

# NETWORK OPTIMIZATION USING LINEAR PROGRAMMING AND REGRESSION

by

JASON LEE

A THESIS

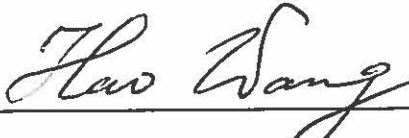
Presented to the Department of Mathematics  
and the Robert D. Clark Honors College  
in partial fulfillment of the requirements for the degree of  
Bachelor of Science

June 2016

## An Abstract of the Thesis of

Jason Lee for the degree of Bachelor of Arts  
in the department of Mathematics to be taken June 2016

Title: Network Optimization Using Linear Programming and Regression

Approved:  \_\_\_\_\_  
Professor Hao Wang

The purpose of this research is to explore the synergistic application of linear programming, regression, and computer science to solve practical economic problems. In particular, this research focuses on network optimization problems in which the aim is to maximize revenue and minimize production, transportation, and other costs by using historical information to predict future market behavior. The first half of the thesis provides background information on linear programming and regression for readers who may not be familiar with the subjects. The remaining sections of this thesis cover the modeling process and computer programming design, where the variables of interest are identified, arranged into an appropriate form, and MATLAB programming is utilized to carry out linear programming and regression operations to provide an optimized solution for the network using given historical data and market conditions.

## **Acknowledgments**

I would like to thank Professor Wang for giving me inspiration to start this project, Professor Sinclair and Professor Prazniak for serving on my defense committee, and all my advisors and peers for giving me feedback along the way.

Without all their support none of this would have been possible.

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Linear Programming</b>	<b>1</b>
2.1	Brief History of Linear Programming . . . . .	2
2.2	The General Linear Programming Problem . . . . .	3
2.3	Properties of Linear Programming Problems . . . . .	5
2.4	The Simplex Method . . . . .	5
2.5	Degeneracy . . . . .	9
<b>3</b>	<b>Regression</b>	<b>10</b>
3.1	Example . . . . .	12
<b>4</b>	<b>MATLAB Implementation</b>	<b>16</b>
4.1	Basic Model . . . . .	16
4.2	Annotated Code . . . . .	18
4.3	Regression Application . . . . .	20
<b>5</b>	<b>Appendix</b>	<b>23</b>

## List of Figures

1	Graph showing the feasible region for the example problem . . . . .	6
2	Graph showing data with a fitted line and residuals . . . . .	11
3	Plots of the residuals for two different fitted lines . . . . .	12
4	Scatter Plot of Sample Data . . . . .	13
5	Sample Data with Fitted Linear Model . . . . .	13
6	Plot of the Residuals for the Linear Model . . . . .	14
7	Sample data with Fitted Polynomial Model . . . . .	15
8	An example network with $n = 3, m = 2$ . . . . .	16

# 1 Introduction

The goal of this work is to create a program which will utilize mathematical methods to determine an efficient production and distribution plan for a network of production, storage, and distribution centers given available data. This is valuable because it eliminates the need for manual calculation, something that is necessary when networks include hundreds of different nodes. The first half of this paper provides a basic introduction of linear programming and regression to provide background for those who may not be familiar with the topics, and covers a complete example of each technique. The second section lays out the problem of interest and includes a walk-through for the MATLAB code used to provide the solutions.

## 2 Linear Programming

Linear programming, also known as linear optimization, is a field of mathematics that deals with finding efficient solutions to systems defined by multiple linear equalities and inequalities. An efficient solution is one where a specific value is minimized or maximized, such as minimum cost or maximum profit. Linear programming is commonly used to solve management problems since the solutions it provides focus on maximizing efficiency. The focus of my work is transportation networks and production schemes, though the same techniques can be applied to any situation which requires allocating a finite amount of resources in an optimal way including investing, route planning, and work shift assignment.

## 2.1 Brief History of Linear Programming

Linear programming does not have as much history to it relative to other fields of mathematics largely due to the difficulty in solving most problems without the aid of computer calculation. One of the first attempts to provide a viable solution to linear programming problems was made by Joseph Fourier, who published a method in 1827. However, his algorithm was inefficient for larger systems since it required at least exponential time to carry out, meaning that problems with a hundred or more variables would still require an impossible amount of time to solve. A few other attempts were made to provide a solution, but they had limited success. Major developments in the field would not be made until the mid-20th century.

The most well known and significant contribution to the field of linear programming was made by George Dantzig in 1947, when he developed the simplex method for solving linear programming problems while working as an military advisor at the Pentagon. The simplex method was the first practical algorithm which had a time efficient solution to linear programming problems. However, at the time of its creation there were still limits on the ability to solve large problems. In fact, one of the first implementations of the algorithm on a large system required 8 hours of feeding cards into a Card Programmable Calculator (CPC) [3]. Despite these limitations, the simplex algorithm allowed people to solve linear programming problems in a much more efficient manner than any previous method. Interestingly enough, the name linear programming has little to do with actual computer programs, and actually has its origins in the military use of the word program to refer to their logistical and deployment plans [5].

Though the simplex method was initially intended to improve various aspects of military logistics, it was quickly adapted and used in commercial applications such as

oil refining and blending [5]. The decade that followed also saw improvements to the simplex method and the development of related fields like nonlinear programming and integer programming.

In the present, linear programming is more useful than ever before for management and logistical challenges due to easy access of large amounts of data. Combined with improved algorithms and improvements in computing power, linear programming is able to solve larger and more complicated problems. The primary challenge now is accurately modeling applicable situations and using available data efficiently.

## 2.2 The General Linear Programming Problem

In linear programming problems, the primary goal is to maximize or minimize a linear function, which we will call  $z$ , that is subject a finite set of linear constraints. The function  $z$  is known as the **objective function** and is a linear combination of the variables  $(x_1, x_2, \dots, x_n)$  with the general form  $z = c_1x_1 + c_2x_2 + \dots + c_nx_n$  where each  $c$  is a constant. The **linear constraints** can be either equality constraints or inequality constraints, and take the general form

$$a_1x_1 + a_2x_2 + \dots + a_nx_n \left\{ \begin{array}{l} \leq \\ = \\ \geq \end{array} \right\} b$$

where  $a$  and  $b$  are constants. A **feasible solution** is any combination of the variables  $(x_1, x_2, \dots, x_n)$  that satisfies the constraints, and the set of these  $n$ -tuples is known as the **feasible region**. A problem which has no solution which satisfies all of the constraints is **infeasible**. The full form of the general linear programming problem



is as follows

$$\begin{array}{ll} \text{Maximize} & c_1x_1 + c_2x_2 + \dots + c_nx_n = z \\ \text{Subject to} & a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq b_1 \\ & a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \leq b_2 \\ & \vdots \\ & a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq b_m \\ & x_i \geq 0 \quad i \in 0, 1, \dots, n \\ & b_i \geq 0 \quad i \in 0, 1, \dots, m \end{array}$$

Notice that all the linear constraints are written using  $\leq$  rather than a mixture of equality and inequality constraints, which is possible since we can easily convert constraints from one form to another. An equality constraint can be converted to an inequality constraint by creating two separate inequalities which converge on the same point. This allows us to have the restriction that all  $x_i$  and  $b_i$  must be positive and makes solving the problem easier.

Additionally, there can be situations where we would want to minimize the objective function rather than maximize it, such as a setup where the goal is to minimize costs. Though these objectives may seem like polar opposites, the distinction between minimizing and maximizing the objective function is actually trivial since minimizing  $z$  is the same thing as maximizing  $-z$ . Therefore, we can change the objective function to a  $-z$  when we want to minimize it, so the difference can be eliminated by a simple sign change. The general linear programming problem is also commonly written in matrix form

$$z = c^T x, \quad Ax \leq b, \quad x_i \geq 0, \quad b_i \geq 0$$

$$c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ & & \ddots & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

## 2.3 Properties of Linear Programming Problems

Linear programming problems have several important properties which allow us to know a solution exists as long as the feasible region is bounded. The feasible region is said to be **bounded** if there exists a number  $M$  such that  $|x_i| \leq M$  holds for every feasible point  $(x_1, x_2, \dots, x_n)$  and each  $i = 1, 2, \dots, n$ . [8]

1. If the set of linear constraints defines bounded feasible region, then there exists a point in that region that maximizes the objective function and a point that minimizes the objective function.
2. If a maximum occurs in the feasible region, it must be a vertex point of the feasible region.
3. If a minimum occurs in the feasible region, it must be a vertex point of the feasible region.

## 2.4 The Simplex Method

The simplex method was the first algorithm which successfully solved large linear programming problems. It functions by first finding a basic feasible point, if one is available, and checks to see if it gives a maximum value for the objective function. If not, it searches for a new point in the feasible region that yields a higher value for

the objective function and repeats this process until a maximum is obtained. The following example shows the basic steps of the simplex method.

### 2.4.1 Example Problem

$$\begin{aligned} \text{Maximize} \quad & x_1 + x_2 = z \\ \text{Subject to} \quad & 2x_1 + x_2 \leq 4 \\ & x_1 + 2x_2 \leq 3 \\ & x_1, x_2 \geq 0 \end{aligned}$$

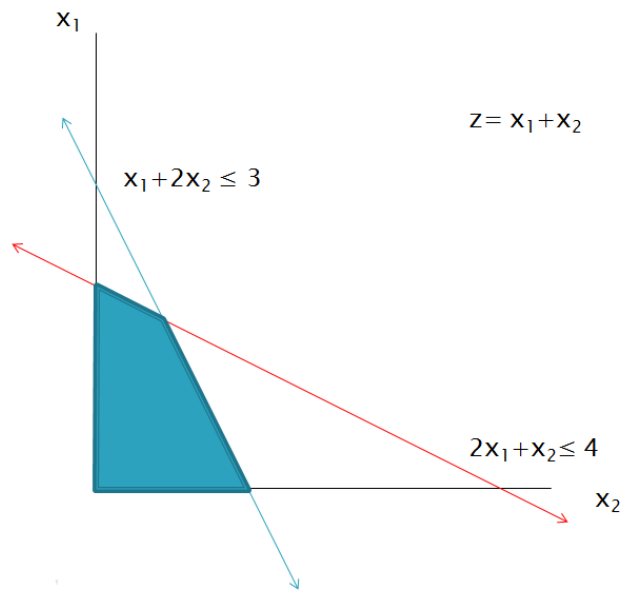


Figure 1: Graph showing the feasible region for the example problem

The first step in the simplex method is to introduce new variables for each constraint, known as **slack variables**, which are used to turn the inequalities into equalities. In this case, we define the slack variables,  $y_i$ , as the positive difference between the two sides of the inequalities. We also rewrite the objective function with all the variables on one side to allow for easier calculation. So we now have the following

equations, with the rewritten objective function placed on the bottom for the sake of convention.

$$\begin{aligned} 2x_1 + x_2 + y_1 &= 4 \\ x_1 + 2x_2 + y_2 &= 3 \\ -x_1 - x_2 + z &= 0 \\ x_1, x_2, y_1, y_2 &\geq 0 \end{aligned}$$

We can write these equations in an augmented matrix form which allows us to perform row operations to obtain a solution in the form of  $(x_1, x_2, y_1, y_2)$ . This form is known as the **simplex tableau**.

$$\begin{array}{ccccc|c} x_1 & x_2 & y_1 & y_2 & z & \\ \hline 2 & 1 & 1 & 0 & 0 & 4 \\ 1 & 2 & 0 & 1 & 0 & 3 \\ -1 & -1 & 0 & 0 & 1 & 0 \end{array}$$

Now that the equations are in matrix form, we can look for our initial solution. The first step is to look for all the variables whose columns consist of 0's and a single 1. These variables are known as **basic variables**. The initial solution consists of setting the value of all the non-basic variables to zero and solving for the basic variables. For this problem, this gives us the solution  $(0, 0, 4, 3)$  and a value of 0 for  $z$ . From here, we can check to see if this is the optimized solution by looking at the bottom row of the matrix. The negative entries indicate that we do not have an optimized solution since increasing either of their values would lead to an increase in  $z$ . Therefore, we must continue with the algorithm.

The next step in the process is to find the variable in the last row which is the most negative and choose one of the entries in that column as a pivot. If there are equivalent numbers, then we pick the leftmost column. To determine which entry to

use as a pivot, we pick the row whose value has the smallest value when dividing the corresponding value in the right-most column. This selection process for the pivot is known as **Bland's Rule**. In this case, we choose the  $x_2$  entry in the 2nd row since  $\frac{3}{2} < \frac{4}{1}$ .

$$\begin{array}{ccccc|c} x_1 & x_2 & y_1 & y_2 & z & \\ \hline 1 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 2 \\ 0 & \frac{3}{2} & -\frac{1}{2} & 1 & 0 & 1 \\ 0 & -\frac{1}{2} & \frac{1}{2} & 0 & 1 & 2 \end{array}$$

Now we repeat the same process for the new basic and non-basic variables, and arrive at a solution of  $(2, 0, 0, 1)$  with a value of 2 for  $z$ . While this new solution has a larger value for the objective function, it still is not the most efficient solution since we can still increase its value by increasing  $x_2$ . Therefore, we repeat the same process of finding the most negative value in the last row and using it as a pivot column over and over till all the entries in the last row are positive, since that would indicate that we have an efficient solution. In this case, we only need to do that one more time before we get

$$\begin{array}{ccccc|c} x_1 & x_2 & y_1 & y_2 & z & \\ \hline 1 & 0 & \frac{1}{3} & -\frac{1}{3} & 0 & \frac{5}{3} \\ 0 & 1 & -\frac{1}{3} & \frac{2}{3} & 0 & \frac{2}{3} \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & 1 & \frac{7}{3} \end{array}$$

This gives us a solution of  $(\frac{5}{3}, \frac{2}{3}, 0, 0)$  and  $z = \frac{7}{3}$  which we know is the optimal solution since our objective function would be

$$z = \frac{7}{3} - \frac{1}{3}y_1 - \frac{1}{3}y_2$$

and since the values of  $y_1, y_2$  can only be positive, that indicates that the value of  $z$

cannot exceed  $\frac{7}{3}$ .

### **General Steps of the Simplex Algorithm**

1. Add slack variables to each inequality to change them into equalities.
2. Format the equations into an augmented matrix form, also known as the simplex tableau.
3. For the initial solution, set the value of all non-basic variables to 0 and solve for the value of the basic variables.
4. Check if there are any negative values for any of the non-basic variables in the last row. If there are any, then the solution is not the most efficient. Otherwise, the solution is optimal and you are done.
5. Find the most negative value in the last row, picking the leftmost value in the case of a tie.
6. Find out which element in that column has the smallest value when dividing the rightmost element of its row. That element becomes a new pivot. Make appropriate changes to the matrix to reflect this.
7. Return to step 4.

### **2.5 Degeneracy**

There is a special case, known as degeneracy, that can sometimes occur when carrying out the simplex algorithm. Degeneracy occurs when the value of one of the coefficients in the rightmost column is 0. This causes the value of the objective function to not change during an iteration of the simplex algorithm despite the variables being different. Degeneracy is not a major issue in linear programming problems

since its primary drawback is that it causes the simplex method to carry out extra iterations [11]. There is a rare case where degeneracy can cause cycling, in which the simplex method gets caught cycling through the same feasible solutions. However, cycling can be avoided by applying Bland's rule when carrying out the algorithm, which determines which value is chosen as the next pivot.

### 3 Regression

Regression analysis is a set of techniques used to make predictions about the relationship between variables. Like linear programming, regression makes extensive use of matrices and linear algebra. Regression has applications in almost every field, whether it is education or weather, physics or biology. Though there are many different regression techniques available, we will only make use of a few of them related to linear models and checking their validity. We can make use of regression analysis to find equations that fit data on demand, shipping prices, and other variables. Using those equations, we can make predictions for future values of those variables and use that information in our model. The regression techniques that we will make use of are linear regression, residual analysis, and forward/backward selection.

One of the most basic linear regression techniques is **ordinary least squares**. This method seeks to fit a line to a set of data by choosing the line which minimizes the sum of the squared errors (SSE). The errors, also known as **residuals**, are the difference between the predicted values and the actual values. The predicted values are the  $y$  values given by the fitted line for any particular  $x$ . The fitted line will have predicted values for the coefficients whose value is indicated by the associated  $p$ -value. A small  $p$ -value,  $\alpha < 0.05$ , means that the term has a statistically significant contribution to the model. If the  $p$ -value is large, we can remove that coefficient

from the model to improve it. In this work, we use linear regression to create and test how various models fit the data and use them to make predictions for future data points.

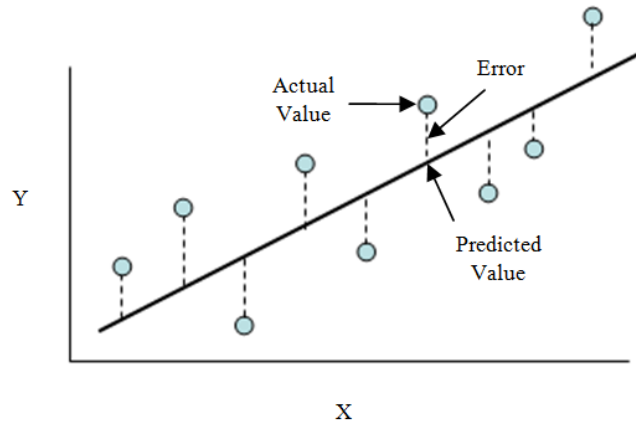


Figure 2: Graph showing data with a fitted line and residuals

Regression also includes many different ways to test if the models are reliable. Examining the plots of the residuals is one way to ensure the model fits the data well. If the model is appropriate for the data, then we would expect the residuals to be normally distributed as shown in the first plot in figure 3. However, some times the model chosen is not appropriate to the data and we get residual plots like the second one where the residuals form a parabolic shape. That suggests that the data would be better described by a polynomial model. In order to compare different models to determine which one fits the data the best, we will use **Akaike's Information Criterion** (AIC) which measures model quality based of the number of parameters and statistical goodness of fit. Similarly the **Bayesian Information Criterion** (BIC) also measures model quality based on the same criteria, but tends to favor models with fewer parameters when compared with the AIC. Ideally, the best model



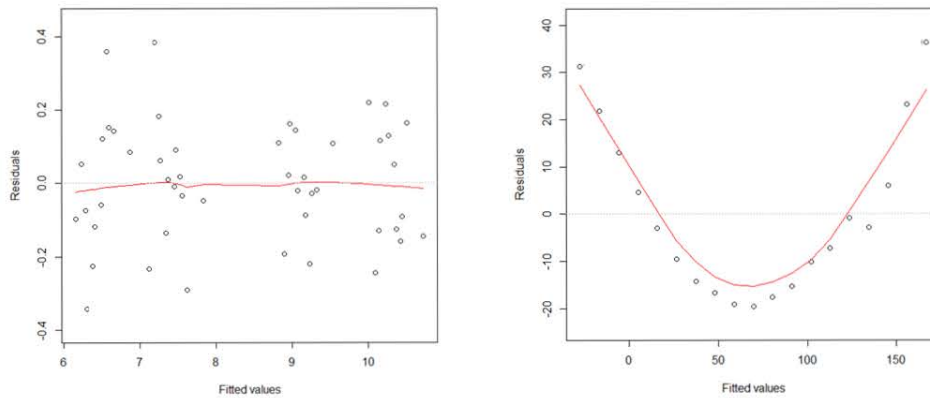


Figure 3: Plots of the residuals for two different fitted lines

will have both the smallest AIC and BIC value. The formulas for AIC and BIC are

$$\text{AIC} = -2 \log(L) + 2K \quad \text{BIC} = -2 \log(L) + K \log(n)$$

where  $L$  is the value of the likelihood function,  $K$  is the number of parameters, and  $n$  is the number of observations.

### 3.1 Example

In this example we will generate a dataset and demonstrate how to evaluate the degree to which a particular model fits the data. The procedure used to generate the data and all relevant MATLAB code are included in the Appendix. The first step when attempting to create a model for the data is to plot the data and try to determine a potential model based on the shape of the graph.

Looking at the graph, there is a clear positive correlation between  $X$  and  $Y$ , which appears to be roughly linear, so our initial model will be  $Y = \beta_0 + \beta_1 X$ . Using the *fitlm* function using the basic linear model yields the result

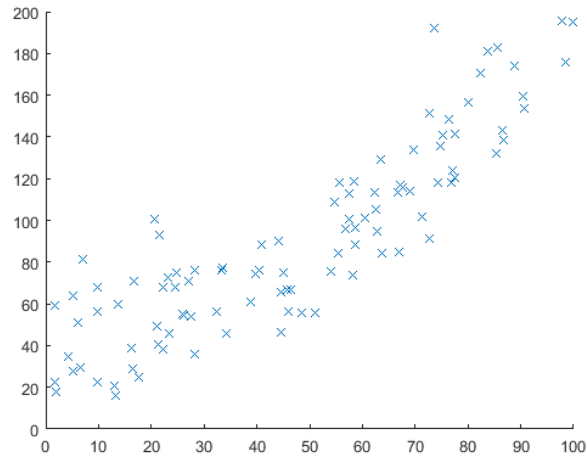


Figure 4: Scatter Plot of Sample Data

	Estimate	SE	tStat	pValue
(Intercept)	22.235	4.2922	5.1804	$1.1849e - 06$
X	1.4401	0.078442	18.359	$1.7449e - 33$

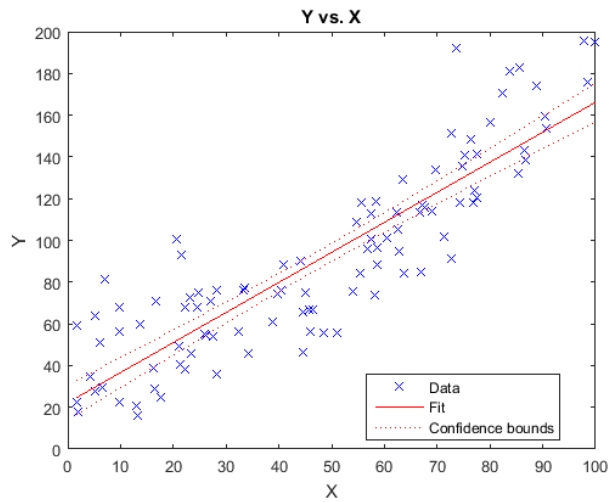


Figure 5: Sample Data with Fitted Linear Model

Both  $p$ -values for the coefficients are extremely small, which means that both terms have a statistically significant effect on the model. This model also has a root mean

squared error (RMSE) of 21.5 and an  $R^2$  value of 0.775. Though this may appear to be a good fit for the data, however, it is always a good idea to test other models to see if a better fit can be achieved. For this example, we can look at a plot of the residuals to see if there are any trends which would suggest that a better model exists. If the current model is a good fit, then the residuals will display a normal distribution.

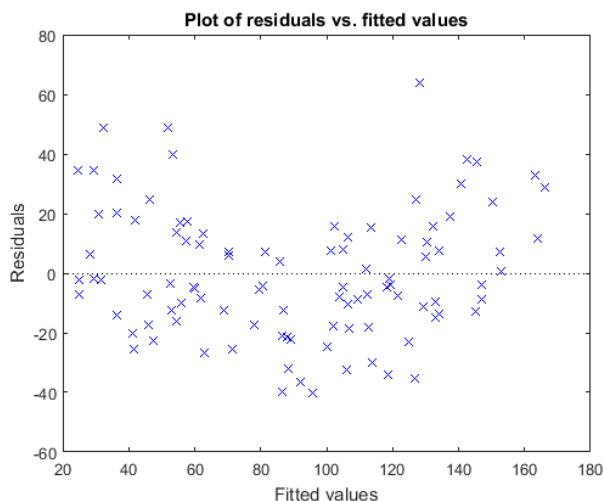


Figure 6: Plot of the Residuals for the Linear Model

Looking at the plot of the residuals versus the fitted values, there is a small but definite curved shape to them which indicates that a polynomial model might be a better fit. The first polynomial model we will try will be  $Y = \beta_0 + \beta_1 X + \beta_2 X^2$ .

	Estimate	SE	tStat	pValue
(Intercept)	42.878	5.7545	7.4512	$3.8595e - 11$
X	0.13419	0.27851	0.48182	0.63102
$X^2$	0.013769	0.0028403	4.8478	$4.7436e - 06$

Though this model has a smaller RMSE of 19.4 and a larger  $R^2$  of 0.819, the X

coefficient is not significant with a  $p$ -value of 0.63102, so we remove that term and try a reduced model with just the  $X^2$  and intercept terms.

	Estimate	SE	tStat	pValue
(Intercept)	45.273	2.8896	15.667	$1.9427e - 28$
X	0.015093	0.00071858	21.004	$4.6156e - 38$

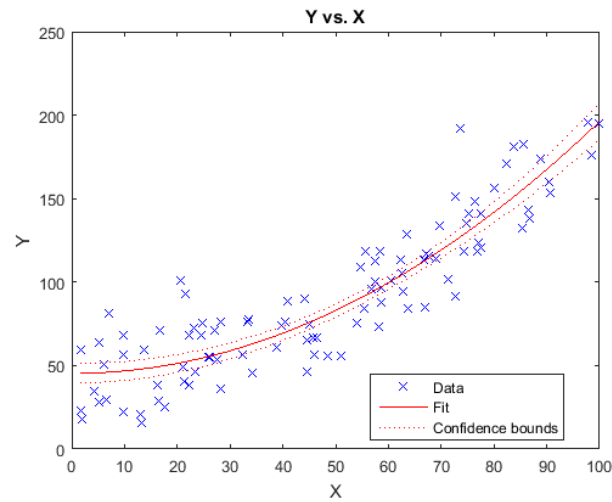


Figure 7: Sample data with Fitted Polynomial Model

In addition to having both coefficients have significant  $p$ -values, this model has the best RMSE of 19.3 and highest  $R^2$  value of 0.818, which indicates that it fits the data better than the other two. The graph of the residuals has a normal distribution of values, and if compare the calculated values to the original function used to generate the data shows that the model is rather accurate.

## 4 MATLAB Implementation

In this section I will walk through the steps of creating and implementing both the linear optimization and regression program. The basic situation I want to model is the production and distribution of one or more products. The image below depicts a very basic model where three distribution nodes are each connected to two destination nodes.

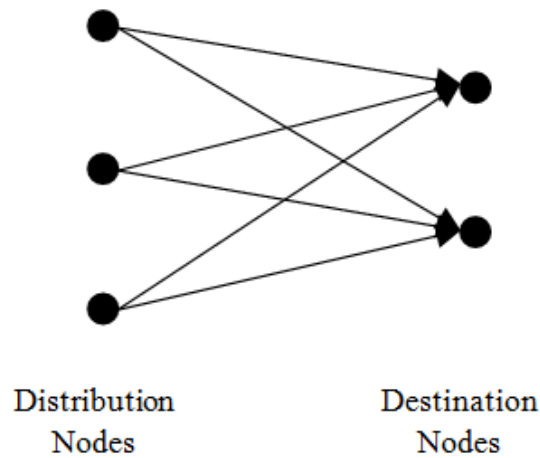


Figure 8: An example network with  $n = 3, m = 2$

### 4.1 Basic Model

The first step in the process is to identify the variables we want to consider and represent the problem in mathematical terms. In this basic model, for the sake of simplicity we will only consider a system of factories and stores, leaving the inclusion of distribution/storage centers to a later model. The variables of interest are:

- $F$  - number of factories

- $S$  - number of stores
- $P$  - number of products
- $D_{ps}$  - demand for a product at a store
- $PL_{fp}$  - production limit at a factory for a particular product
- $TC_p$  - the transportation cost of a product
- $PC_{fp}$  - production cost of a product at a particular factory
- $DI_{fs}$  - distance between a factory and store

The goal is to minimize the total costs, while having the demand for every product and store satisfied. Therefore the objective function for this problem would be the total production costs for all of the products and the total transportation costs. In the objective function  $x(f, p, s)$  represents the amount of a product that a particular factory sends to a particular store.

$$z = \sum_{i=1}^F \sum_{j=1}^P \sum_{k=1}^S x(f, p, s) * (PC_{fp} + TC_p * DI_{fs})$$

For the constraints, we already have the condition that all the demands need to be satisfied, and the only other conditions are that the production limits are not exceeded.

$$\sum_{k=1}^S x(f, p, s) \leq PL_{fp} \quad f = 1, 2, \dots, F \text{ and } p = 1, 2, \dots, P$$

$$\sum_{i=1}^F x(f, p, s) = D_{ps} \quad p = 1, 2, \dots, P \text{ and } s = 1, 2, \dots, S$$

Now that we have the objective function and all the constraints defined, we now need to set up the problem so that it can be solved in MATLAB. For this problem,

we will be using the *linprog* function from MATLAB's optimization toolbox and will use the dual-simplex algorithm.

## 4.2 Annotated Code

In order to solve a linear programming problem in MATLAB using *linprog*, we need to create and fill several matrices containing the coefficients for the objective function, inequalities, equalities, and any lower or upper bounds that exist.

```
%=====
%This program provides an efficient solution to the system of factories and
%stores with multiple products
%=====

% Set the rng to default to ensure code works, change later for tests
rng('default')

% Number of factories, stores, and products
Size = 8;
Size2 = Size*Size;
F = floor(.1 *Size2);
S = floor(.12 *Size2);
P = 3;

% Randomly determine constraints for the problem for testing data
d = round(40*rand(P,S) + 10); %Demand at stores (10-50)
pl = round(35*rand(F,P) + 40); %Production limit at factories (40-75)
tc = round(2*rand(1,P) + 1); %Transportation cost per unit distance (1-3)
pc = round(10*rand(F,P) + 5); %Production cost (5-15)

% Place the factories and stores onto a grid
xy = randperm(Size2,F+S);
[x,y] = ind2sub([Size Size],xy);

% Plot the data on a graph for a visual representation
h = figure;
plot(x(1:F),y(1:F),'^',x(F+1:F+S),y(F+1:F+S),'rs')
legend('Factory','Store','Location','WestOutside')
xlim([0 Size+1]);ylim([0 Size+1])

% Determine the distance between each pair of F and S, no diagonals
dist = zeros(F,S);
for a = 1:F
    for b = 1:S
```

```

        dist(a,b) = abs(x(a)-x(F+b))+abs(y(a)-y(F+b));
    end
end

% Create the objective function
Z = zeros(F,P,S);
for a = 1:F
    for b = 1:P
        for c = 1:S
            Z(a,b,c) = pc(a,b) + dist(a,c)*tc(b);
        end
    end
end

Z1 = Z(:);
w = length(Z1); %Size of constraints determined by length of Z

% Create the empty inequality matrices
A = sparse(F*P,w,F*P*S);
b = zeros(F*P,1);

% Create the empty equality matrices
Aeq = sparse(P*S,w,F*P*S);
beq = zeros(P*S,1);

% Create empty rows to fill in with data
empty = zeros(size(Z));
empty1 = zeros(1,F*S);

% Fill in the matrix A and matrix b
count = 1;
for m = 1:F
    for n = 1:P
        entry = empty;
        entry(m,n,:) = 1;
        entry = sparse(entry(:));
        A(count,:) = entry';
        b(count) = pl(m,n);
        count = count+1;
    end
end

% Fill matrices Aeq and beq, assume demand is always met
count = 1;
for q = 1:P
    for r = 1:S
        entry = empty;
        entry(:,q,r) = 1;
        entry = sparse(entry(:));
        Aeq(count,:) = entry';
        beq(count) = d(q,r);
    end
end

```



```

        count = count+1;
    end
end

lb = zeros(w,1); %sets the lower bound as 0
ub = []; %no upper bound for values

opts = optimoptions('linprog','Display','final','Algorithm','dual-simplex');

[x,fval,exitflag,output] = linprog(Z,A,b,Aeq,beq,lb,ub,[],opts);

%Shows allocation reccomendations, rows are different factories,
%columns are different products
FIN = reshape(x,[F,P,S])

fval,exitflag,output;

```

### 4.3 Regression Application

For most realistic production networks, the amount of demand for a particular product is never a constant value. Therefore, in this system I plan to use regression to predict future values for the demand. For the purposes of testing, and since I do not have access of specific demand data for specific products, I will model demand with the assumption that it follows one of the following forms with a single variable: constant  $Y = \beta_0$ , linear  $Y = \beta_0 + \beta_1 X$ , or polynomial  $Y = \beta_0 + \beta_1 X + \beta_2 X^2$ . It is definitely possible and not very difficult to include more variables. However, the main goal of this section is to lay out a basic system for comparing models and determining the best fit.

Given the data set of quarterly demand (generated through a process which can be found in the Appendix), we first need a method to read the data and convert in into variables that can be used in the program. The *dlmread* command allows for easy converting of data in text files to a usable matrix regardless of whether the data is seperated by commas, spaces, or some other delimiter.

The next step is to test the two models and compare the AIC and BIC values to determine which one is superior. Though we could use the F test to compare the two different models in this example because the polynomial model contains all the terms of the linear model, AIC and BIC will be used because they are also applicable in the cases where the models are not nested which gives the possibility of including more models in the comparison if desired. Ideally, this program would select the best model automatically and then give an estimate for the value of the demand for the coming quarter. Therefore, we need to create a function which selects the lowest AIC and BIC values, chooses that model, and then uses that model to estimate demand. The *fitlm* command already calculates the values of the AIC and BIC for the models, so we can simply call on that value for comparison. In the case that the AIC and BIC do not agree on the best model we will default to the choice indicated by the AIC, however in the majority of cases that should not be an issue.

Finally, we will have the program take the model which has the best prediction, and output the prediction to a text file automatically which can be read by the linear optimization program.

```

%=====
%This program reads data from a text file and tests several different
%polynomial and linear models for fit, chooses the best one, and prints the
%results to a text file.
%=====

A = dlmread('file.txt','\t',1)

%modify the quarter data so q1 2013 is 2013, q2 2013 is 2013.25, q3 2013 is
%2013.5, and q4 2013 is 2013.75
D = [A(:,1) + .25*(A(:,2)-1) A(:,3)]

tbl=table(D(:,1),D(:,2),'VariableNames',{'X','Y'});

%test each of the models using fitlm
mdl1 = fitlm(tbl,'Y~1')
MC1 = [mdl1.ModelCriterion.AIC,mdl1.ModelCriterion.BIC]

```

```

mdl2 = fitlm(tbl, 'Y~X')
MC2 = [mdl2.ModelCriterion.AIC,mdl2.ModelCriterion.BIC]

mdl3 = fitlm(tbl, 'Y~X^2-X')
MC3 = [mdl3.ModelCriterion.AIC,mdl3.ModelCriterion.BIC]

mdl4 = fitlm(tbl, 'Y~X^2')
MC4 = [mdl4.ModelCriterion.AIC,mdl4.ModelCriterion.BIC]

comp = [ MC1 ; MC2 ; MC3 ; MC4]
[M,I] = min(comp)

%select the best model based on lowest AIC
select = min(I(1),I(2))

%give a warning if the two measurements do not agree
if I(1)==I(2);
else
    disp('Warning: AIC and BIC do not agree on solution')
end

%Set the value for the prediction to be one quarter later than the most
%recent demand value provided
Xnew = D(end/2) + .25

%display value for the estimate of the selected model
p=[]
p(1) = predict(mdl1,Xnew)
p(2) = predict(mdl2,Xnew)
p(3) = predict(mdl3,Xnew)
p(4) = predict(mdl4,Xnew)
p(select)

%Change the prediction time period into the old form with the prediction
%attached
Z1 = [mod((Xnew/.25),4)+1,floor(Xnew),p(select)]

%Print the results to a text file.
fileID = fopen('Results.txt','w');
formatSpec = ...
    'The predicted demand in Quarter %1.0f of %4.0f is %8.4f units.\n';
fprintf(fileID,formatSpec,Z1);
fclose(fileID);

```

## 5 Appendix

### Regression Example Code

```
%=====
%This code was used for solving the example problem in the regression
%section
%=====

%Generate 100 numbrers from 0-100 from a uniform distribution and order them
X = 0 + (100)*rand(100,1)
X = sort(X)

%Generate Y values based on the function Y = 45 +.015X^2
Y2= []
for a = 1:100;
    Add = 45 + (20)*randn + .015*X(a)^2;
    Y2 = [Y2;Add];
end

Y = Y2

%Create scatter plot for data
scatter(X,Y, 'x')

%Test the model Y = aX + b and create appropriate plots
tbl = table(X,Y)
mdl = fitlm(tbl, 'Y~X')
plot(mdl)
plotResiduals(mdl, 'fitted')
mdl.ModelCriterion.AIC

%Test the model Y = aX^2 + bX + c
mdl = fitlm(tbl, 'Y~X^2')

%Test the model Y = aX^2 + b and create appropriate plots
mdl = fitlm(tbl, 'Y~X^2-X')
plot(mdl)
plotResiduals(mdl, 'fitted')

%Compare result to original function
hold on
plot(X,45+.015*X.^2)
hold off
```

## Data Generation Code

```
%=====
%This program generates data that will be used to test the effectiveness of
%the regression program in choosing the best model for the data.
%=====

%sets the random seed for consistency when testing and for replicability
rng('default')

%Creating Demand data for quarters from 2007-2016
A = 2003:1:2015
B = 1:1:4
[B,A] = meshgrid(B,A)
C = cat(2,A',B')
D = reshape(C, [], 2)
Z = [D(:,1) + .25*(D(:,2)-1)]
%function is same as  $2.4Z^2 - 9.55e3 Z + 9.5992e6$ 
for num = 1:52
    E(num) = ( 200 + .5*(Z(num)-2000)+ 2.4*(Z(num)-2000)^2 )+ 50*randn;
end
F = [D E']

%Writes the data into a text file called file.txt
name = {'Year';'Quarter';'Demand'}
T = table(F(:,1),F(:,2),F(:,3), 'VariableNames', name)
writetable(T, 'file.txt', 'Delimiter', '\t', 'WriteRowNames', true)
```

## References

- [1] Arsham, H. (2015). Deterministic Modeling: Linear Optimization with Applications. Retrieved May 6, 2015.
- [2] Attaway, S. (2013). MATLAB: A Practical Introduction to Programming and Problem Solving. Waltham, MA: Elsevier.
- [3] Bixby, R. E. (2012). A brief history of linear and mixed-integer programming computation. Documenta Mathematica, pp. 107-121.
- [4] Dantzig, G. B. (1951). Application of the Simplex Method to a Transportation Problem. In T. C. Koopmans (Ed.), Activity Analysis of Production and Allocation. pp. 359-373. New York: John Wiley & Sons, Inc.
- [5] Dantzig, G. B. (2002). Linear Programming. Operations Research 50(1): pp. 42-47.
- [6] Sierksma, G. (2001). Linear and Integer Programming: Theory and Practice. New York, NY: CRC Press.
- [7] Klee, Victor, Minty, George J. (1972). How good is the simplex algorithm?. In Shisha, Oved. Inequalities III. New York-London: Academic Press. pp. 159-175
- [8] Nicholson, K.W. (1990). Elementary Linear Algebra. Boston, MA: PWS-Kent.
- [9] Reeb, J.E., & Leavengood, S.A. (2002). Transportation Problem: A Special Case for Linear Programming Problems. Corvallis, OR: Extension Service, Oregon State University.
- [10] Vanderbei, Robert J. (2008). Linear Programming Foundations and Extensions. New York, NY: Springer.
- [11] Xie, F. (2010). An Example of Degeneracy in Linear Programming. Retrieved May 6, 2015.