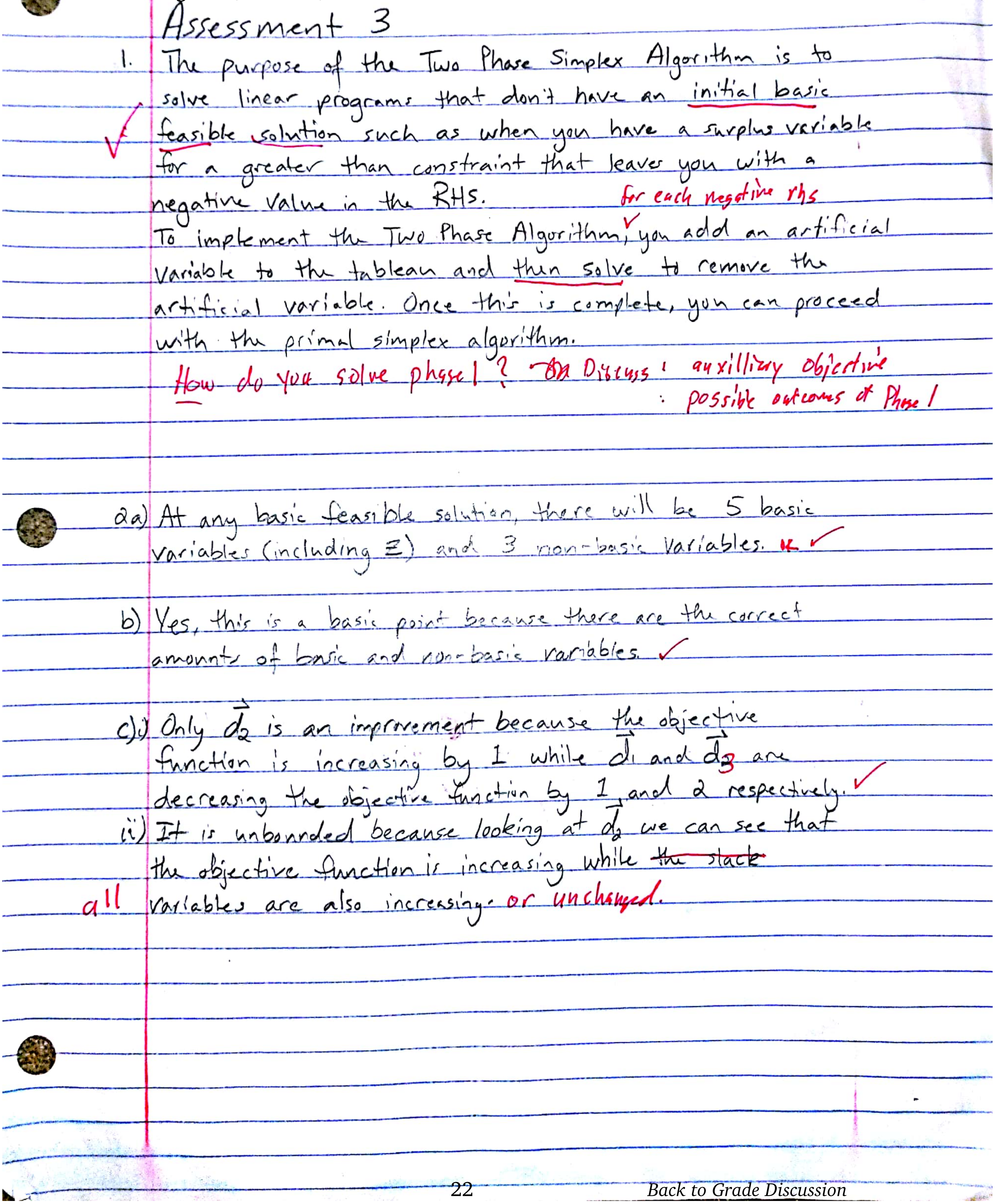
v. $P_0, P_2, P_5, P_6, P_{11}$
This solution could not have come from the simplex algorithm or an interior point algorithm because P_6 violates the constraints and oversteps a boundary.

1.3.1 Reflection

I feel that I have a very good grasp on all of the topics included in Assessment 3. The only part that gave me any struggle was Question 3 which I could not figure out how to set up. I will continue to work on that problem. I have a good understanding of how the Two Phase Simplex Algorithm works and how to perform sensitivity analysis. I'm relatively certain I was able to correctly identify the algorithms used in Question 6(b).

Alex Kloska

1.3.2 Original (Scanned)



Assessment 3

1. The purpose of the Two Phase Simplex Algorithm is to solve linear programs that don't have an initial basic feasible solution such as when you have a surplus variable for a greater than constraint that leaves you with a negative value in the RHS. ✓ for each negative rhs

To implement the Two Phase Algorithm, ✓ you add an artificial variable to the tableau and then solve to remove the artificial variable. Once this is complete, you can proceed with the primal simplex algorithm.

How do you solve phase I ? ~~On~~ Discuss : auxillary objective
: possible outcomes of Phase I

2a) At any basic feasible solution, there will be 5 basic variables (including z) and 3 non-basic variables. ✓

b) Yes, this is a basic point because there are the correct amounts of basic and non-basic variables. ✓

c) i) Only $\vec{d}_2$ is an improvement because the objective function is increasing by 1 while $\vec{d}_1$ and $\vec{d}_3$ are decreasing the objective function by 1, and 2 respectively. ✓

ii) It is unbounded because looking at $\vec{d}_2$ we can see that the objective function is increasing while ~~the slack~~ all variables are also increasing. or unchanged.

Scanned by CamScanner

1.4 Assessment 4

1.4.1 Take Home Portion

1. Perform one step of the affine scaling interior point algorithm for the following LP, with current solution as shown:

$$\begin{array}{rl} \text{maximize:} & x_1 + 2x_2 + x_3 \\ \text{subject to:} & x_1 + 2x_2 + 2x_3 = 7 \\ & x_1 + 2x_3 = 3 \\ & x_1, x_2, x_3 \geq 0 \end{array}$$

$$\vec{x}_c = \begin{bmatrix} 1.0 \\ 2.0 \\ 1.0 \end{bmatrix}$$

2. Let $f : \mathbb{R}^2 \to \mathbb{R}$ be defined by

$$f(x,y) = x^4 - 12x^3 + 54x^2 + y^2 - 108x + 2y + 82.$$

The gradient, $\nabla f(x,y)$ and Hessian, $\nabla^2 f(x,y)$ are given by

$$\nabla f(x,y) = \begin{bmatrix} 4x^3 - 36x^2 + 108x - 108 \\ 2y + 2 \end{bmatrix} \quad \nabla^2 f(x,y) = \begin{bmatrix} 12x^2 - 72x + 108 & 0 \\ 0 & 2 \end{bmatrix}$$

(a) Verify that $\vec{x} = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$ is a global minimum for f.

$\vec{x} = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$ **is a global minimum because for a step in any direction from this point results in a positive Hessian determinant, meaning that the slopes are positive.**

(b) Perform two steps to the steepest descent algorithm for f with initial point $\vec{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

(c) Perform two steps to Newton's method for f with initial point $\vec{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

(d) Compare the results of Newton's method and steepest descent.

3. Briefly explain the KKT conditions and why they make sense for an NLP of the form

$$\begin{array}{rl} \text{Maximize:} & f(x_1,x_2) \\ \text{Subject to:} & g(x_1,x_2) \leq 0. \end{array}$$

4. Let $f : \mathbb{R}^2 \to \mathbb{R}$ be defined by

$$f(x,y) = x^4 - 12x^3 + 54x^2 + y^2 - 108x + 2y + 82.$$

The gradient, $\nabla f(x,y)$ and Hessian, $\nabla^2 f(x,y)$ are given by

$$\nabla f(x,y) = \begin{bmatrix} 4x^3 - 36x^2 + 108x - 108 \\ 2y + 2 \end{bmatrix} \quad \nabla^2 f(x,y) = \begin{bmatrix} 12x^2 - 72x + 108 & 0 \\ 0 & 2 \end{bmatrix}$$

Find x, y, and λ that satisfy the KKT conditions for the NLP

$$\begin{array}{rl} \text{Minimize:} & f(x \ \ , y\ ) \\ \text{Subject to:} & x^2 + y^2 \ \leq 9. \end{array}$$

1.4.2 Reflection

Unfortunately, I did not have enough time to fully go through Assessment 4. Problem 1 was difficult because I did not know how to handle the equality constraints for affine scaling. However, I do understand the process of affine scaling as shown in Section 2.8. I added a section on the KKT conditions (Section 2.10.6) later in my portfolio.

2 Practice Problems

2.1 Linear Programming

2.1.1 Fertilizer for Corn

A farmer has 160 acres and plans to plant corn. The corn requires at least 0.16 lbs. of nitrogen (f_1) per acre and at least 0.24 lbs. of phosphorus (f_2) per acre. The first fertilizer is 4% nitrogen and 8% phosphorous and costs 30 cents per pound. The second fertilizer is 8% nitrogen and 4% phosphorous and costs 25 cents per pound. Determine the best course of action.

$$\begin{array}{rl} \text{Minimize:} & 0.30f_1 + 0.25f_2 = \quad C \\ \text{Subject to:} & 0.04f_1 + 0.08f_2 \leq 25.6 \\ & 0.08f_1 + 0.04f_2 \leq 38.4 \\ & \quad f_1, f_2 \geq 0 \end{array}$$

The first equation is the objective function (the function we are trying to optimize) and the following three are its constraints. To find a solution, we start by determining the boundaries set by the constraints.

2.1.2 Choosing Crops

The second problem that we worked on for class involved deciding how much of each crop to plant to maximize the profits utilizing 45 acres of farmland. Here are is the objective function and constraints derived from the problem.

c = Acres of corn
w = Acres of wheat
P = Profit

$$\begin{array}{rrrrl} \text{Maximize:} & 400c + & 300w = & P \\ \text{Subject to:} & c + & w \leq & 45 \\ & 4c + & 2w \leq & 165 \\ & 40c + & 60w \leq & 2320 \\ & c, w \geq 0 & & \end{array}$$

We converted this into matrix form which produced the following:

$$\begin{bmatrix} P & c & w & s_1 & s_2 & s_3 & & RHS \\ 1 & -400 & -300 & 0 & 0 & 0 & \vdots & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & \vdots & 45 \\ 0 & 4 & 2 & 0 & 1 & 0 & \vdots & 165 \\ 0 & 60 & 40 & 0 & 0 & 1 & \vdots & 2320 \end{bmatrix} \tag{2.1}$$

where s_1, s_2, s_3 are slack variables

2.1.3 Corn, Wheat, or Rye

This problem is the same as the previous except with the addition of another crop, rye, which provides a profit of $150 per bushel. This was our tableau setup:

z	c	w	r	L	F	H	RHS
1	−400	−300	−250	0	0	0	0
0	1	1	1	1	0	0	45
0	$\boxed{4}$	2	1	0	1	0	165
0	40	60	20	0	0	1	2320

We then selected *c* as our entering variable because it had the most negative coefficient. To determine the leaving variable, we performed the ratio test by dividing the *RHS* column by column *c*. The smallest ratio of the three was given by *F* which was 41.25 so that was chosen as our leaving variable. We performed a pivot at [3,2] and arrived at our next tableau.

z	c	w	r	L	F	H	RHS
1	0	−100	−150	0	100	0	16500
0	0	1/2	$\boxed{3/4}$	1	−1/4	0	15/4
0	1	1/2	1/4	0	1/4	0	165/4
0	0	40	10	0	−10	1	670

This tableau tells us how the situation has changed after planting 41.25 acres of corn. Planting 41.25 acres of corn and nothing else leaves us a profit of $16500 with 3.75 unused acres and 670 unused labor hours. The next move is to choose another entering variable and leaving variable and continue to optimize it using the simplex algorithm. Rye has the most negative coefficient so it will be the entering variable since we are using a greedy algorithm. The smallest ratio is given by L so land will be the leaving variable. We perform a pivot at [2,4] and end up with the following:

z	c	w	r	L	F	H	RHS
1	0	0	0	200	50	0	$\boxed{17250}$
0	0	2/3	1	4/3	−1/3	0	5
0	1	1/3	0	−1/3	1/3	0	40
0	0	100/3	0	−40/3	−20/3	1	620

This is our optimal solution because there are no more negative coefficients in the objective function, so there is nothing we can do from here that would increase profits. This tableau tells us that we planted 40 acres of corn and 5 acres of rye with 620 unused labor hours and brought in $17250.

2.2 Bland's Anti-Cycling Rule

In the previous subsection, we use a greedy algorithm to determine an optimal solution which means always choosing the variable that increases the objective function the most. This algorithm works in most cases, but it risks cycling. In other words, you may continue to pivot forever and never reach an optimal solution. To avoid this issue, Bland's anti-cycling rule is used. If you are doing any automation on computers, this is the selection process you will most likely want to use. The rule is essentially to select the first viable index that is found for the entering variable. The leaving variable is selected the same way because we still do not want to RHS to become negative.

2.2.1 Example: Corn, Wheat, Rye

I will use the Corn, Wheat, and Rye problem to demonstrate the selection process for Bland's rule which will happen to end up finding a different solution, but will retain the same optimized objective function value. The implications of this will be discussed in the next section on convex combinations. So we begin with the same initial tableau:

z	c	w	r	L	F	H	RHS
1	-400	-300	-250	0	0	0	0
0	1	1	1	1	0	0	45
0	$\boxed{4}$	2	1	0	1	0	165
0	40	60	20	0	0	1	2320

For selecting the entering variable, we choose the first viable column index. In this case, corn will be our entering variable because it is the first negative variable. The leaving variable will be fertilizer because it has the smallest ratio.

z	c	w	r	L	F	H	RHS
1	0	-100.0	-50.00	0	100.00	0	16500.00
0	0	$\boxed{0.5}$	0.75	1	-0.25	0	3.75
0	1	0.5	0.25	0	0.25	0	41.25
0	0	40.0	10.00	0	-10.00	1	670.00

For the next iteration, we will choose wheat as our entering variable because it is the first viable entering variable. Our leaving variable will be land because it has the smallest ratio.

z	c	w	r	L	F	H	RHS
1	0	0	100.0	200.0	50.0	0	17250.0
0	0	1	1.5	2.0	-0.5	0	7.5
0	1	0	-0.5	-1.0	0.5	0	37.5
0	0	0	-50.0	-80.0	10.0	1	370.0

We now have an optimal solution. Notice that the objective function value is the same, but it is achieved at a different point than when the greedy algorithm is used. Both solutions are correct because they are a convex combination.

2.3 Convex Combinations of Solutions

As seen in the last section, we had performed pivots at different points and still arrived at a different optimized tableau with the same objective value. It is entirely possible for a linear program to have multiple solutions if the objective function runs along a boundary. These solutions are called convex combinations of vectors. They are represented using the two end point vectors with proportion scalars. The equation for a convex combination of points is:

$$\vec{x}_{optimal} = t\vec{b}_1 + (1-t)\vec{b}_2$$

where b_1 is the RHS for one optimal solution, b_2 is the RHS for another optimal solution along the same boundary, and t is a scalar between 0 and 1.

Any value of t will provide a vector of points that is between the two given points, and this vector will retain the same objective function value. Using the two solutions that were determined for the Corn, Wheat, and Rye problems, the convex combination is:

$$\begin{bmatrix} z \\ c \\ w \\ r \\ l \\ f \\ h \end{bmatrix} = t \begin{bmatrix} 17250 \\ 37.5 \\ 7.5 \\ 0 \\ 0 \\ 0 \\ 370 \end{bmatrix} + (1-t) \begin{bmatrix} 17250 \\ 40 \\ 0 \\ 5 \\ 0 \\ 0 \\ 620 \end{bmatrix}$$

2.4 Duality

For this problem we are introduced to Transco, a manufacturer of cars, trucks, and locomotives. The goal is to optimize profits under the constraints of steel, rubber, and labor hours. This is the information that we were given:

Vehicle Type	Steel (tons)	Rubber (tons)	Labor (hours)	Profit
Cars	2	0.4	40	1000
Trucks	3	0.7	50	1500
Locomotives	5	0.2	100	2600
Resources Available	40	6	600	

Setting up the linear program:

$$\begin{array}{rrrrl} \text{Maximize:} & 1000c + & 1500t + & 2600l = & P \\ \text{Subject to:} & 2c + & 3t + & 5l \leq & 40 \\ & 0.4c + & 0.7t + & 0.2l \leq & 6 \\ & 40c + & 50t + & 100l \leq & 600 \\ & c, t, l \geq 0 & & & \end{array}$$

Setting up in vector notation:

$$\vec{C} = \begin{bmatrix} 1000 \\ 1500 \\ 2600 \end{bmatrix}, \text{ A} = \begin{bmatrix} 2 & 3 & 5 \\ 0.4 & 0.7 & 0.2 \\ 40 & 50 & 100 \end{bmatrix}, \vec{b} = \begin{bmatrix} 40 \\ 6 \\ 600 \end{bmatrix}$$

$$\begin{array}{rl} \text{Maximize:} & \vec{C}^T \vec{x} \\ \text{Subject to:} & A\vec{x} \leq \vec{b} \\ & \vec{x} \geq 0 \end{array}$$

The purpose of this problem was to introduce duality of linear programs. One way to look at this problem is to maximize the profit, but another way to look at it is to minimize the resources used. To do so, the problem will have to be set up differently. Setting up in vector notation:

$$\vec{C} = \begin{bmatrix} 40 \\ 6 \\ 600 \end{bmatrix}, \text{ A} = \begin{bmatrix} 2 & 0.4 & 40 \\ 3.4 & 0.7 & 50 \\ 5 & 0.2 & 100 \end{bmatrix}, \vec{b} = \begin{bmatrix} 1000 \\ 1500 \\ 2600 \end{bmatrix}$$

$$\begin{array}{rl} \text{Minimize:} & \vec{b}^T\vec{y} \\ \text{Subject to:} & A^T\vec{y} \geq \vec{C} \\ & \vec{x} \geq 0 \end{array}$$

2.4.1 Primal

First I will solve the problem as a maximization. Setting up the tableau:

z	Cars	Trucks	Locomotives	Steel	Rubber	Hours	RHS
1	-1000	-1500	-2600	0	0	0	0
0	2	3	5	1	0	0	40
0	0.4	0.7	0.2	0	1	0	6
0	40	50	$\boxed{100}$	0	0	1	600

Entering variable: Locomotives
Leaving variable: Hours
Pivoting at column 3, row 4

z	Cars	Trucks	Locomotives	Steel	Rubber	Hours	RHS
1	40	-200	0	0	0	26	15600
0	0	0.1	0	0.2	0	-0.01	2
0	1.6	$\boxed{3}$	0	0	5	-0.01	24
0	0.4	0.5	1	0	0	.01	6

Entering variable: Trucks
Leaving Variable: Rubber
Pivoting at column 2, row 3

z	Cars	Trucks	Locomotives	Steel	Rubber	Hours	RHS
1	146.67	0	0	0	333.33	25.33	17200
0	-0.27	0	0	1	-0.83	-0.05	6
0	0.53	1	0	0	1.67	-0.00	8
0	0.13	0	1	0	-0.83	0.01	2

Profits are now maximized subject to the objective function and its constraints. The best course of action is to build 8 trucks and 2 locomotives. This will net a profit of $17200 given the resources on hand.

2.4.2 Dual

Now we will solve its dual and see that by minimizing utilization of supplies, we will also maximize the profits.

Initial Tableau:

z	Steel	Rubber	Hours	Cars	Trucks	Locomotives	RHS
1	40	6.0	600	0	0	0	0
0	-2	-0.4	-40	1	0	0	-1000
0	-3	-0.2	-50	0	1	0	-1500
0	-5	-0.7	$\boxed{-100}$	0	0	1	-2600

Since all of the objective row is positive and all of the RHS is negative (excluding the objective value), we must use the dual simplex algorithm to solve this. Using a greedy algorithm, we will select the most negative RHS as our leaving variable first. This is -2600 which corresponds to the hours constraint. Next we will divide the objective row by the leaving row and select the value closest to zero as the entering variable. In this case, we will pivot at [4,3] where locomotive is the entering variable and hours is the leaving variable. The result of this pivot is:

z	Steel	Rubber	Hours	Cars	Trucks	Locomotives	RHS
1	10.00	4.800	0	0	0	6.00	-15600
0	0.00	-0.320	0	1	0	-0.40	40
0	-0.50	$\boxed{-0.600}$	0	0	1	-0.50	-200
0	0.05	0.002	1	0	0	-0.01	26

The first thing that stands out here is the objective value. It's absolute value is equivalent to the absolute value after the first iteration of the primal simplex above. Additionally, notice that the values in the RHS also correspond to the values of the non-basic variables in the objective row of the primal. Going through the same selection process, the next pivot is on [3,2] and the result is:

z	Steel	Rubber	Hours	Cars	Trucks	Locomotives	RHS
1	6.00	0	0	0	8.00	2.00	-17200.00
0	0.27	0	0	1	-0.53	-0.13	146.67
0	0.83	1	0	0	-1.67	0.83	333.33
0	0.05	0	1	0	0.00	-0.01	25.33

Now the dual is at optimal because there are no more negative values in the RHS. The absolute value of the objective value for the dual is equivalent to that of the primal. Interestingly, the pivot locations in the dual were the same as the pivot locations in the primal.

2.4.3 Strong Duality and Weak Duality

There are two types of duality, strong and weak. These concepts define the relationship between a primal and its dual. When a linear program has strong duality, this means that the solution to the dual will also be the same solution to the primal. This is typically the case in strictly linear programming. Weak duality becomes prominent in integer programming. Weak duality simply states that the solution to the dual problem represents an upper bound to the solution for the primal.

The reasoning behind weak duality is that the dual is trying to minimize usage while the primal is trying to maximize the cost function. When these variables are integers, different portions of their planes may be restricted to achieve integer values. For strong duality to be true, the planes would need to be restricted in the exact same way, otherwise the allowed integer values may be different.

2.5 Sensitivity Analysis

Sensitivity analysis is a very useful technique in mathematical optimization. It allows us to determine if it is worth the money to buy more of our restricting factors. For example, in the bakery problem in Section 3.1, the bakery used up a few of their ingredients on hand during the optimization. Using sensitivity analysis, we can determine if they would have higher profits if they initially had more of those ingredients.

The simplest form of sensitivity analysis is as described above, where there is a change to the RHS constraints. To demonstrate this, I will use the Corn, Wheat, and Rye problem. We will start at the optimized tableau.

z	Corn	Wheat	Rye	Land	Fertizizer	Hours	RHS
1	0	0	0	200	$\boxed{50}$	0	17250
0	0	2/3	1	4/3	-1/3	0	5
0	1	1/3	0	-1/3	1/3	0	40
0	0	100/3	0	-40/3	-20/3	1	620

How would this problem change if the farmer had more land or fertilizer? Let us analyze fertilizer. The first step is to look at what is called the shadow price of the fertilizer. Simply put, this value is how much the objective value will change if an additional unit of that variable is added. It is the variables coefficient in the objective row of the tableau (I have boxed it in the tableau above). The shadow price of fertilizer is $50, meaning that every unit of fertilizer that is added to the RHS will increase the objective value by $50. But we need to be careful because if we add too many additional units of fertilizer, it may change the basis of the optimized program. The maximum amount of a shadow variable that may be added without altering the basis is limited by the rows beneath the shadow price. As we add more units of fertilizer, this will change the values in the RHS. We do not want any negatives in the RHS, so we will add fertilizer until one of the RHS approaches 0. In this case, we are limited by row 2, or wheat. For each additional fertilizer that is added, we will plant $\frac{1}{3}$ less acres of wheat, but we only have 5 acres of wheat available to reduce. So the maximum amount of fertilizer we can add without changing the basis is 15 units. After purchasing 15 units of fertilizer, our optimal tableau will become:

z	Corn	Wheat	Rye	Land	Fertilizer	Hours	RHS	ΔRHS
1	0	0	100.0	200	50.0	0	$\boxed{18000}$	750.0
0	0	1	1.5	2	-0.5	0	0	-5
0	1	0	-0.5	-1	0.5	0	45	5
0	0	0	-50.0	-80	10.0	1	520	100

2.6 Two Phase Simplex Algorithm

The Two Phase Simplex Algorithm is a method that is used when there is no initial basic feasible solution to the linear program. The purpose of phase 1 is to create a initial basic feasible solution so that in phase 2 another algorithm may be applied to solve the linear program.

2.6.1 Example: Corn, Wheat, Rye

Say another constraint was added to the Corn, Wheat, and Rye problem. Now we are required to plant at least 8 acres of wheat. This complicates the linear program because now we have a greater than constraint while the simplex algorithm only works with less than constraints. To tackle this problem, we will need to approach the problem using the two phase simplex. Here is the new initial setup:

$$\begin{array}{rrrrl}
\text{Maximize:} & 400c+ & 300w+ & 250r & \\
\text{Subject to:} & c+ & w+ & r & \leq 45 \\
 & 4c+ & 2w+ & r & \leq 165 \\
 & 40c+ & 60w+ & 20r & \leq 2320 \\
 & & w & & \geq 8 \\
 & c,w,r \geq 0 & & &
\end{array}$$

Now we will add slack and excess variables to create equality put it into tableau form:

z	Corn	Wheat	Rye	Land	Fertilizer	Hours	e_1	RHS
1	-400	-300	-150	0	0	0	0	0
0	1	1	1	1	0	0	0	45
0	4	2	1	0	1	0	0	165
0	40	60	20	0	0	1	0	2320
0	0	1	0	0	0	0	-1	8

Now we do not have an initial basic solution because there is no identity matrix. Next we will introduce our artificial variable a to create an initial basic solution.

z	Corn	Wheat	Rye	Land	Fertilizer	Hours	e_1	a_1	RHS
1	-400	-300	-150	0	0	0	0	0	0
0	1	1	1	1	0	0	0	0	45
0	4	2	1	0	1	0	0	0	165
0	40	60	20	0	0	1	0	0	2320
0	0	$\boxed{1}$	0	0	0	0	-1	1	8

If we select our artificial variable as the leaving variable and wheat as the entering variable and perform a pivot at [5,2] then we arrive at:

z	Corn	Wheat	Rye	Land	Fertilizer	Hours	e_1	a_1	RHS
1	-400	0	-150	0	0	0	-300	300	2400
0	1	0	1	1	0	0	1	-1	37
0	4	0	1	0	1	0	2	-2	149
0	40	0	20	0	0	1	60	-60	1840
0	0	1	0	0	0	0	-1	1	8

Now we have an initial basic feasible solution and the artificial variable is equivalent to zero, so we can remove it. Our initial tableau for phase 2 is:

z	Corn	Wheat	Rye	Land	Fertilizer	Hours	e_1	RHS
1	-400	0	-150	0	0	0	-300	2400
0	1	0	1	1	0	0	1	37
0	4	0	1	0	1	0	2	149
0	40	0	20	0	0	1	60	1840
0	0	1	0	0	0	0	-1	8

And now using the simplex algorithm, the optimal solution becomes:

z	Corn	Wheat	Rye	Land	Fertilizer	Hours	e_1	RHS
1	0	0	250	400	0	0	100	17200
0	1	0	1	1	0	0	1	37
0	0	0	-3	-4	1	0	-2	1
0	0	0	-20	-40	0	1	20	360
0	0	1	0	0	0	0	-1	8

Given the new requirement to plant at least 8 acres of wheat, the best choice is to plant 37 acres of corn and 8 acres of wheat. This will net a profit of $17200.

2.7 Big M Method

The Big M method is an alternative to the Two Phase Simplex Algorithm that approaches the problem a little differently. Instead of eliminating the artificial variables that are created in phase 1, the Big M method creates artificial variables that have unrealistic coefficients that force the simplex algorithm to push them to zero. This is called a penalty function, meaning the program is heavily penalized for choosing the artificial variable.

2.7.1 Example: Corn, Wheat, Rye

Using the same Corn, Wheat, and Rye problem that was used in the Two Phase Simplex section, our initial tableau setup will be the same except the value for the cost function will have an extremely large positive value (because this is a maximization). This algorithm was performed on a computer, so I selected the machine max integer value as M.

z	Corn	Wheat	Rye	Land	Fertilizer	Hours	e_1	a_1	RHS
1	-400	-300	-150	0	0	0	0	2147483647	0
0	1	1	1	1	0	0	0	0	45
0	4	2	1	0	1	0	0	0	165
0	40	60	20	0	0	1	0	0	2320
0	0	1	0	0	0	0	-1	$\boxed{1}$	8

The next step is to pivot on the element [5,8] and arrive at the initial setup for the next phase.

z	Corn	Wheat	Rye	Land	Fertilizer	Hours	e_1	a_1	RHS
1	-400	-1300	-150	0	0	0	2147483647	0	-8000
0	1	1	1	1	0	0	0	0	45
0	4	2	1	0	1	0	0	0	165
0	40	60	20	0	0	1	0	0	2320
0	0	1	0	0	0	0	-1	1	8

Now we have an initial basic feasible solution and we will simply proceed with the Primal Simplex Algorithm and obtain the following optimized linear program.

z	Corn	Wheat	Rye	Land	Fertilizer	Hours	e_1	a_1	RHS
1	0	0	250	400	0	0	100	2147483547	17200
0	0	1	0	0	0	0	-1	1	8
0	1	0	1	1	0	0	1	-1	37
0	0	0	-20	-40	0	1	20	-20	360
0	0	0	-3	-4	1	0	-2	2	1

Without needing to remove the artificial variable, its value has been forced to zero in the optimized linear program so it can be ignored and the best solution is still to plant 37 acres of corn and 8 acres of wheat for a profit of $17200.

2.8 Affine Scaling

Affine Scaling is an interior point algorithm for solving linear programs. The idea is that there is always a "best" direction to move the current point by. This algorithm uses that direction and iterates in steps until the objective function has reached a boundary without going over any boundary. Affine Scaling is a mathematically intense algorithm comparatively. For the demonstration, I will continue to use the Corn, Wheat, and Rye problem.

$$\begin{array}{rl}
\text{Maximize:} & 400c+300w+\ 250r \\
\text{Subject to:} & c+\quad w+\quad r\leq\ 45 \\
 & 4c+\ \ 2w+\quad r\leq 165 \\
 & 40c+\ 60w+20r\leq 2320 \\
 & c,w,r\geq 0
\end{array}$$

In matrix notation, that is:

$$\begin{array}{rl}
\text{Maximize:} & \begin{bmatrix}400 & 300 & 250 & 0 & 0 & 0\end{bmatrix}\cdot\vec{x} \\
\text{Subject to:} & \begin{bmatrix}1 & 1 & 1 & 1 & 0 & 0\\ 4 & 2 & 1 & 0 & 1 & 0\\ 40 & 60 & 20 & 0 & 0 & 1\end{bmatrix}\cdot\vec{x}=\begin{bmatrix}45\\165\\2320\end{bmatrix} \\
 & \vec{x}\geq 0
\end{array}$$

To begin, I will pick an initial basic feasible point that is not on a boundary and does not violate any constraints. The initial point I have chosen is

$$x_i=\begin{bmatrix}3\\2\\1\\39\\148\\2060\end{bmatrix}.$$

Next we will decompose x_i into its two components, the diagonal (D) of x_i and $\vec{1}$.

$$x_c = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 39 & 0 & 0 \\ 0 & 0 & 0 & 0 & 148 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2060 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

The constraints can now be written as $A \times (D \times \vec{1}) = \vec{b}$.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 4 & 2 & 1 & 0 & 1 & 0 \\ 40 & 60 & 20 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 39 & 0 & 0 \\ 0 & 0 & 0 & 0 & 148 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2060 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 45 \\ 165 \\ 2320 \end{bmatrix}$$

Now we need to scale the problem so that we are able to take a step in any direction. To do this, we will start by multiplying $A \times D$. We will call this matrix A^k.

$$\begin{aligned} A^k &= \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 4 & 2 & 1 & 0 & 1 & 0 \\ 40 & 60 & 20 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 39 & 0 & 0 \\ 0 & 0 & 0 & 0 & 148 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2060 \end{bmatrix} \\ &= \begin{bmatrix} 3 & 2 & 1 & 39 & 0 & 0 \\ 12 & 4 & 1 & 0 & 148 & 0 \\ 120 & 120 & 20 & 0 & 0 & 2060 \end{bmatrix} \end{aligned}$$

We will also multiply matrix D by our cost function C. We will call this matrix C^k.

$$C^k = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 39 & 0 & 0 \\ 0 & 0 & 0 & 0 & 148 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2060 \end{bmatrix} \times \begin{bmatrix} 400 & 300 & 250 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1200 \\ 600 \\ 150 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Now we must calculate the orthogonal projection of $\vec{x}$ onto the null space of A^k. To do this, we will use the following equation:

$$Proj_{Null(A)}\vec{x} = I_{c^k} - ((A^k)^T \cdot (A^k \cdot (A^k)^T)^{-1}) \cdot A^k$$

First calculate $A^k \cdot (A^k)^T$:

$$A^k \cdot (A^k)^T = \begin{bmatrix} 3 & 2 & 1 & 39 & 0 & 0 \\ 12 & 4 & 1 & 0 & 148 & 0 \\ 120 & 120 & 20 & 0 & 0 & 2060 \end{bmatrix} \begin{bmatrix} 3 & 12 & 120 \\ 2 & 4 & 120 \\ 1 & 1 & 20 \\ 39 & 0 & 0 \\ 0 & 148 & 0 \\ 0 & 0 & 2060 \end{bmatrix}$$

$$= \begin{bmatrix} 1535 & 45 & 620 \\ 45 & 22065 & 1940 \\ 620 & 1940 & 4272800 \end{bmatrix}$$

Take the inverse:

$$(A^k \cdot (A^k)^T)^{-1} = \begin{bmatrix} 0.0006515425 & -0.0000013205 & -0.0000000939 \\ -0.0000013205 & 0.0000453251 & -0.0000000204 \\ -0.0000000939 & -0.0000000204 & 0.0000002341 \end{bmatrix}$$

And now we have all of the information needed to calculate the projection.

$$\text{Proj}_{Null(A)}\vec{x} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 3 & 12 & 120 \\ 2 & 4 & 120 \\ 1 & 1 & 20 \\ 39 & 0 & 0 \\ 0 & 148 & 0 \\ 0 & 0 & 2060 \end{bmatrix} \times$$

$$\begin{bmatrix} 0.0006515425 & -0.0000013205 & -0.0000000939 \\ -0.0000013205 & 0.0000453251 & -0.0000000204 \\ -0.0000000939 & -0.0000000204 & 0.0000002341 \end{bmatrix} \times \begin{bmatrix} 3 & 2 & 1 & 39 & 0 & 0 \\ 12 & 4 & 1 & 0 & 148 & 0 \\ 120 & 120 & 20 & 0 & 0 & 2060 \end{bmatrix}$$

$$\text{Proj}_{Null(A)}\vec{x} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0.01554 & 0.00931 & 0.00302 & 0.07517 & 0.07955 & 0.05678 \\ 0.00931 & 0.00662 & 0.00202 & 0.05017 & 0.02608 & 0.05730 \\ 0.00302 & 0.00202 & 0.00078 & 0.02529 & 0.00645 & 0.00941 \\ 0.07517 & 0.05017 & 0.02529 & 0.99100 & -0.00762 & -0.00755 \\ 0.07955 & 0.02608 & 0.00645 & -0.00762 & 0.99280 & -0.00622 \\ 0.05678 & 0.05730 & 0.00941 & -0.00755 & -0.00622 & 0.99326 \end{bmatrix}$$

$$\text{Proj}_{Null(A)}\vec{x} = \begin{bmatrix} 0.98446 & -0.00931 & -0.00302 & -0.07517 & -0.07955 & -0.05678 \\ -0.00931 & 0.99338 & -0.00202 & -0.05017 & -0.02608 & -0.05730 \\ -0.00302 & -0.00202 & 0.99922 & -0.02529 & -0.00645 & -0.00941 \\ -0.07517 & -0.05017 & -0.02529 & 0.00900 & 0.00762 & 0.00755 \\ -0.07955 & -0.02608 & -0.00645 & 0.00762 & 0.00720 & 0.00622 \\ -0.05678 & -0.05730 & -0.00941 & 0.00755 & 0.00622 & 0.00674 \end{bmatrix}$$

This is the orthogonal projection of our initial point onto the null space of matrix A. Now that we have this, we can determine how big of a step to take in the best direction. To do this, we will multiply our null projection matrix by C^k. This gives us a vector for the best direction to take a step.

$$\text{Legal Direction} = C^k \times Proj_{Null(A)}\vec{x} = \begin{bmatrix} 1175.31248 \\ 584.55274 \\ 145.05158 \\ -124.10499 \\ -112.07441 \\ -103.92469 \end{bmatrix}$$

Next we will select our scalar for the step. This is the minimum value in the legal direction. In this case, our lambda is -124.10499. We will now use the following equation to determine our step.

$$Step = (\frac{-\alpha}{\lambda}) \times \left(C^k \times Proj_{Null(A)}\vec{x}\right)$$

Using an arbitrary alpha of 0.9 we obtain the step:

$$\text{Step} = \frac{-0.9}{-124.10499} \begin{bmatrix} 1175.31248 \\ 584.55274 \\ 145.05158 \\ -124.10499 \\ -112.07441 \\ -103.92469 \end{bmatrix} = \begin{bmatrix} 8.52328 \\ 4.23913 \\ 1.05190 \\ -0.90000 \\ -0.81276 \\ -0.75365 \end{bmatrix}$$

Now multiplying matrix D by the step and adding it to our initial point, we arrive at the solution:

$$\begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 39 & 0 & 0 \\ 0 & 0 & 0 & 0 & 148 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2060 \end{bmatrix} \times \begin{bmatrix} 8.52328 \\ 4.23913 \\ 1.05190 \\ -0.90000 \\ -0.81276 \\ -0.75365 \end{bmatrix} = \begin{bmatrix} 28.56983 \\ 10.47826 \\ 2.05190 \\ 3.90000 \\ 27.71224 \\ 507.47277 \end{bmatrix}$$

This is not an optimal solution, this is only one iteration of the Affine Scaling Algorithm. Using the Affine function written in Section 4.1.8 to iterate until the solution is essentially not moving, we arrive at the near optimal solution:

$$X_{\text{optimum}} = \begin{bmatrix} 37.49987 \\ 7.50006 \\ 0.00000 \\ 0.00000 \\ 0.00000 \\ 369.99668 \end{bmatrix}$$

2.9 Integer Programming

Linear programming is only effective when the variables in the program are all continuous. In reality, many things are not continuous, they are integers. For example, it doesn't make sense to sell 3.5637 cars or use half an egg for baking. This is where integer programming is used. The two most common algorithms are the Branch and Bound Method and the Cutting Plane Method.

2.9.1 Integer Programming Using Branch and Bound

Branch and Bound is a method that can very quickly become extremely resource consuming depending on how many variables are included in the integer program. The idea of Branch and Bound is to introduce new constraints that restrict variables to integers. I will use a modified version of the Transco problem in Section 2.4 to demonstrate this algorithm.

The initial setup:

$$\begin{array}{rrrrl} \text{Maximize:} & 1000c + & 1500t + & 2600l = & P \\ \text{Subject to:} & 2c + & 3t + & 5l \leq & 40 \\ & 0.4c + & 0.7t + & 0.1l \leq & 6 \\ & 40c + & 50t + & 100l \leq & 600 \\ & c, t, l \geq 0 & & & \end{array}$$

Before beginning, there is an issue that must be fixed. The second constraint is currently non-integer. For this method, it must be scaled to an integer so I will multiply that row by 10 and arrive at the new setup:

$$\begin{array}{rrrrl}
\text{Maximize:} & 1000c + & 1500t + & 2600l = & P \\
\text{Subject to:} & 2c + & 3t + & 5l \leq & 40 \\
 & 4c + & 7t + & 1l \leq & 60 \\
 & 40c + & 50t + & 100l \leq & 600 \\
 & c, t, l \geq 0 & & &
\end{array}$$

And now to create the tableau

z	Cars	Trucks	Locomotives	Steel	Rubber	Hours	RHS
1	-1000	-1500	-2600	0	0	0	0
0	2	3	5	1	0	0	40
0	4	7	2	0	1	0	60
0	40	50	100	0	0	1	600

Now applying the primal simplex algorithm as usual, the optimal solution is:

z	Cars	Trucks	Locomotives	Steel	Rubber	Hours	RHS
1	150.77	0	0	0	30.77	25.69	17261.538
0	-0.28	0	0	1	-0.08	-0.05	5.846
0	0.55	1	0	0	0.15	-0.00	8.308
0	0.12	0	1	0	-0.08	0.01	1.846

This will be referred to as Node 0. Here we have arrived at an issue, our solution is to produce 8.308 trucks and 1.846 locomotives, but that is not a realistic solution. This is when we start to branch and bound. At Node 0, the objective value is 17261.54 which is the current upper bound. To obtain the lower bound, we will take the floor value of all basic integer variables (in other words, Cars = 0, Trucks = 8, Locomotives = 1). Plugging these values into the objective function gives an objective value of 14600. Using the upper bound and lower bound of Node 0, we now know that our optimal integer solution will be between 14600 and 17261.54. The next step is to branch. For the first branch, we will select the variable trucks and create two nodes that use the floor and ceiling values of the RHS as constraints. Node 1 will have the constraint t $\leq \lfloor 8.308 \rfloor$ and Node 2 will have the constraint t $\geq \lceil 8.308 \rceil$.

Bringing the constraints into the original setups, we have:

Node 1:

$$\begin{array}{rrrrl}
\text{Maximize:} & 1000c + & 1500t + & 2600l = & P \\
\text{Subject to:} & 2c + & 3t + & 5l \leq & 40 \\
 & 4c + & 7t + & 1l \leq & 60 \\
 & 40c + & 50t + & 100l \leq & 600 \\
 & & t & \leq & 8 \\
 & c, t, l \geq 0 & & &
\end{array}$$

Node 2:

$$\begin{array}{rrrrrr}
\text{Maximize:} & 1000c + & 1500t + & 2600l = & P \\
\text{Subject to:} & 2c + & 3t + & 5l \leq & 40 \\
 & 4c + & 7t + & 1l \leq & 60 \\
 & 40c + & 50t + & 100l \leq & 600 \\
 & & t & \geq & 9 \\
 & c, t, l \geq 0 & & &
\end{array}$$

We can solve Node 1 using the primal simplex algorithm and arrive at the optimal tableau:

z	Cars	Trucks	Locomotives	Steel	Rubber	Hours	s_1	RHS
1	40	0	0	0	0	26	200	17200
0	0	0	0	1	0	-0.05	-0.5	6
0	3.6	0	0	0	1	-0.01	-6.5	2
0	0.4	0	1	0	0	0.01	-0.5	2
0	0	1	0	0	0	0	1	8

Node 2 requires the Two Phase Simplex Algorithm before beginning. After applying it, we arrive at the initial tableau:

z	Cars	Trucks	Locomotives	Steel	Rubber	Hours	e_1	a_1	RHS
1	-1000	-1500	-2600	0	0	0	0	0	0
0	2	0	5	1	0	0	13	3	13
0	4	0	1	0	1	0	-3	7	-3
0	40	0	100	0	0	1	150	50	150
0	0	1	0	0	0	0	9	-1	9

This is not an initial feasible solution because Rubber = -3. We will stop at this node.

Node 2 is the optimal integer solution (Cars = 0, Trucks = 8, Locomotives = 2) because all of the variables satisfy the integer restriction as well as the other constraints. There are no other combinations of production that will an integer solution between \$17200 and \$17261.53.

2.9.2 Integer Programming Using Cutting Planes Algorithm

The Cutting Planes Algorithm is an alternative to the Branch and Bound method. The idea behind the Cutting Planes Algorithm is to cut the feasible region down by adding new constraints until the boundary now contains integer values. Let's set up a general problem first.

$$\begin{array}{rl} \text{Maximize:} & \vec{C}^T\vec{x} \\ \text{Subject to:} & A\vec{x} \leq \vec{b} \\ & \vec{x} \geq 0, integer \end{array}$$

Let $S = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_k \end{bmatrix}$ and $S = b - A_{k \times n}x$.

For any feasible x $(Ax \leq 0)$, $S \geq 0$ and if all entries of A and b are integers then all entries of S will be integers.

The goal of the cutting planes algorithms is to continuously add constraints that cut off edges of the boundaries until all of the variables are integers.

To demonstrate, we will using the following linear program:

$$\begin{array}{rl} \text{Maximize:} & 30x_1 + 50x_2 \\ \text{Subject to:} & 2x_1 + \ \ 3x_2 \leq 14 \\ & 9x_1 + \ \ 6x_2 \leq 40 \\ & 4x_1 + \ \ 4x_2 \leq 20 \\ & x_1, x_2 \geq 0, integer \end{array}$$

We will set up the tableau as usual.

z	x_1	x_2	s_1	s_2	s_3	RHS
1	−30	−50	0	0	0	0
0	2	3	1	0	0	14
0	$\boxed{9}$	6	0	1	0	40
0	4	4	0	0	1	20

Now using the simplex algorithm to solve, we arrive at the following LP:

z	x_1	x_2	s_1	s_2	s_3	RHS
1	10.33	0	16.66	0	0	233.33
0	5	0	−2	1	0	12
0	1.33	0	−1.33	0	1	1.33
0	0.66	1	0.33	0	0	4.66

The next step is to choose which row we are going to cut. To do this, select the row index of the RHS with the value closest to it's lower bound + 0.5. We will choose the third row for our cut. To perform a cut, we will do the following steps:

$$\begin{aligned}1.33x_1 - 1.33s_2 + s_3 &= 1.33\\ \lfloor 1.33 \rfloor = 1, \lfloor -1.33 \rfloor = -2, \lfloor 1 \rfloor = 1, \lfloor 1.33 \rfloor &= 1\\ (1+0.33)x_1 + (-2+0.66)s_1 + s_3 &= (1+0.33)\\ x_1 + 0.33x_1 - 2s_1 + 0.66s_1 + s_3 &= 1 + 0.33\\ x_1 - 2s_1 + 3s_3 - 1 &= 0.33 - 0.33x_1 - 0.66s_2\end{aligned}$$

Every feasible solution to the IP satisfies the above equation. For any integer solution, the left hand side will be an integer. Therefore, for any integer solution, the right hand side of this equation must be an integer. For any feasible solution, $\vec{x} \geq 0$ and $\vec{s} \geq 0$, so the RHS solution must be less than or equal to $\frac{1}{3}$ for this problem ($\frac{1}{3} - \frac{1}{3}x_1 - \frac{2}{3}s_1 \leq \frac{1}{3}$).

The new constraint is thus $\frac{1}{3} - \frac{1}{3}x_1 - \frac{2}{3}s_1 \leq 0$. Putting this back into the tableau:

z	x_1	x_2	s_1	s_2	s_3	cut	RHS
1	10.33	0	16.66	0	0	0	233.33
0	5	0	−2	1	0	0	12
0	1.33	0	−1.33	0	1	0	1.33
0	0.66	1	0.33	0	0	0	4.66
0	$\boxed{-0.33}$	0	−0.66	0	0	1	−0.33

Now we can use the dual simplex algorithm to pivot on the new point and continue to optimize the problem until an integer solution has been reached.

2.9.3 Mixed Integer Linear Programming

Mixed Integer Linear Programming include both continuous and integer variables in the program. It applies the Branch and Bound Method of integer programming, but instead of restricting all variables to integers, you only restrict integer variables to integers while the continuous variables remain continuous. This method is easiest to do using computers, the only new thing to do is to apply a new property to the variables. In R, you can do this using lists. In Section 4.1.1, I have written a function that creates a matrix. There is an optional argument called integers which takes in a logical vector of TRUE and FALSE then assigns the vector to the variables respectively. The Mixed Integer Linear Program function in Section 4.1.7 can take in this object and output 2 nodes on the integer variables.

2.10 Nonlinear Programming

Everything up to this point has been for strictly first order variables. In the real world, many applications will involve higher order variables. Standard linear programming algorithms do not work for nonlinear programs, so new methods must be used.

2.10.1 Newton's Method

Newton's Method for Optimization uses basis calculus principles to approximate a solution. The idea is to find the zero of the tangent line that approximates the function, and this solution will be close to the zero of the actual function. This becomes the new point and then another iteration is done until the line is no longer moving.

If $f(x)$ is continuously differentiable at $x = x_c$ then

$$\begin{aligned} f(x) &\approx f(x_c) + f'(x_c)(x - x_c) \\ x &= \frac{-f(x_c)}{f'(x_c)} + x_c \text{ or} \\ x &= x_c - \frac{f(x_c)}{f'(x_c)} \end{aligned}$$

In the following algorithm x_c is an arbitrary starting point and ϵ is a pre-specified level of error (often machine epsilon):

$$\begin{aligned} x_n = x_c - \frac{-f(x_c)}{f'(x_c)} \\ \text{if } x_n - x_c < \epsilon \text{ then stop} \\ \text{else } x_c = x_n \\ iterate \end{aligned}$$

Now, for a twice differentiable function $h(x)$, we can apply Newton's method to find where $h'(x) = 0$.

$$\begin{aligned} \text{Choose an initial point: } x_c = x_0 \\ \text{While } \Delta x_c < \epsilon \text{ do:} \\ x_c = x_c - \frac{h'(x_c)}{h''(x_c)} \end{aligned}$$

Now expanding this into several variables...

$$\begin{aligned} \nabla f(x) \approx & \nabla f(\vec{x}_c) + \nabla^2 f(x_c^2)(\vec{x} - \vec{x}_c) \\ & \nabla f(\vec{x}_c) + \nabla^2 f(x_c^2)(\vec{x} - \vec{x}_c) = \vec{0} \\ & \nabla^2 f(x_c^2)(\vec{x} - \vec{x}_c) = -\nabla f(\vec{x}_c) \\ & H\vec{y} = -\vec{g} \end{aligned}$$

Find H=$\nabla^2 f(\vec{x}_c)$
Find $\vec{g} = -\nabla f(\vec{x}_c)$
Solve $H\vec{y} = -\vec{g}$
$\vec{X}_N = \vec{X}_c + \vec{y}$

2.10.2 Steepest Descent with Line Search

The method of steepest descent uses the gradient of the function to find a search direction. Since the idea is to descend the objective function, we use the opposite of the gradient which is ascending. The normalized direction of steepest descent is given by the negative gradient divided by the magnitude of the gradient:

$$u = \frac{-\nabla f(x)}{\|\nabla f(x)\|}$$

This direction can then be used to line search. To take a steps in the direction of the steepest descent, we use the following iterative sequence:

$$u_i = \frac{-\nabla f(x^{i-1})}{\|\nabla f(x^{i-1})\|}$$

$$x^i = x^0 + \lambda_i u_i \text{ where } \lambda_i \text{ satisfies}$$
$$\phi(\lambda_i) = min_\lambda f(x^{i-1} + \lambda_i u_i)$$

This sequence will be repeated until the difference in magnitudes is less than a specified value ϵ.

2.10.3 Lagrange Multipliers Method

The Lagrange Multiplier Method is a very common method for solving nonlinear programs because it can be easily expanded into multiple constraints. I will use the following problem to demonstrate the Lagrange method:

A rectangular beam is to be cut from a cylindrical log of radius 30cm. The strength, S of the beam is given by the formula

$$S = kwh^2$$

where w is the width of the beam, h is the height, and k is a positive constant. Find the dimensions that should be used to create the beam of greatest strength. Setting this up as an LP, we get:

$$\begin{array}{rr} \text{Maximize:} & kwh^2 \\ \text{Subject to:} & w^2 + h^2 = 30^2 \\ & k, w, h \geq 0 \end{array}$$

The first is to setup the Lagrange function which is done by multiplying the constraints by λ and subtracting them from the objective function.

So we will move all of our constraint terms to one side:

$$w^2 + h^2 - 30^2 = 0$$

Now we will multiply by λ and subtract the constraint from the objective function, giving us our Lagrange function:

$$L = kwh^2 - \lambda(w^2 + h^2 - 30^2)$$

The next step is to compute the gradient of the Lagrange function by taking the first order partial derivative of each variable and λ.

$$\begin{aligned} \frac{\delta L}{\delta w} &= kh^2 - 2\lambda w \\ \frac{\delta L}{\delta h} &= 2kwh - 2\lambda h \\ \frac{\delta L}{\delta \lambda} &= -(w^2 + h^2 - 900) \end{aligned}$$

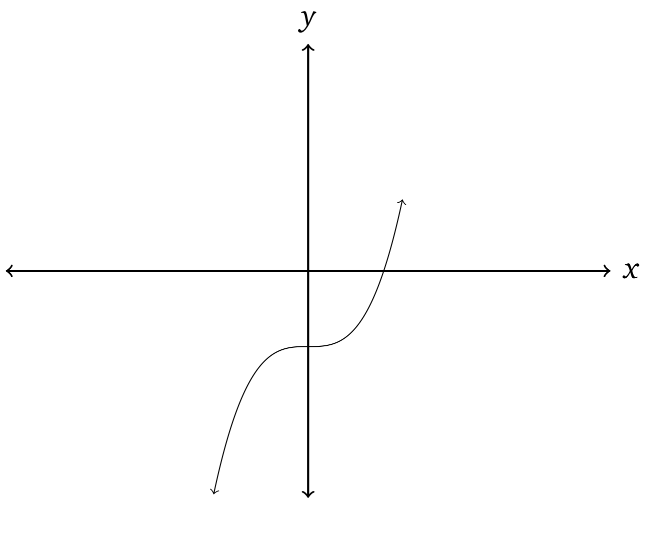
Now we have a system that can be solved for w,h, and λ. We will set each of these to zero and solve. Doing this will give us the points that maximize the function under the constraint when we plug the values back into the Lagrange function.

2.10.4 Bisection Line Search

Bisection line search requires a function that is continuous between two points. The idea of bisection is to take two points, one of which the function f evaluates to a positive value and one which f evaluates to a negative value. Somewhere between these points, there is a root (where f evaluates to 0).

To demonstrate, I will use a simple equation:

$$f(x) = x^3 - 1$$



Now I will choose two points. One where the function is negative and one where it is positive. I will select $x_L = 0.5$ and $x_U = 2$ and while this point is arbitrary, we must choose the point carefully because if we include the inflection point (as seen in the graph) then we will run into issues with bisecting. $f(0.5) = -0.9844$ and $f(2) = 7$ so we have one point that evaluates to negative and one that evaluates positive. The next step is to find the midpoint between these two and evaluate. The point x_m will be used to denote the midpoint.

$$x_m = \frac{0.5+2}{2} = 1.25$$

And now evaluate at the point x_m

$$f(1.25) = (1.25)^3 - 1 = 0.953125$$

Since $f(x_m)$ evaluated to a positive value, the point will replace x_U as the positive entry. We will iterate again using the new point to calculate another midpoint.

$$x_m = \frac{0.5+1.25}{2} = 0.875$$

And evaluating at the new x_m

$$f(0.875) = (0.875)^3 - 1 = -0.33$$

Since this is negative, we will replace the value of x_L with it iterating again with $x_L = 0.875$

$$x_m = \frac{0.875+1.25}{2} = 1.0625$$

And evaluating again...

$$f(1.0625) = (1.0625)^3 - 1 = 0.1995$$

Continue to iterate until we reach the point where $f(x_m) \approx 0$, in this particular case it is where $x_m^* \approx 1$. This is the bisection method to approximate a root of a function.

2.10.5 Golden Section Search

Slightly modifying this approach, we can use it to find the a minimum or maximum of a function. This is called the Golden Section Search because it uses the golden ratio to specify the length between the two points.

Assuming $f(x)$ is unimodal between $[x_L, x_U]$, set $x_1 = x_L + \phi^2 L$ and $x_2 = x_U + \phi L$ where $L = x_U - x_L$ and $\phi = \frac{1+\sqrt{5}}{2}$. Evaluate $f(x_1)$ and $f(x_2)$.

$$\begin{aligned}
\text{If } f(x_1) > f(x_2) \text{ then do:} \\
\text{set } x_L = x_1, x_1 = x_2, L = x_U - x_L, x_2 = x_L + \phi L \\
\text{else if } f(x_2) > f(x_1) \text{ then do:} \\
\text{set } x_U = x_2, x_2 = x_1, L = x_U - x_L, x_1 = x_L + \phi^2 L
\end{aligned}$$

Repeat this until $L < \epsilon$ where ϵ is a pre-specified satisfactory error range (in other words how far off you are willing to be from the true value). Once this condition has been meet, $x^* \approx \frac{x_L + x_U}{2}$.

2.10.6 Karush-Kuhn-Tucker Conditions

For a nonlinear program, the Karush-Kuhn-Tucker (KKT) conditions must be met otherwise the NLP cannot be considered optimized. The idea behind the KKT conditions is that if x^* is a maximum for f then there is no valid direction of increase from $\vec{x}_c$. The KKT conditions for optimization are:

If $\vec{x}^*$ is a maximum, then

$$\begin{aligned}
\text{Condition 1: } \nabla f(\vec{x}^*) = \lambda_1 \nabla g_1(\vec{x}^*) + \lambda_2 \nabla g_2(\vec{x}^*) + \ldots + \lambda_k \nabla g_k(\vec{x}^*) \Big\} \text{ Stationarity} \\
\text{Condition 2: } \left.\begin{matrix} g_1(\vec{x}^*) \leq 0 \\ g_2(\vec{x}^*) \leq 0 \\ \vdots \\ g_k(\vec{x}^*) \leq 0 \end{matrix}\right\} \text{ Feasibility} \\
\text{Condition 3: } \left.\begin{matrix} \lambda_1 g_1(\vec{x}^*) = 0 \\ \lambda_2 g_2(\vec{x}^*) = 0 \\ \vdots \\ \lambda_k g_k(\vec{x}^*) = 0 \end{matrix}\right\} \text{ Complementary Slackness} \\
\text{Condition 4: } \lambda_1, \lambda_2, \ldots, \lambda_k \geq 0 \Big\} \text{ Dual Feasibility}
\end{aligned}$$

If all $\lambda = 0$ then the NLP is at a boundary and there are no more legal moves. If any $\lambda_i > 0$ then the NLP is feasible and still has legal directions it can move in.

3 Application Problems

3.1 Baking with Pumpkin Spice

3.1.1 Optimizing Ingredient Allocations

Here we are asked to maximize the profits for a bakery using the following recipes:

Pumpkin Spice Cupcakes	
32	per batch
\$1	each
16	ounces sugar
15	ounces pumpkin
14	ounces all-purpose flour
0.75	teaspoon salt
4	teaspoons pumpkin pie spice
2	ounces corn starch
2	teaspoons baking powder
4	eggs
6	ounces butter

Pumpkin Spice Cookies	
36	per batch
\$0.75	each
12	ounces sugar
8	ounces pumpkin
0.5	teaspoon salt
3	teaspoons pumpkin pie spice
20	ounces all-purpose flour
1	teaspoon baking powder
1	egg
4	ounces butter

Pumpkin Spice Bagels	
8	per batch
\$2.75	each
3	ounces sugar
6	ounces pumpkin
24	ounces all-purpose flour
1	teaspoon salt
3.25	teaspoons pumpkin pie spice
0.25	ounces active dry yeast

Table 1: Pumpkin Spice Recipes

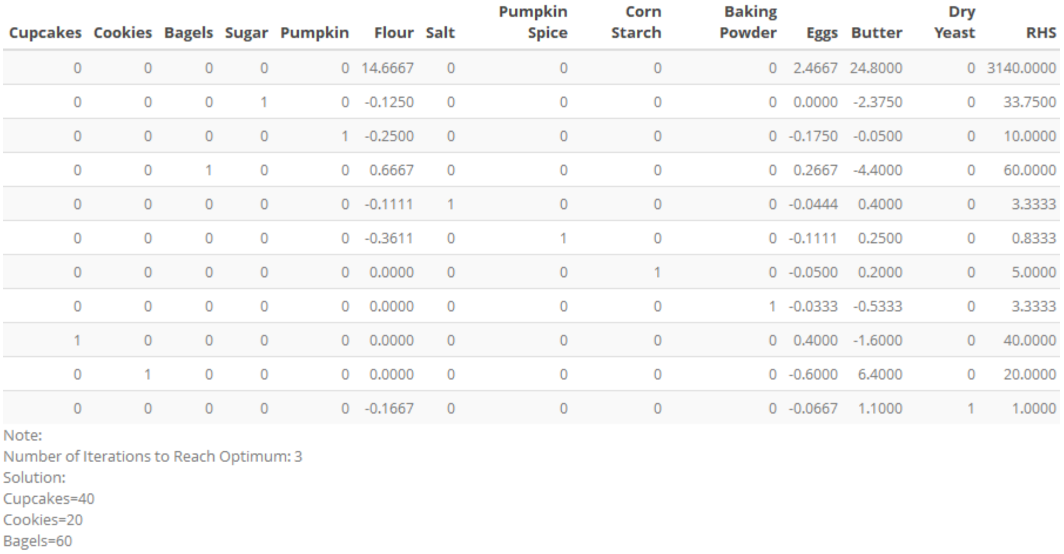
100 pounds	sugar	10 pounds	corn starch
80 pounds	pumpkin	20 ounces	baking powder
150 pounds	all-purpose flour	15 dozen	eggs
20 ounces	salt	20 pounds	butter
70 ounces	pumpkin pie spice	16 ounces	active dry yeast

Table 2: Ingredients on Hand

The first step to solving this linear program is to set it up in a manipulable matrix. One thing to watch out for is the units on each of the ingredients as they are occasionally different between cupcakes, cookies, and bagels. I converted the units in each recipe to the units for the ingredients on hand. X1=Cupcakes, X2=Cookies, X3=Bagels, S1=Sugar, S2=Pumpkin, S3=Flour, S4=Salt, S5=Pumpkin Pie Spice, S6=Corn Starch, S7=Baking Powder, S8=Eggs, S9=Butter, S10=Yeast. This is the tableau:

z	X_1	X_2	X_3	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	RHS
1	-32	-27	-22	0	0	0	0	0	0	0	0	0	0	0
0	1	3/4	3/16	1	0	0	0	0	0	0	0	0	0	100
0	15/16	8/16	6/16	0	1	0	0	0	0	0	0	0	0	80
0	14/16	20/16	24/16	0	0	1	0	0	0	0	0	0	0	150
0	1/8	1/12	1/6	0	0	0	1	0	0	0	0	0	0	20
0	2/3	1/2	13/24	0	0	0	0	1	0	0	0	0	0	70
0	1/8	0	0	0	0	0	0	0	1	0	0	0	0	10
0	1/3	1/6	0	0	0	0	0	0	0	1	0	0	0	20
0	4	1	0	0	0	0	0	0	0	0	1	0	0	180
0	6/16	4/16	0	0	0	0	0	0	0	0	0	1	0	20
0	0	0	1/4	0	0	0	0	0	0	0	0	0	1	16

Inputting the linear program into the R script (written in Section 4.1.1 and Section 4.1.5) returns the following optimal solution:



Cupcakes	Cookies	Bagels	Sugar	Pumpkin	Flour	Salt	Pumpkin Spice	Corn Starch	Baking Powder	Eggs	Butter	Dry Yeast	RHS
0	0	0	0	0	14.6667	0	0	0	0	2.4667	24.8000	0	3140.0000
0	0	0	1	0	-0.1250	0	0	0	0	0.0000	-2.3750	0	33.7500
0	0	0	0	1	-0.2500	0	0	0	0	-0.1750	-0.0500	0	10.0000
0	0	1	0	0	0.6667	0	0	0	0	0.2667	-4.4000	0	60.0000
0	0	0	0	0	-0.1111	1	0	0	0	-0.0444	0.4000	0	3.3333
0	0	0	0	0	-0.3611	0	1	0	0	-0.1111	0.2500	0	0.8333
0	0	0	0	0	0.0000	0	0	1	0	-0.0500	0.2000	0	5.0000
0	0	0	0	0	0.0000	0	0	0	1	-0.0333	-0.5333	0	3.3333
1	0	0	0	0	0.0000	0	0	0	0	0.4000	-1.6000	0	40.0000
0	1	0	0	0	0.0000	0	0	0	0	-0.6000	6.4000	0	20.0000
0	0	0	0	0	-0.1667	0	0	0	0	-0.0667	1.1000	1	1.0000

Note:
Number of Iterations to Reach Optimum: 3
Solution:
Cupcakes=40
Cookies=20
Bagels=60

Figure 2: Output for the Pumpkin Spice Dilemma

To maximize profits given the ingredients on hand, the bakery should make 40 cupcakes, 20 cookies, and 60 bagels. They will have used all of their flour, eggs, and butter but will have leftovers for the rest of the ingredients. Taking a look into the current prices of ingredients, we will determine whether any of them are worth buying. We will use sensitivity analysis to make these decisions.

3.1.2 Sensitivity Analysis

Ingredient Prices	
flour:	\$0.87/16 ounces
corn starch:	\$1.64/16 ounces
sugar:	\$0.60/16 ounces
pumpkin pie spice:	\$0.57/teaspoon
pumpkin:	\$2.10/16 ounces
butter:	\$2.88/48 ounces
yeast:	\$6.06/16 ounces
salt:	\$0.89/26 ounces
baking powder:	\$1.48/8 ounces
eggs:	\$2.15/dozen

Table 3: Prices for Additional Ingredients

I will start by taking a look at flour since it is the first non-basic variable in the optimal tableau (meaning that there is no flour left). Its shadow price of \$14.67 tells us that we can increase profits by \$14.67 for each pound of flour that is added. Since flour is only \$0.87 per pound, each pound that we buy will net \$13.80. Only 2 pounds of flour should be purchased at first because if we purchase 3 pounds then we will not have enough in pumpkin spice.

z	X_1	X_2	X_3	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	RHS	ΔRHS
1	0	0	0	0	0	14.67	0	0	0	0	2.47	24.80	0	$\boxed{3169.33}$	29.33
0	0	0	0	1	0	–0.12	0	0	0	0	0.00	–2.38	0	33.50	–0.25
0	0	0	0	0	1	–0.25	0	0	0	0	–0.17	–0.05	0	9.50	–0.50
0	0	0	1	0	0	0.67	0	0	0	0	0.27	–4.40	0	61.33	1.33
0	0	0	0	0	0	–0.11	1	0	0	0	–0.04	0.40	0	3.11	–0.22
0	0	0	0	0	0	–0.36	0	1	0	0	–0.11	0.25	0	0.11	–0.72
0	0	0	0	0	0	0.00	0	0	1	0	–0.05	0.20	0	5.00	0.00
0	0	0	0	0	0	0.00	0	0	0	1	–0.03	–0.53	0	3.33	0.00
0	1	0	0	0	0	0.00	0	0	0	0	0.40	–1.60	0	40.00	0.00
0	0	1	0	0	0	0.00	0	0	0	0	–0.60	6.40	0	20.00	0.00
0	0	0	0	0	0	–0.17	0	0	0	0	–0.07	1.10	1	0.67	–0.33

After purchasing 2 pounds of flour, the profit for the bakery is now
\$3169.33 - \$1.74 = \$3167.59
This is an increase of \$27.59 for spending \$1.74 on more flour.

If instead of purchasing more flour, the bakery purchased 6 more pounds of butter at a cost of $2.88 per 3 pounds then they would have the following tableau.

z	X_1	X_2	X_3	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	RHS	ΔRHS
1	0	0	0	0	0	14.67	0	0	0	0	2.47	24.80	0	$\boxed{3288.80}$	148.80
0	0	0	0	1	0	−0.12	0	0	0	0	0.00	−2.38	0	19.50	−14.25
0	0	0	0	0	1	−0.25	0	0	0	0	−0.17	−0.05	0	9.70	−0.30
0	0	0	1	0	0	0.67	0	0	0	0	0.27	−4.40	0	33.60	−26.40
0	0	0	0	0	0	−0.11	1	0	0	0	−0.04	0.40	0	5.73	2.40
0	0	0	0	0	0	−0.36	0	1	0	0	−0.11	0.25	0	2.33	1.50
0	0	0	0	0	0	0.00	0	0	1	0	−0.05	0.20	0	6.20	1.20
0	0	0	0	0	0	0.00	0	0	0	1	−0.03	−0.53	0	0.13	−3.20
0	1	0	0	0	0	0.00	0	0	0	0	0.40	−1.60	0	30.40	−9.60
0	0	1	0	0	0	0.00	0	0	0	0	−0.60	6.40	0	58.40	38.40
0	0	0	0	0	0	−0.17	0	0	0	0	−0.07	1.10	1	7.60	6.60

The profit is now $3288.80 - $5.76 = $3283.04 which is an increase of $143.04 for spending just $5.76 on butter. The bakery will now make 33.60 batches of bagels, 30.40 batches of cupcakes, and 58.40 batches of cookies.

All of this is assuming that partial batches can be made.

3.2 Minimizing Shipping Costs

A building supply has two warehouses in town, one on the east side of town and one on the west.

The office receives orders from two customers for 3/4-inch plywood. Customer *A* needs fifty sheets, and Customer *B* needs seventy sheets.

The warehouse on the east side of town has eighty sheets in stock; the west-side warehouse has forty-five sheets in stock.

Delivery costs per sheet are as follows: $0.50 from the eastern warehouse to Customer *A*, $0.60 from the eastern warehouse to Customer *B*, $0.40 from the western warehouse to Customer *A*, and $0.55 from the western warehouse to Customer *B*.

We have been asked to find the shipping arrangement which minimizes costs.

3.2.1 Optimizing of Costs

The first step was to set up the linear program. Here is how the LP was organized with the objective function and its constraints. The variables *E* and *W* are the east and west warehouses respectively and the *A* and *B* subscripts are the customers that are being shipped to. So E_A is the number of shipments to Customer A coming from the East warehouse.

$$\begin{array}{rl}
\text{Minimize:} & 0.5E_A + 0.6E_B + 0.4W_A + 0.55W_B \\
\text{Subject to:} & \begin{array}{ccccccccc}
E_A + & E_B & & & & & \le 80 \\
 & & & W_A + & & W_B & \le 45 \\
E_A & & + & W_A & & & \ge 50 \\
 & E_B & & & + & W_B & \ge 70
\end{array} \\
& E_A, E_B, W_A, W_B \ge 0
\end{array}$$

The first thing to do is to convert the program to a maximization problem so that simplex algorithms may be used. The second issue arises at the greater than or equal to constraints. To apply simplex algorithms to the linear program, all constraints must be strictly less than or equal to. To fix these problems, we multiply the objective function and the two greater than or equal to constraints by -1 to convert the problem into a usable state. Doing so, our new linear program becomes:

$$\begin{array}{rl}
\text{Maximize:} & -0.5E_A - 0.6E_B - 0.4W_A - 0.55W_B \\
\text{Subject to:} & \begin{array}{ccccccccc}
E_A + & E_B & & & & & \le & 80 \\
 & & & W_A + & & W_B & \le & 45 \\
-E_A & & - & W_A & & & \le & -50 \\
 & -E_B & & & - & W_B & \le & -70
\end{array} \\
& E_A, E_B, W_A, W_B \ge 0
\end{array}$$

Now using this to set up the tableau:

z	E_A	E_B	W_A	W_B	S_1	S_2	e_1	e_2	RHS
1	0.5	0.6	0.4	0.6	0	0	0	0	0
0	1	1	0	0	1	0	0	0	80
0	0	0	1	1	0	1	0	0	45
0	-1	0	$\boxed{-1}$	0	0	0	1	0	-50
0	0	-1	0	-1	0	0	0	1	-70

Looking at the initial tableau, we can see that it is not initial primal feasible because there are no negative values in the objective row. However, there are negatives in the RHS so this is initial dual feasible. To solve this LP, we will use the dual simplex algorithm. For the first iteration, W_A will be our entering variable and e_1 will be our leaving variable.

z	E_A	E_B	W_A	W_B	S_1	S_2	e_1	e_2	RHS
1	0.1	0.6	0	0.6	0	0	0.4	0	−20
0	1	1	0	0.0	1	0	0	0	80
0	$\boxed{-1}$	0	0	1	0	1	1	0	−5
0	1	−0	1	0	0	0	1	0	50
0	0	−1	0	−1	0	0	0	1	−70

For the second iteration, E_A will be our entering variable and S_2 will be the leaving variable.

	z	E_A	E_B	W_A	W_B	S_1	S_2	e_1	e_2	RHS
	1	0	0.6	0	0.65	0	0.1	0.5	0	−20.5
$[H]$	0	0	1	0	1	1	1	1	0	75
	0	1	0	0	−1	0	1	1	0	5
	0	0	0	1	1	0	1	0	0	45
	0	0	$\boxed{-1}$	0	−1	0	0	0	1	−70

For the third iteration, E_B will be our entering variable and e_2 will be the leaving variable.

	z	E_A	E_B	W_A	W_B	S_1	S_2	e_1	e_2	RHS
	1	0	0	0	0.05	0	0.1	0.5	0.6	−62.5
$[H]$	0	0	0	0	0	1	1	1	1	5
	0	1	0	0	−1	0	−1	−1	0	5
	0	0	0	1	1	0	1	0	0	45
	0	0	1	0	1	0	0	0	−1	70

This is our optimized linear program because there are no more negative values in the RHS. We cannot proceed further with the primal simplex algorithm because there are no negative values in the objective row either.

According to the optimized solution, the shipping arrangement that will minimize costs is as follows:
Ship 5 sheets from the East warehouse to Customer A.
Ship 70 sheets from the East warehouse to Customer B.
Ship 45 sheets from the West warehouse to Customer A.
There will be 5 sheets remaining in the East warehouse.
This will cost a total of \$62.50.

3.3 Minimizing Production Costs

A smelting plant uses three different smelting procedures, each of which produces various amounts of each of their four grades of steel (grades 1, 2, 3, and 4) as well as waste in the form of slag.

The production methods cost \$20, \$20, and \$30 per ton of iron processed.

For each ton of iron processed:
method 1 produces grades 1, 2, 3, 4 and slag in a ratio of 1 : 2 : 1 : 3 : 1;
method 2 produces grades 1, 2, 3, 4 and slag in a ratio of 1 : 1 : 2 : 1 : 1; and
method 3 produces grades 1, 2, 3, 4 and slag in a ratio of 4 : 1 : 2 : 2 : 1.

(There is always one ton of total output per ton of iron used; thus, for each ton of iron processed, method 1 produces $^1/_8$ ton of grade 1, $^1/_4$ ton of grade 2, $^1/_8$ ton of grade 3, $^3/_8$ ton of grade 4, and $^1/_8$ ton of slag.)

The plant has orders this week from four clients. Their orders are shown in table 4.

	Iron Grade			
Client	1	2	3	4
1	1 ton	2 tons		
2		1 ton	3 tons	3 tons
3	2 tons		4 tons	1 ton
4	1 ton	1 ton	1 ton	1 ton

Table 4: Client Orders for Various Grades of Iron

Each ton of iron takes 1.75, 1.5, or 1.2 hours of time for each of the process, respectively, and they have 39 hours of production time available.

How many tons of iron should they process with each of their processes to minimize their production cost?

3.3.1 Optimizing Iron Processing

The first step to optimizing this linear program was to determine how much of each iron grade was needed. Based on the client orders, we will need 4 tons of iron grade 1, 4 tons of iron grade 2, 8 tons of iron grade 3, and 5 tons of iron grade 4. Slag can essentially be ignored, but we will keep it in the program to tell us how much slag will be produced if the company is interested in that. The linear program is set up as follows:

$$\begin{array}{rl}
\text{Minimize:} & 20M_1 + 20M_2 + 30M_3 \\
\text{Subject to:} & \frac{1}{8}M_1 + \frac{1}{6}M_2 + \frac{2}{5}M_3 \geq 4 \\
 & \frac{1}{4}M_1 + \frac{1}{6}M_2 + \frac{1}{10}M_3 \geq 4 \\
 & \frac{1}{8}M_1 + \frac{1}{3}M_2 + \frac{1}{5}M_3 \geq 8 \\
 & \frac{3}{8}M_1 + \frac{1}{6}M_2 + \frac{1}{5}M_3 \geq 5 \\
 & \frac{1}{8}M_1 + \frac{1}{6}M_2 + \frac{1}{10}M_3 \geq 0 \\
 & 1\frac{3}{4}M_1 + 1\frac{1}{2}M_2 + 1\frac{1}{5}M_3 \leq 39 \\
 & M_1, M_2, M_3 \geq 0
\end{array}$$

The next step is to convert this to a maximization problem.

$$\begin{array}{rl}
\text{Maximize:} & -20M_1 - 20M_2 - 30M_3 \\
\text{Subject to:} & -\frac{1}{8}M_1 - \frac{1}{6}M_2 - \frac{2}{5}M_3 \leq -4 \\
 & -\frac{1}{4}M_1 - \frac{1}{6}M_2 - \frac{1}{10}M_3 \leq -4 \\
 & -\frac{1}{8}M_1 - \frac{1}{3}M_2 - \frac{1}{5}M_3 \leq -8 \\
 & -\frac{3}{8}M_1 - \frac{1}{6}M_2 - \frac{1}{5}M_3 \leq -5 \\
 & -\frac{1}{8}M_1 - \frac{1}{6}M_2 - \frac{1}{10}M_3 \leq 0 \\
 & 1\frac{3}{4}M_1 + 1\frac{1}{2}M_2 + 1\frac{1}{5}M_3 \leq 39 \\
 & M_1, M_2, M_3 \geq 0
\end{array}$$

And now converting the linear program into a tableau:

	z	M_1	M_2	M_3	e_1	e_2	e_3	e_4	s_1	RHS
	1	20	20	30.0	0	0	0	0	0	0
	0	−0.125	−0.1667	$\boxed{-0.4}$	1	0	0	0	0	−4
$[H]$	0	−0.250	−0.1667	−0.1	0	1	0	0	0	−4
	0	−0.125	−0.3333	−0.2	0	0	1	0	0	−8
	0	−0.375	−0.1667	−0.1	0	0	0	1	0	0
	0	1.25	1.5	1.2	0	0	0	0	1	39

The tableau is not initial primal feasible, but it is initial dual feasible so we will proceed with the dual simplex algorithm. Our first iteration will select Method 3 as the entering variable and e_1 as the leaving variable.

	z	M_1	M_2	M_3	e_1	e_2	e_3	e_4	s_1	RHS
	1	10.6250	7.5000	0	75.00	0	0	0	0	−300
	0	0.3125	0.4167	1	−2.50	0	0	0	0	10
$[H]$	0	$\boxed{-0.2188}$	−0.1250	0	−0.25	1	0	0	0	−3
	0	−0.0625	−0.2500	0	−0.50	0	1	0	0	−6
	0	−0.3438	−0.1250	0	−0.25	0	0	1	0	1
	0	1.375	1	0	3.00	0	0	0	1	27

For the second iteration, Method 1 will be the entering variable and e_2 will be the leaving variable.

	z	M_1	M_2	M_3	e_1	e_2	e_3	e_4	s_1	RHS
	1	0	1.4286	0	62.8571	48.5714	0	0	0	−445.714
	0	0	0.2381	1	−2.8571	1.4286	0	0	0	5.714
[H]	0	1	0.5714	0	1.1429	−4.5714	0	0	0	13.714
	0	0	$\boxed{-0.2143}$	0	−0.4286	−0.2857	1	0	0	−5.143
	0	0	0.0714	0	0.1429	−1.5714	0	1	0	5.714
	0	0	0.2143	0	1.4286	6.2857	0	0	1	8.1429

For the third iteration, Method 2 will be the entering variable and e_3 will be the leaving variable.

	z	M_1	M_2	M_3	e_1	e_2	e_3	e_4	s_1	RHS
	1	0	0	0	60	46.667	6.667	0	0	−480
	0	0	0	1	−3.333	1.111	1.111	0	0	0
[H]	0	1	0	0	0	−5.333	2.667	0	0	0
	0	0	1	0	2	1.333	−4.667	0	0	24
	0	0	0	0	0	−1.667	0.333	1	0	4
	0	0	0	0	1	6	1	0	1	3

This is the optimized linear program because there are no more negative entries in the RHS. The results of this optimization tell us that the best course of action is to process 24 tons of iron using only method 2. This will result in an excess of 4 tons of grade 4 steel and 3 unused hours of processing time while cost $480.

4 Projects

4.1 Linear Programming in R

4.1.1 Creating a Matrix

Below I have written two ways of creating a matrix in R. The first method requires the coefficients to be entered separately in individual vectors where they will then be combined into a matrix. This function assumes that all of the constraints are in ≤ form.

```
@param C = objective function
@param ... = constraints
createMatrix <-
   function(C,...,varnames=NULL,integers=NULL,basic=NULL){
  x <- list(...)
  M <- as.matrix(C)
  for(i in 1:length(x)){
    M <- cbind(M,x[[i]])
  }
  M <- t(M)
  I <- diag(nrow=nrow(M),ncol=nrow(M))
  I <- apply(I,2,lag)
  I[1,] <- 0
  I <- I[,-ncol(I)]
  RHS <- as.matrix(M[,ncol(M)])
  M <- M[,-ncol(M)]
  M <- cbind(M,I,RHS)
  M[1,] <- -1*M[1,] # making objective row negative
  tab <- cbind(c(1,rep(0,nrow(M)-1)), M)
  tab -> M
  if(is.null(varnames)){
    varnames <-
       c('z',paste0("x",1:(ncol(M)-length(x)-2)),paste0("s",1:(length(x))),'RHS')
  }
  if(is.null(integers)){
    integers <- rep(F,ncol(M))
  }
  variables <- list(varnames = varnames,
                    integers = integers,
                    basic = basic)
  M <- unname(M)
  return(M)
}
```

The second method is much quicker if you already have the matrix ready to be input. This time I am using the Corn, Wheat, and Rye problem from Section 2.1.3.

```
A <- matrix(c(-400,-300,-150,0,0,0,0,
1,1,1,1,0,0,45,
4,2,1,0,1,0,165,
40,60,20,0,0,1,2320), nrow=4, byrow=T)
varnames <- c(’Corn’,’Wheat’,’Rye’,’Land’,’Fertilizer’,’Hours’,’RHS’)
```

4.1.2 Pivoting

Row operations in R are very easy to do. To perform a pivot simply divide the pivot row by the pivot element, find the multiplier required to reduce all other rows in the pivot column to zero, then multiply and subtract. Here is the pivot function I wrote. The input is the matrix, the column (entering variable), and the row (leaving variable). This code can also be found in the simplex algorithm above.

```
pivot <- function(A,r,c){
  if(A[r,c]==0){
    stop("Cannot Pivot on 0")
  }
  varnames <- dimnames(A)[[2]]
  colnames(A) = c(1:ncol(A))
  rownames(A) = c(1:nrow(A))
  A[r,] <- A[r,]/A[r,c] # reduce pivot row
  # row operations to pivot at [r,c]
  for(i in 1:nrow(A)){
    if(rownames(A)[i]==rownames(A)[r]){ # skip pivot row
      next
    }
    A[i,] <- A[i,] - A[i,c]*A[r,]
  }
  if(!is.null(varnames)){
    colnames(A)<-varnames
  }
  return(A)
}
```

4.1.3 Initial Basic Primal Feasible Solution

This function can be used to check for an initial basic primal feasible solution.

```
checkIBFS <- function(A){
  RHS <- A[2:nrow(A),ncol(A)] # Selecting RHS
  basic <- rep(FALSE,ncol(A)) # Setting all variables to non-basic
  for(i in 1:ncol(A)){ # Identifying basic variables
    if(identical(unique(A[,i]),c(0,1)) |
       identical(unique(A[,i]),c(1,0))){
      basic[i] = TRUE
    }
  }
  if(ncol((A[,which(basic==TRUE)])==ncol(diag(nrow(A))) & all(RHS >=
     0)){
    return(TRUE) # is IBFS
  } else {
    return(FALSE) # is not IBFS
  }
}
```

4.1.4 Checking Basis

This function will locate the basic variables within a matrix and return their RHS values as a vector.

```
checkBasis <- function(M){
  basis <- c()
  varnames <- c()
  for(i in 1:ncol(M)){
    if(identical(unique(M[,i]),c(0,1)) |
       identical(unique(M[,i]),c(1,0))){
      r <- which.max(M[,i])
      varnames <- append(varnames,colnames(M)[i])
      basis <- append(basis, M[r,ncol(M)])
    }
  }
  if(length(basis)!=nrow(M)){
    warning(paste("Incomplete Basis. Expected",nrow(M),"basic
       variables, but only",length(basis),"were found."))
  }
  basis <- setNames(basis,varnames)
  return(basis)
}
```

4.1.5 Primal Simplex Algorithm

The best approach to automating a simplex algorithm is to use Bland's rule to prevent infinite loops from occurring in the code. Below is a function I wrote that solves linear programs using Bland’s anti-cycling algorithm. I have placed comments throughout the code to attempt to explain what is going on at specific points.

```
‘‘‘{r message=FALSE, warning=FALSE}
# Step 1: Select a viable entering variable (column)
# Step 2: Select a viable leaving variable (row)
# Step 3: Pivot at [row, column]
# Step 4: Determine if optimal. Return to step 1 if not optimal.
simplex_primal <- function(lp) {
  if(typeof(lp)=="list"){
    A <- lp[[1]]
    if(!is.null(lp[[2]][1])){
      varnames <- lp$variables$varnames
    } else {
      varnames<-c("z",paste0("x",1:(ncol(A)-2)),"RHS")
    }
  } else {
    A <- lp
    varnames <- c(’z’,paste0("x",1:(ncol(A)-2)),’RHS’)
    objVar=1
  }
  if(any(A[,ncol(A)]<0)){
    stop("Not Initial Basic Primal Feasible. Negative in RHS.")
  }
  if(all(A[1,])>0){
    stop("No negative coefficients in cost function.")
  }
  # setting column and row names to numeric values
  colnames(A) = c(1:ncol(A))
  rownames(A) = c(1:nrow(A))
  count <- 0
  repeat{
    count <- count + 1
    # Step 1 -------------------------------------------------------
    # Locating column of first negative coefficient in objective
       function
    for(i in 1:ncol(A)){
      x <- A[2:nrow(A),i]
      if(A[1,i] < 0 & any(x > 0)){ # checking obj coefficient is
          negative and at least 1 row below it is positive
        which(colnames(A)==i) -> c
        break
      }
    }
    # Step 2 -------------------------------------------------------
    # Dividing entering variable by RHS and selecting row with lowest
       value
    rvec <- c(NA) # NA so objective function is ignored
    for(i in 2:nrow(A)){
      if(A[i,c] > 0){
        rvec <- append(rvec,(1/((A[i,c] / A[i,ncol(A)])))) # add row
            to row vector
      } else if(A[i,c] <= 0){
        rvec <- append(rvec,NA) # non-negativity constraint
      }
    }
    if(isTRUE(all.equal(rvec,NA))){ # Entering variable but no leaving
       variable
      break
    }
    which.min(rvec) -> r # select minimum of row vector
    # Step 3 ----------------------------------------------------------
    A[r,] <- A[r,]/A[r,c] # reduce pivot row
    # row operations to pivot at [r,c]
    for(i in 1:nrow(A)){
      if(rownames(A)[i]==rownames(A)[r]){ # skip pivot row
        next
      }
      A[i,] <- A[i,] - A[i,c]*A[r,]
    }
    # Step 4 ----------------------------------------------------------
    # breaks if all coefficients in objective row are 0 or positive
    if(all(t(A[1,]) >= 0)){
      break
    }
    if(count==50){ #
      stop("Too Many Iterations")
    }
  }
  return(A)
}
‘‘‘
```

This function will take in a pre-built matrix A (from the script I wrote to create a linear program in R in Section 4.1.1) and return the matrix and a table output. I have used the Kable package to provide better looking output compared to base R.

4.1.6 Dual Simplex Algorithm

To create the dual simplex algorithm, I slightly modified the code for the primal simplex algorithm. The only real difference is that it selects the row first then the column and the checks have changed to suit the dual simplex algorithm constraints.

```
# Step 1: Select a viable leaving variable (row)
# Step 2: Select a viable entering variable (column)
# Step 3: Pivot at [row, column]
# Step 4: Determine if optimal. Return to step 1 if not optimal.

simplex_dual <- function(lp){
  if(typeof(lp)=="list"){
    A <- lp[[1]]
    varnames <- lp$variables$varnames
    identity <- lp[[3]]
  } else {
    A <- lp
  }
  if(!any(A[,ncol(A)]<0)){
    stop("Not Initial Basic Dual Feasible. No negatives in RHS.")
  }
  if(all(A[1,]<0)){
    stop("No positive coefficients in cost function.")
  }
  # setting column and row names to numeric values
  colnames(A) = c(1:ncol(A))
  rownames(A) = c(1:nrow(A))
  count <- 0
  repeat{
    count <- count + 1
    # Step 1 ------------------------------------------------------------
    # Choosing leaving variable
    for(i in 2:nrow(A)){
      x <- A[i,ncol(A)]
      if(A[i,ncol(A)] < 0 & any(x < 0)){
        which(rownames(A)==i) -> r
        break
      }
    }
    # Step 2 ------------------------------------------------------------
    # Dividing entering variable by RHS and selecting row with lowest
        value

    ##### Change to choose first viable column
    cvec <- c() # NA so objective function is ignored
    for(i in 1:(ncol(A)-1)){
      if(A[r,i] < 0){
        cvec <- append(cvec,(A[1,i]/A[r,i])) # add row to row vector
      } else if(A[r,i] >= 0){
        cvec <- append(cvec,NA) # negativity constraint
      }
    }
    if(isTRUE(all.equal(cvec,NA))){ # No leaving variable
      break
    }
    which.max(cvec) -> c # select column closest to zero
    # Step 3 -------------------------------------------------------
    A[r,] <- A[r,]/A[r,c] # reduce pivot row
    # row operations to pivot at [r,c]
    for(i in 1:nrow(A)){
      if(rownames(A)[i]==rownames(A)[r]){ # skip pivot row
        next
      }
      A[i,] <- A[i,] - A[i,c]*A[r,]
    }
    # Step 4 -------------------------------------------------------
    # breaks if all coefficients in objective row are 0 or negative
    if(all(A[2:nrow(A),ncol(A)]>=0)){
      break
    }
    if(count==50){ #
      stop("Too Many Iterations")
    }
  }
  return(A)
}
```

4.1.7 Mixed Integer Linear Programming Using Branch and Bound

The code for mixed integer linear programming is significantly more complex than the primal and dual simplex algorithms. The mixed integer function has to check for many more conditions in its process. It implements code from several other functions such as checkIBFS (Section 4.1.3), primal simplex (Section 4.1.5), and two phase simplex (Not coded separately yet) and also determines if the specified integer variables are indeed integers. This function requires a special R object called a list-column. The necessary input for this function is created in Section 4.1.1. In its current state, it is not fully automated. At the moment, it will only return a list containing the matrices and related information from the lower and upper nodes. From there, the user will need to select which node to branch on and run the function again.

Note: Skip to integer programming section for the related coding.

```
# Step 1: Select a viable entering variable (column)
# Step 2: Select a viable leaving variable (row)
# Step 3: Pivot at [row, column]
# Step 4: Check integer. Apply restriction
# Step 5: Determine if optimal. Return to step 1 if not optimal.
```

```
simplex_mixedInt <- function(lp) {
  if(typeof(lp)=="list"){
    A <- lp[[1]]
    varnames <- lp[[2]][[1]]
    isInteger <- lp[[2]][[2]]
    basic <- lp[[2]][[3]]
    identity <- lp[[3]]
  } else {
    return(print("Invalid Input"))
  }
  sPrimal <- function(lp) {
    if(typeof(lp)=="list"){
      A <- lp[[1]]
      varnames <- lp[[2]][[1]]
      identity <- lp[[3]]
    } else {
      A <- lp
      varnames <- c(’z’,paste0("x",1:(ncol(A)-2)),’RHS’)
    }
    library(kableExtra)
    # setting column and row names to numeric values
    colnames(A) = c(1:ncol(A))
    rownames(A) = c(1:nrow(A))
    count <- 0
    repeat{
      count <- count + 1
      # Locating column of first negative coefficient in objective
         function
      for(i in 1:ncol(A)){
        x <- A[2:nrow(A),i]
        if(A[1,i] < 0 & any(x > 0)){
          which(colnames(A)==i) -> c
          break
        }
      }
      rvec <- c(NA) # NA so objective function is ignored
      for(i in 2:nrow(A)){
        if(A[i,c] > 0){
          rvec <- append(rvec,(1/((A[i,c] / A[i,ncol(A)])))) # add row
             to row vector
        } else if(A[i,c] <= 0){
          rvec <- append(rvec,NA) # non-negativity constraint
        }
      }
      if(isTRUE(all.equal(rvec,NA))){ # Entering variable but no
         leaving variable
        count = 'Unbounded. Ignore Solution'
        break
      }
      which.min(rvec) -> r # select minimum of row vector
      A[r,] <- A[r,]/A[r,c] # reduce pivot row
      # row operations to pivot at [r,c]
      for(i in 1:nrow(A)){
        if(rownames(A)[i]==rownames(A)[r]){ # skip pivot row
          next
        }
        A[i,] <- A[i,] - A[i,c]*A[r,]
      }
      if(all(t(A[1,]) >= 0)){
        break
      }
      if(count==100){ #
        count <- 'Too Many'
        break
      }
    }
    return(A)
  }
  pivot <- function(A,r,c){
    colnames(A) = c(1:ncol(A))
    rownames(A) = c(1:nrow(A))
    A[r,] <- A[r,]/A[r,c] # reduce pivot row
    # row operations to pivot at [r,c]
    for(i in 1:nrow(A)){
      if(rownames(A)[i]==rownames(A)[r]){ # skip pivot row
        next
      }
      A[i,] <- A[i,] - A[i,c]*A[r,]
    }
    return(A)
  }
  checkIBFS <- function(A){
    RHS <- A[2:nrow(A),ncol(A)]
    basic <- rep(FALSE,ncol(A))
    for(i in 1:ncol(A)){
      if(identical(unique(A[,i]),c(0,1)) |
          identical(unique(A[,i]),c(1,0))){
        basic[i] = TRUE
      }
    }
    if(ncol((A[,which(basic==TRUE)])==ncol(diag(nrow(A))) & all(RHS
        >= 0)){
      return(TRUE)
    } else {
      return(FALSE)
    }
  }
  library(kableExtra)
  # setting column and row names to numeric values
  colnames(A) = c(1:ncol(A))
  rownames(A) = c(1:nrow(A))
  count <- 0
  # LINEAR PROGRAMMING
  repeat{
    count <- count + 1
    # Step 1 --------------------------------------------------------
    # Locating column of first negative coefficient in objective
        function
    for(i in 1:ncol(A)){
      x <- A[2:nrow(A),i]
      if(A[1,i] < 0 & any(x > 0)){ # checking obj coefficient is
          negative and at least 1 row below it is positive
        which(colnames(A)==i) -> c
        break
      }
    }
    # Step 2 --------------------------------------------------------
    # Dividing entering variable by RHS and selecting row with lowest
        value
    rvec <- c(NA) # NA so objective function is ignored
    for(i in 2:nrow(A)){
      if(A[i,c] > 0){
        rvec <- append(rvec,(1/((A[i,c] / A[i,ncol(A)])))) # add row
           to row vector
      } else if(A[i,c] <= 0){
        rvec <- append(rvec,NA) # non-negativity constraint
      }
    }
    if(isTRUE(all.equal(rvec,NA))){ # Entering variable but no leaving
        variable
      count = 'Unbounded. Ignore Solution'
      break
    }
    which.min(rvec) -> r # select minimum of row vector
    # Step 3 --------------------------------------------------------
    A[r,] <- A[r,]/A[r,c] # reduce pivot row
    # row operations to pivot at [r,c]
    for(i in 1:nrow(A)){
      if(rownames(A)[i]==rownames(A)[r]){ # skip pivot row
        next
      }
      A[i,] <- A[i,] - A[i,c]*A[r,]
    }
    # Step 4 -------------------------------------------------------
    # Check basis
    for(i in 1:ncol(A)){
      if(identical(unique(A[,i]),c(0,1))){
        basic[i] <- TRUE
      } else {
        basic[i] <- FALSE
      }
    }
    # Step 5 -------------------------------------------------------
    # breaks if all coefficients in objective row are 0 or positive
    if(all(t(A[1,]) >= 0)){
      break
    }
    if(count==100){ #
      count <- 'Too Many'
      break
    }
  }
  # INTEGER PROGRAMMING
  A[1,ncol(A)] -> best
  for(c in 1:(ncol(A)-1)){
    if(isInteger[c]==TRUE & basic[c]==TRUE){
      r <- which(A[,c]==1,arr.ind = TRUE) # Selecting row of basic
         integer
      if(near(A[r,ncol(A)],as.integer(A[r,ncol(A)]))){
        next
      }
      # Lower Bound
      f <- c(rep(0,ncol(A))) # Creating new vector for new constraint
      f[c] <- 1
      f[ncol(A)] <- floor(A[r,ncol(A)]) # Floor value of basic integer
      Af <- rbind(lp[[1]],f) # Adding vector to original matrix
      Af[,ncol(Af)] -> RHS # Chop RHS to add slack var
      Af <- Af[,-ncol(Af)]
      s1 <- c(rep(0,nrow(Af)-1),1)
      Af <- cbind(Af,s1,RHS) # Floored matrix
      sPrimal(Af) -> Af # Simplex on floored matrix
      # Grab objective value for comparison
      if(checkIBFS(Af)==TRUE){
        Af[1,ncol(Af)] -> lb
      } else {
        lb <- 0 # Infeasible
      }
```

```
variables <- list(varnames = varnames,
                integers = isInteger,
                basic = basic)
lpAf <- list(matrix = Af, variables = variables, identity =
   identity)
# Upper Bound
f[ncol(A)] <- ceiling((A[r,ncol(A)])) # Changing constraint to
   ceiling value of basic integer
Ac <- rbind(lp[[1]],f) # Row binding f to original matrix
Ac[,ncol(Ac)] -> RHS # Chop RHS
e1 <- c(rep(0,nrow(Ac)-1),-1) # Adding excess var
Ac <- cbind(Ac,e1,RHS)
objRow <- Ac[1,] # Locking objective row
a1 <- c(.Machine$integer.max, rep(0,nrow(Ac)-2),1) # Adding
   artificial var
#Ac[1,] <- c(1,rep(0,ncol(Ac)-1))
Ac <- Ac[,-ncol(Ac)]
Ac <- cbind(Ac,a1,RHS)
pivot(Ac,nrow(Ac),ncol(Ac)-1) -> Ac
sPrimal(Ac) -> Ac
#Ac[,-(ncol(Ac)-1)] -> Ac # Removing artificial variable
#Ac[1,] <- objRow # Reintroducing objective
if(checkIBFS(Ac)==TRUE){ # Check to primal
  sPrimal(Ac) -> Ac
}
if(checkIBFS(Ac)==TRUE){ # Check to feasible
  Ac[1,ncol(Ac)] -> ub
} else {
  ub <- 0 # Infeasible
}
compVec <- c(best,lb,ub)
variables <- list(varnames = varnames,
                integers = isInteger,
                basic = basic)
lpAc <- list(matrix = Ac, variables = variables, identity =
   identity)
return(list(lpAf=lpAf, lpAc=lpAc))
#break # needs fixing DO NOT PROCEED
if(which.max(compVec)==2){ # if lower bound is better, select as
   next node
  A <- Af
  best <- lb
  if(near(A[r,ncol(A)],as.integer(A[r,ncol(A)]))){
    return(A)
  }
} else if(which.max(compVec)==3){ # if upper bound is better,
   select as next node
```

```
        A <- Ac
        best <- ub
        if(near(A[r,ncol(A)],as.integer(A[r,ncol(A)]))){
          return(A)
        }
      }
    }
  }
}
```

4.1.8 Affine Scaling

Below is a function that will perform the Affine Scaling Algorithm until the difference between the current solution and the previous solution is less than machine double epsilon or until one of the solution elements reaches zero. The default alpha is set to 0.9, but it can be changed.

```
affine <- function(lp,x, alpha=0.9){
  if(typeof(lp)=="list"){
    lp <- lp[[1]]
  } else {
    lp <- lp
  }
  if(missing(x)){
    stop("Initial point x must be specified")
  }
  if(identical(unique(lp[,1]),c(1,0))){
    lp <- lp[,-1] # remove z column if present
  }
  xprev <- x
  repeat{
    C <- -1*lp[1,c(-ncol(lp))]
    M <- lp[-c(1),-c(ncol(lp))]
    D <- diag(x)
    MD <- M%*%D
    CD <- D%*%C
    # (transpose(MD)*inverse(MD*transpose(MD)))*MD
    np <- (t(MD)%*%(ginv(MD%*%t(MD))))%*%MD
    nullProjection <- diag(1,nrow(np),ncol(np)) - np # I - np
    legalDirection <- nullProjection%*%CD
    lambda <- min(legalDirection)
    step <- (-alpha/lambda)*legalDirection
    x <- (x + D%*%step)
    x <- as.vector(x)
    if(any(near(x,0))){
      return(xprev)
    } else if(all(near(x,xprev,tol=.Machine$double.eps))){
      return(x)
    } else {
      xprev <- as.vector(x)
    }
  }
}
```

4.1.9 Sensitivity Analysis

This is a simple function that will perform sensitivity analysis on an optimized matrix.

```
# Sensitivity Analysis
‘‘‘{r}
# For when the RHS changes
sensRHS <- function(M, x, delta){
  d <- delta*M[,x] # total change
  M[,ncol(M)] <- M[,ncol(M)] + t(d) # adding to RHS
  M <- cbind(M,d)
  return(M)
}

# For when the objective function values change
sensCost <- function(M, x, delta){
  r <- match(1,t(M[,x])) # Selecting pivot row
  c <- match(x,colnames(M)) # Selecting pivot column
  M[1,x] <- M[1,x] + delta
  M[r,c] -> pivot # pivot point at r,c
  M[r,] <- (1/pivot)*M[r,] # reduce pivot row
  # row operations to pivot at [r,c]
  for(i in 1:nrow(M)){
    if(rownames(M)[i]==rownames(M)[r]){ # skip pivot row
      next
    }
    M[i,] <- M[i,] - M[i,c]*M[r,]
  }
  return(M)
}
‘‘‘
```

4.1.10 Outputting R to LaTeX

To make it easier to import the tableaus and matrices into LaTeX from R, I use the xtable package to write a .tex file of the matrix which can be opened in RStudio and copy and pasted quickly. It needs a little work afterwards to make it exactly how it should look, but this greatly speeds up the process.

```
# @param A = matrix to write to LaTeX
# @param lengthvec = vector containing the number of digits for each
   column. Add a 0 entry to first position.
library(xtable)
latex <- function(A,lengthvec=2){
  print(xtable(A, type = "latex", digits = lengthvec), file =
     "LatexOutput.tex")
}
```

5 Client Reports

5.1 Report: Corn, Wheat, Rye

5.1.1 Introduction

Based on your available land, labor, and fertilizer on hand, I have created the following system of linear equations to model your expected profit. In this model, c = corn, w = wheat, and r = rye.

$$\begin{array}{rrrrl} \text{Maximize:} & 400c+300w+250r & & & \\ \text{Subject to:} & c+ & w+ & r\leq & \text{45 acres} \\ & 4c+ & 2w+ & r\leq & \text{165 lbs. of fertilizer} \\ & 40c+ & 60w+ & 20r\leq & \text{2320 labor hours} \\ & c,w,r\geq 0 & & & \end{array}$$

This linear program was optimized using a primal simplex algorithm (Detailed in Section 2.1.3).

5.1.2 Recommendations

The best allocation of your resources will be achieved by planting 40 acres of corn and 5 acres of rye. Doing so will net you a revenue of $17250. This will use up all of your available land and fertilizer and you will have 620 labor hours unused.

5.1.3 Additional Considerations

After optimizing your resource allocation, I performed sensitivity analysis to determine if it is worth it for you to purchase any more of your resources. Should you be able to purchase fertilizer for less than $50 per pound, I would recommend doing so. You will be able to purchase up to 15lbs of fertilizer without needing any more land or labor hours. If you purchase this fertilizer, your best allocation of resources is to plant corn in all 45 acres which will net you $18000 in revenue, an increase of $750.

To cut costs further, I would also recommend reducing your available labor hours if your workers are hourly. Cutting up to 620 labor hours may save significantly on costs.

Back to Grade Discussion

5.2 Report: Bakery

5.2.1 Introduction

The first step to setting up the model for your baking options was to convert all of the units of your ingredients to universal units. This was necessary so that all of the operations remained consistent throughout the model. I setup the following matrix to be optimized:

z	X_1	X_2	X_3	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	RHS
1	-32	-27	-22	0	0	0	0	0	0	0	0	0	0	0
0	1	3/4	3/16	1	0	0	0	0	0	0	0	0	0	100
0	15/16	8/16	6/16	0	1	0	0	0	0	0	0	0	0	80
0	14/16	20/16	24/16	0	0	1	0	0	0	0	0	0	0	150
0	1/8	1/12	1/6	0	0	0	1	0	0	0	0	0	0	20
0	2/3	1/2	13/24	0	0	0	0	1	0	0	0	0	0	70
0	1/8	0	0	0	0	0	0	0	1	0	0	0	0	10
0	1/3	1/6	0	0	0	0	0	0	0	1	0	0	0	20
0	4	1	0	0	0	0	0	0	0	0	1	0	0	180
0	6/16	4/16	0	0	0	0	0	0	0	0	0	1	0	20
0	0	0	1/4	0	0	0	0	0	0	0	0	0	1	16

In this matrix, X1=Cupcakes, X2=Cookies, X3=Bagels, S1=Sugar, S2=Pumpkin, S3=Flour, S4=Salt, S5=Pumpkin Pie Spice, S6=Corn Starch, S7=Baking Powder, S8=Eggs, S9=Butter, S10=Yeast. The first row is the amount of money received for each batch made, and all rows below it are the ingredient constraints.

5.2.2 Recommendations

To make the most money from your ingredients on hand, I would recommend baking 60 bagels, 40 cupcakes, and 20 cookies. This will net you a revenue of $3140.00 assuming that you sell everything that is baked.

5.2.3 Additional Considerations

Given the prices of ingredients on the list you provided, I performed sensitivity analysis to determine if there were any ingredients worth buying additional units of to further increase profits.

You will have used the entirety of your flour, eggs, and butter so I looked to those ingredients.

The first ingredient I analyzed was flour. You are able to purchase an additional 2lbs of flour for $1.74 without purchasing anything else and this will increase your profits by $27.59. You will be able to make $1\frac{1}{3}$ additional batches of bagels with this butter.

The next ingredient I looked at was butter. You are able to purchase 6lbs of butter for $2.88 and will increase your profits by $143.04. Under this circumstance, you will make $33\frac{2}{3}$ batches of bagels, $33\frac{1}{3}$ batches of cupcakes, and $58\frac{1}{3}$ batches of cookies for a total profit of $3288.80.

5.3 Report: Shipping Costs

5.3.1 Introduction

Given the two orders from your customers, the amount of sheets that you have in stock at your east and west warehouses, and the cost of shipping, I have set up the following model for the cost of shipping to your customers. In this model, E_A is the amount of sheets shipped from the East warehouse to Customer A, E_B is the amount shipped from the East warehouse to Customer B, and so on...

$$\begin{array}{rl} \text{Minimize:} & 0.5E_A + 0.6E_B + 0.4W_A + 0.55W_B \\ \text{Subject to:} & \begin{array}{ccccccccc} E_A & + & E_B & & & & & \leq & 80 \\ & & & & W_A & + & W_B & \leq & 45 \\ E_A & & & + & W_A & & & \geq & 50 \\ & & E_B & & & + & W_B & \geq & 70 \end{array} \\ & E_A, E_B, W_A, W_B \geq 0 \end{array}$$

This linear program was optimized using the dual simplex algorithm. For a highly detailed walkthrough, see Section 3.2.

5.3.2 Recommendations

To minimize your shipping expenses, you will want to arrange your shipments as follows:
Ship 5 sheets from the East warehouse to Customer A.
Ship 70 sheets from the East warehouse to Customer B.
Ship 45 sheets from the West warehouse to Customer A.
This will cost a total of $62.50 and you will have 5 sheets remaining in your East warehouse.

5.4 Report: Iron Processing

5.4.1 Introduction

Based on the information given about the production ratios for each method, I have devised the following model to determine how much it will cost to produce the amount of each steel grade ordered by your clients. In this model, M_k denotes the amount of iron processed in Methods 1,2,3 respectively.

$$\begin{array}{rl}
\text{Minimize:} & 20M_1 + 20M_2 + 30M_3 \\
\text{Subject to:} & \frac{1}{8}M_1 + \frac{1}{6}M_2 + \frac{2}{5}M_3 \geq 4 \\
 & \frac{1}{4}M_1 + \frac{1}{6}M_2 + \frac{1}{10}M_3 \geq 4 \\
 & \frac{1}{8}M_1 + \frac{1}{3}M_2 + \frac{1}{5}M_3 \geq 8 \\
 & \frac{3}{8}M_1 + \frac{1}{6}M_2 + \frac{1}{5}M_3 \geq 5 \\
 & \frac{1}{8}M_1 + \frac{1}{6}M_2 + \frac{1}{10}M_3 \geq 0 \\
 & 1\frac{3}{4}M_1 + 1\frac{1}{2}M_2 + 1\frac{1}{5}M_3 \leq 39 \\
 & M_1, M_2, M_3 \geq 0
\end{array}$$

The dual simplex algorithm was used to optimize this linear program. For a highly detailed walkthrough, see Section 3.3

5.4.2 Recommendations

The results show that it would be best to process 24 tons of iron using Method 2 only. This will cost a total of $480 and there will be an additional 4 tons of grade 4 steel leftover to sell. You will also be 3 hours beneath the maximum processing time of 39 hours.