Curve Fitting and Parameter Estimation

Glenn Lahodny Jr.

Spring 2015

1 Least Squares Regression

The first step of the modeling process often consists of simply looking at data graphically and trying to recognize trends. In this section, we will study the most standard method of curve fitting and parameter estimation, least squares regression.

Example 1.1. Suppose that eBay hires us to predict its net income for the year 2003, based on its net income for 2000, 2001, and 2002 (see Table 1.1).

Year	Net Income
2000	\$48.3 million
2001	\$90.4 million
2002	\$249.9 million

Table 1.1: Annual net income for eBay.

We begin by simply plotting this data as a *scatterplot* of points. In MATLAB, enter the following commands to obtain Figure 1.1.

```
>> Year=[0 1 2];
>> Income=[48.3 90.4 249.9];
>> plot(Year,Income,'ro')
>> xlabel('Years Since 2000')
>> ylabel('Income (in Millions)')
>> axis([-0.5 2.5 25 275])
```

Our first approach toward predicting eBay's future profits might be to simply find a curve that best fits this data. The most common form of curve fitting is *linear least squares regression*.

1.1 Polynomial Regression

In order to develop an idea of what we mean by "best fit" in this context, we begin by trying to draw a line through the three points of Example 1.1, in such a way that the distance between the points and the line is minimized.

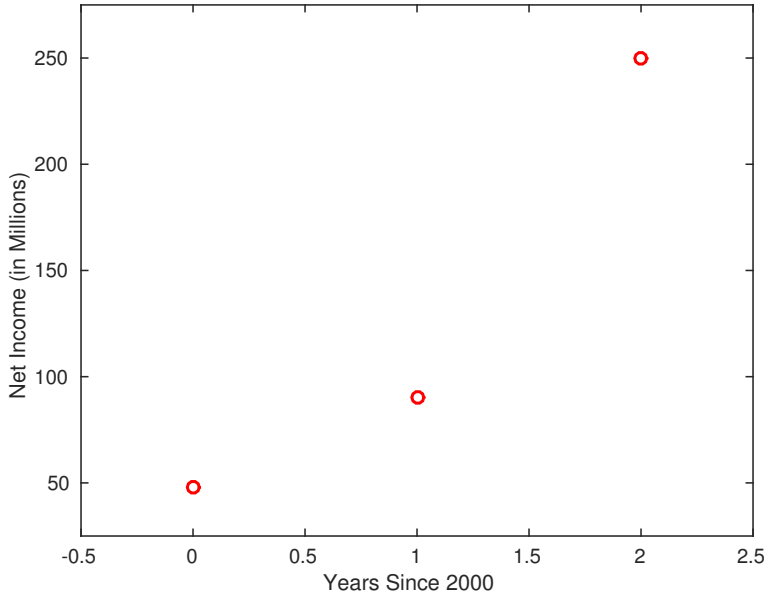

Figure 1.1: eBay net income by year

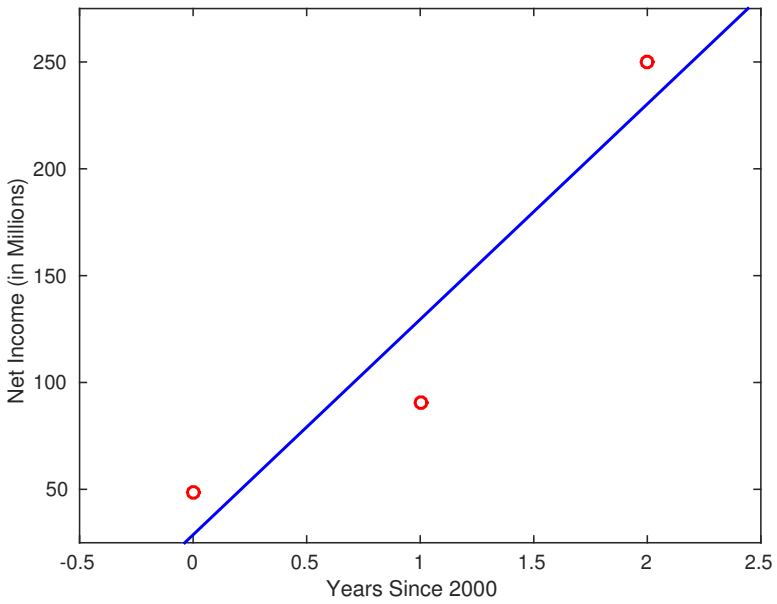

Figure 1.2: Linear least squares curve

Labeling our three points (x_1, y_1), (x_2, y_2), and (x_3, y_3), we observe that the vertical distance between the line and the point (x_i, y_i) is given by the error $E_i = |y_i - mx_i - b|$. The idea behind the least squares method is to sum these vertical distances and minimize the total error. In practice, we square the errors to keep them positive and to avoid possible difficulty with differentiation (due to the absolute values), which will be required for minimization. Our total least squares error is

$$E(m, b) = \sum_{k=1}^{n} (y_k - mx_k - b)^2.$$

We note that instead of these vertical distances, we could also use horizontal distances between the points and the line. Alternatively we could use direct distances (the shortest distances between the points and the line). While either of these methods can be carried out for a best fit line, they become considerably more complicated in the case of more general curves. In the case of a parabola, for example, a point would have two different horizontal distances from the curve, and while it could only have one shortest distance to the curve, computing that distance would be a fairly complicated problem.

Returning to our example, our goal is not to find values of m and b that minimize the error function $E(m, b)$. In order to maximize or minimize a function of multiple variables, we compute the partial derivatives with respect to each of the variables and set them equal to zero. Here, we compute

$$\begin{aligned} \frac{\partial E}{\partial m} &= -2 \sum_{k=1}^{n} x_k (y_k - mx_k - b) = 0, \\ \frac{\partial E}{\partial b} &= -2 \sum_{k=1}^{n} (y_k - mx_k - b) = 0, \end{aligned}$$

which can be solved as a linear system of two equations for the two unknowns m and b. Rearranging the terms and dividing by 2 gives

$$\begin{aligned} m \sum_{k=1}^{n} x_k^2 + b \sum_{k=1}^{n} x_k &= \sum_{k=1}^{n} x_k y_k, \\ m \sum_{k=1}^{n} x_k + b \sum_{k=1}^{n} 1 &= \sum_{k=1}^{n} y_k. \end{aligned}$$

Since $\sum_{k=1}^{n} 1 = n$, we multiply the second equation by $\frac{1}{n} \sum_{k=1}^{n} x_k$ and subtract it from the first to get the relation

$$m \left[\sum_{k=1}^{n} x_k^2 - \frac{1}{n} \left(\sum_{k=1}^{n} x_k \right)^2 \right] = \sum_{k=1}^{n} x_k y_k - \frac{1}{n} \left(\sum_{k=1}^{n} x_k \right) \left(\sum_{k=1}^{n} y_k \right).$$

or

$$m = \frac{\displaystyle \sum_{k=1}^{n} x_k y_k - \frac{1}{n} \left(\sum_{k=1}^{n} x_k \right) \left(\sum_{k=1}^{n} y_k \right)}{\displaystyle \sum_{k=1}^{n} x_k^2 - \frac{1}{n} \left(\sum_{k=1}^{n} x_k \right)^2}.$$

Finally, substituting m into the one of the previous equations yields

$$\begin{aligned} b &= \frac{1}{n}\sum_{k=1}^{n} y_k - \left(\sum_{k=1}^{n} x_k\right) \frac{\displaystyle\sum_{k=1}^{n} x_k y_k - \frac{1}{n}\left(\sum_{k=1}^{n} x_k\right)\left(\sum_{k=1}^{n} y_k\right)}{\displaystyle\sum_{k=1}^{n} x_k^2 - \frac{1}{n}\left(\sum_{k=1}^{n} x_k\right)^2} \\ &= \frac{\left(\displaystyle\sum_{k=1}^{n} y_k\right)\left(\sum_{k=1}^{n} x_k^2\right) - \left(\sum_{k=1}^{n} x_k\right)\left(\sum_{k=1}^{n} x_k y_k\right)}{\displaystyle n\sum_{k=1}^{n} x_k^2 - \left(\sum_{k=1}^{n} x_k\right)^2}. \end{aligned}$$

By continuity, there must be at least one local minimum for E. Since m and b are uniquely determined, these values yield the minimum error.

Observe that we can proceed similarly for any polynomial. For second degree polynomials of the form $y = a_0 + a_1 x + a_2 x^2$, the error becomes

$$E(a_0, a_1, a_2) = \sum_{k=1}^{n} (y_k - a_0 - a_1 x_k - a_2 x_k^2)^2.$$

In this case, we must compute a partial derivative of E with respect to each of the three parameters, and consequently solve three linear equations for the three unknowns.

The MATLAB command for polynomial fitting is *polyfit(x,y,n)*, where x and y are vectors representing the data and n is the degree of the polynomial. For the eBay data, we have

```
>> polyfit(Year,Income,1)
ans =
100.8000 28.7333
```

Note that, left to right, MATLAB returns the coefficient of the highest power of x first, the second highest power of x second, etc., continuing until the y-intercept is given last. Alternatively, for polynomial fitting up to degree 10, MATLAB has the option of choosing it directly from the graphics menu. In the case of our eBay data, while Figure 1 is displayed in MATLAB, we choose **Tools, Basic Fitting**. A new window opens and offers a number of fitting options. We can easily experiment by choosing the **linear** option and then the **quadratic** option, and comparing. (Since we only have three data points in this example, the quadratic fit necessarily passes through all three points. This, of course, does not mean that the quadratic fit is best, only that we need more data.) For this small a set of data, the linear fit is safest, so select that one and click on the black arrow at the bottom right corner of the menu. Checking that the fit given in the new window is linear, select the option **Save to workspace**. MATLAB saves the polynomial fit as a *structure*, which is a MATLAB array variable that can hold data of varying types. The elements of a structure can be accessed through the notation *structurename.structureelement*. Here, the default structure name is *fit*, and the first element is *type*. The element *fit.type* contains a string describing the structure.

The second element of the structure is *fit.coeff*, which contains the polynomial coefficients of our fit. Finally, we can make a prediction with the MATLAB command *polyval(P,X)*. This command returns the value of a polynomial P evaluated at X. In particular, P is a vector of length $n+1$ whose elements are the coefficients of the polynomial in descending powers. For the eBay example, we have

```
>> polyval(fit.coeff,3)
ans =
  331.1333
```

We obtain the prediction 331.1333 or approximately \$331.13 million in net income. Finally, we mention that MATLAB refers to the error for its fit as the *norm of the residuals*, which is precisely the square root of E as we've defined it.

Example 1.2. (Crime and Unemployment) Suppose we are asked to model the connection between unemployment and crime in the United States during the period 1994–2001. We might suspect that in some general way increased unemployment leads to increased crime, but our first step is to collect data. First we contact the Federal Bureau of Investigation and study their Uniform Crime Reports (UCRs), which document, among other things, the United States' crime rate per 100,000 citizens. Next, we contact the U.S. Bureau of Labor and obtain unemployment percentages for each year in our time period. The data is summarized in Table 1.2.

Year	Crime Rate	Percent Unemployment
1994	5,373.5	6.1%
1995	5,277.6	5.6%
1996	5,086.6	5.4%
1997	4,922.7	4.9%
1998	4,619.3	4.5%
1999	4,266.8	4.2%
2000	4,124.8	4.0%
2001	4,160.5	4.8%

Table 1.2: United States crime rate and unemployment data.

As in Example 1.1, we begin by looking at a scatterplot of the data (see Figure 1.3).

Certainly the first thing that we observe about the scatterplot is that their does seem to be a distinct connection: as unemployment increases, crime rate increases. In fact, aside from the point for 2001, the trend appears fairly steady. In general, we would study this point at the year 2001 very carefully and try to deterine whether this is an anomaly or a genuine shift in the paradigm. For the purposes of this example, however, we're going to treat it as an *outlyer*— a point that for one reason or another does not follow an otherwise genuinely predictive model. The important point to keep in mind is that discarding outlyers when fitting data is a valid approach, *as long as you continue to check and reappraise your model as future data becomes available*. Discarding the outlyer, we try both a linear and quadratic fit, each shown in Figure 1.5.

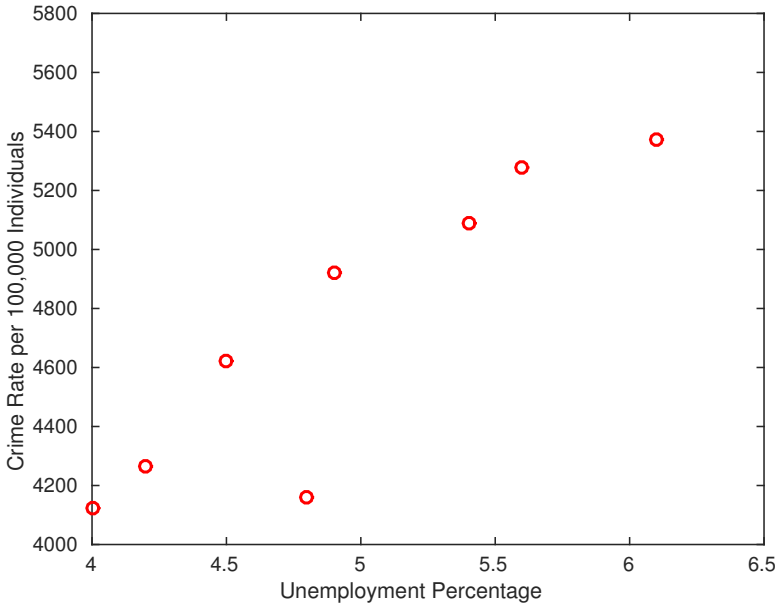

Figure 1.3: Scatterplot of crime rate versus unemployment percentage in the U.S.

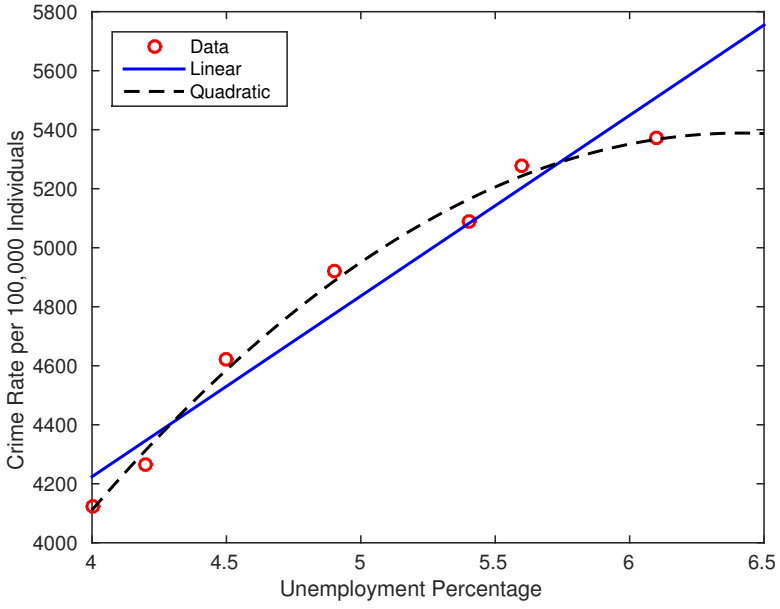

Figure 1.4: Best fit curves for crime and unemployment data.

Clearly the quadratic fit is better, and though we have not yet quantitatively developed the socioeconomic reasons for this particular relation, we do have a genuinely predictive model that we can test against future data. For example, the unemployment percentage in 2002 was 5.8%, which our quadratic model would predict should be associated with a crime rate of 5,306.2 crimes per 100,000 individuals. The actual crime rate for 2002 was 4118.8. At this point, we are led to conclude that our model is not sufficiently accurate. In this case, the problem is most likely a lag effect. While a short-term rise in the unemployment rate does not seem to have much effect on crime, perhaps a sustained rise in unemployment would.

1.2 Alternative Methods of Curve Fitting

Although least squares regression is the most popular form of basic curve fitting, it is not the only method that can be applied. We could also think, for example, of defining our error as the maximum of all the vertical distances between the points and our curve. That is, for n data points, we could define

$$E = \max_{k=1,\ldots,n} |y_k - f(y_k)|.$$

This is called the Chebyshev method. We could also sum the unsquared absolute values,

$$E = \sum_{k=1}^{n} |y_k - f(x_k)|,$$

or take any other reasonable measure.

1.3 Regression with More General Functions

Example 1.3. Yearly temperature fluctuations are often modeled by trigonometric expressions, which lead to more difficult regression analyses. Here, we'll consider the case of monthly average maximum temperatures in Big Bend National Park. In Table 1.3 the first column lists raw data of average maximum temperatures each month. In order to model this data with a simple trigonometric model, we'll subtract the mean (which gives Column 3) and divide by the maximum absolute value (which gives Column 4) to arrive at a column of dependent variables that vary like sine and cosine between -1 and 1.

A reasonable model for this data is $T(m) = \sin(m - a)$, where T represents scaled temperatures and m represents a scaled index of months ($m \in [0, 2\pi]$), where by our reductions we've limited our analysis to a single parameter, a. Proceeding as above, we consider the regression error

$$E(a) = \sum_{k=1}^{n} (T_k - \sin(m_k - a))^2.$$

To minimize the error, we set the derivative $E'(a)$ equal to zero which gives

$$\frac{dE}{da} = 2 \sum_{k=1}^{n} (T_k - \sin(m_k - a)) \cos(m_k - a) = 0,$$

Month	Average Max Temp.	Minus Mean	Scaled
Jan	60.9	-18.0	-1
Feb	66.2	-12.7	-0.71
Mar	77.4	-1.5	-0.08
Apr	80.7	1.8	0.10
May	88.0	9.1	0.51
Jun	94.2	15.3	0.85
Jul	92.9	14.0	0.78
Aug	91.1	12.2	0.68
Sep	86.4	7.5	0.42
Oct	78.8	-0.1	-0.01
Nov	68.5	-10.4	-0.58
Dec	62.2	-16.7	-0.93

Table 1.3: Average maximum temperatures for Big Bend National Park.

a nonlinear equation for the parameter a. Though nonlinear algebraic equations are typically difficult to solve analytically, they can be solved numerically.

In this case, we will use MATLAB's *fzero(FUN,X0)* function which tries to find a zero of the function FUN near X0, if X0 is a scalar. First, we write a script M-file that contains the function we're setting to zero, listed below as *bigbend.m*.

```
% Glenn Lahodny Jr.
% Math 442 - Mathematical Modeling

% This function defines the derivative of the error function needed to
% minimize error for fitting average maximum temperatures for Big Bend
% National Park.

function E = bigbend(a)

% Scaled Temperatures
T=[-1 -0.71 -0.08 0.1 0.51 0.85 0.78 0.68 0.42 -0.01 -0.58 -0.93];

% Initial Value
E=0;

% Loop defining the derivative of the error function.
for k=1:12
    m=2*pi*k/12;
    E=E+(T(k)-sin(m-a))*cos(m-a);
end
```

Finally, we solve for a and compare our model with the data, arriving at Figure 1.5.

```
% Glenn Lahodny Jr.
% Math 442 - Mathematical Modeling

clear

% Data
Months=1:12;
Temps=[60.9 66.2 77.4 80.7 88 94.2 92.9 91.1 86.4 78.8 68.5 62.2];

% Computes the zeros of the derivative of the error function defined by
% bigbend.m needed to obtain the best fit.
a=fzero('bigbend',1);

modeltemps=18*sin(2*pi*Months/12-a)+mean(Temps);

% Plot Information
plot(Months,Temps,'ro',Months,modeltemps,'b-','linewidth',1.5)
xlabel('Months')
ylabel('Average Max Temperature')
legend('Data','Model')
```

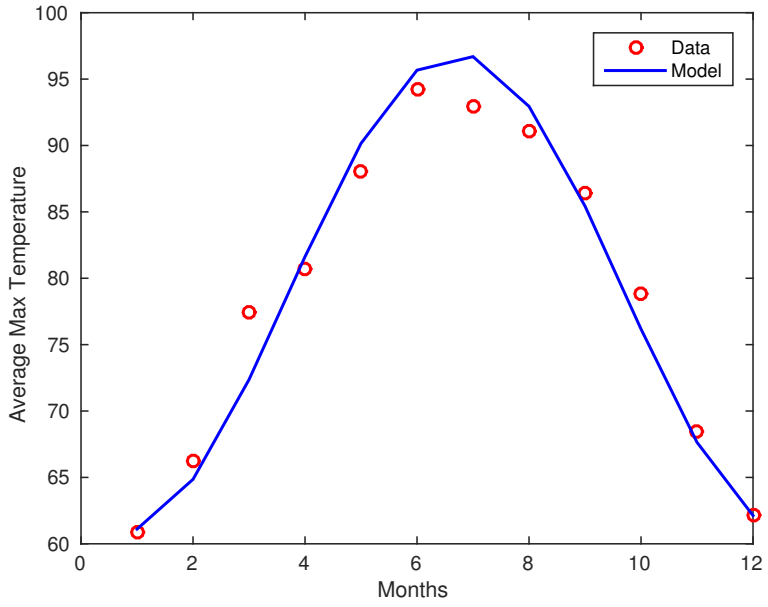

Figure 1.5: Trigonometric model for average monthly temperatures in Big Bend.

For models with multiple parameters, we must solve a system of nonlinear equations, which can be quite difficult in general, even computationally. The MATLAB function *lsqcurvefit* can be used to solve nonlinear least squares problems.

Example 1.4. Let us consider a model for population growth in Bryan, TX from 1900–2010 (see Table 1.4).

Year	Population	Year	Population
1900	3,589	1960	27,542
1910	4,132	1970	33,141
1920	6,307	1980	44,337
1930	7,814	1990	55,002
1940	11,842	2000	65,660
1950	18,072	2010	76,201

Table 1.4: Population of Bryan, TX from 1900–2010 according to the U.S. Census.

Let $N(t)$ denote the population size (in thousands) at time $t \geq 0$. The Verhulst (logistic) growth equation is given by

$$\frac{dN}{dt} = rN\left(1 - \frac{N}{K}\right), \quad N(0) = N_0$$

where $N_0 > 0$ is the initial population, $r > 0$ is a growth parameter for the population, and $K > 0$ is the carrying capacity. This differential equation can be solved using separation of variables and partial fractions. The solution is given by

$$N(t) = \frac{N_0 K}{N_0 + (K - N_0)e^{-rt}}.$$

Although the time $t = 0$ corresponds to the year 1900, we assume that the estimate that year was fairly crude and obtain a value of N_0 by fitting the entirety of the data. In this way, we have three parameters to solve for, and carrying out the full regression analysis can be tedious.

The first step in finding values for the parameters with MATLAB consists of defining the model, $N(t)$, as a function M-file, with the three parameters N_0, r, and K stored as a vector $p = (p(1), p(2), p(3)) = (N_0, r, K)$.

```
% Glenn Lahodny Jr.
% Math 442 - Mathematical Modeling

% This function computes the logistic model N=N0*K/(N0+(K-N0)*exp(-r*t)).

function N = logistic(ArK,t)

N=p(1)*p(3)./(p(1)+(p(3)-p(1))*exp(-p(2).*t));
```

Next, we write a MATLAB script M-file which uses the function *lsqcurvefit* to obtain least squares parameter estimates for N_0, r, and K.

```
% Glenn Lahodny Jr.
% Math 442 - Mathematical Modeling

% This script uses the function lsqcurvefit to fit the solution of the
% logistic growth model to the Bryan, TX population data for 1790-2000.

clear
close all

% Data
Time=0:10:110;
Pops=[3.589 4.132 6.307 7.814 11.842 18.072 27.542 33.141 44.337...
    55.002 65.660 76.201];
Year=Time+1900;

% Initial Guess for Parameters
N0=3.589;
r=0.01;
K=2*max(Pops);
p0=[N0 r K];

% Nonlinear Least Squares Best Fit
[p,Error] = lsqcurvefit('logistic',p0,Time,Pops)
N = logistic(p,Time);

% Plot Information
plot(Year,Pops,'ro',Year,N,'b-','linewidth',1.5)
xlabel('Year')
ylabel('Population (in Thousands)')
legend('Data','Logistic Model','location','NorthWest')
```

After defining the data, we entered our initial guess as to what the parameter values should be, the vector p_0. (It should be noted that MATLAB uses a routine similar to *fzero*, and can typically only find roots reasonably near our guess.) In this case, we guessed a small value of r, a carrying capacity of twice the maximum population in the dataset, and an initial population equal to the population in 1900. Finally, we use *lsqcurvefit*, entering our function file, our initial parameter guesses, and our data, respectively.

The function *lsqcurvefit* provides two outputs, our parameters and a sum of squared errors, which we call "Error". Although the error seems large, keep in mind that this is a sum of all errors squared:

$$\text{Error} = \sum_{\text{Time}} (\text{Pops}(\text{Time}) - N(\text{Time}))^2.$$

The output is displayed in the MATLAB Command Window:

```
p =
    2.7583    0.0410  107.8958
Error =
    9.0175
>> sqrt(Error)
ans =
    3.0029
```

A more reasonable measure of error is the square root of this value, from which we see that over 12 decades our model is only off by around 3,000 people. The data and logistic fit are illustrated in Figure 1.6.

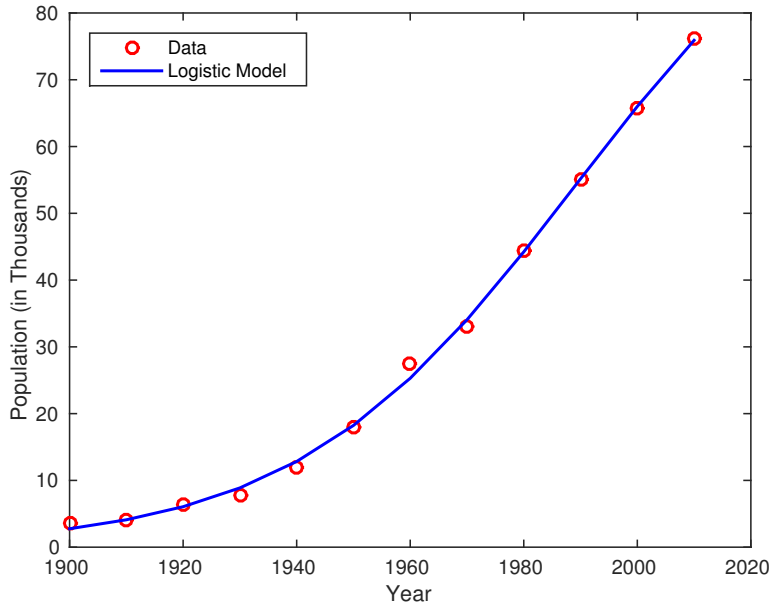

Figure 1.6: Population data for Bryan, TX and the logistic model approximation.

1.4 Transformations to Linear Form

In the previous examples, we found that nonlinear regression can be used to estimate the parameters for arbitrary functions. So it is not required that our functions be in linear form. However, lines are certainly the easiest fits to recognize visually, and so it is often worthwhile to at least look for a linear relationship for your data.

Each of the following nonlinear relationships can be put into a linear form:

$$\begin{aligned} y &= ax^b \\ y &= a \cdot b^x \\ y &= \frac{1}{ax+b}. \end{aligned}$$

This is not an exhaustive list, but these are some very useful examples. For example, if we suspect our data has the form $y = ax^b$, we might take the natural logarithm of both sides to write

$$\ln y = \ln a + b \ln x.$$

If we now plot $\ln y$ versus $\ln x$, the slope of the plot will be b and the y-intercept will be $\ln a$, from which we can obtain a value for a by exponentiation.

1.5 Parameter Estimation Directly from Differential Equations

In modeling a new phenomenon, we often find that we can write down a differential equation that models the phenomenon, but we cannot solve the differential equation analytically. We would like to solve it numerically, but all of our techniques thus far for parameter estimation assume we know the exact form of the function.

1.5.1 Derivative Approximation Method

Example 1.5. In population models, we often want to study the interaction between various populations. One of the simplest interaction models is the Lotka-Volterra predator-prey model

$$\begin{aligned} \frac{dx}{dt} &= (a_1 - b_1 y)x, \\ \frac{dy}{dt} &= (-a_2 + b_2 x)y, \end{aligned}$$

where $x(t)$ and $y(t)$ denote the number of prey and predators at time $t \geq 0$, respectively. Observe that the interaction terms, $-b_1xy$ and b_2xy correspond to death of prey in the presence of predators and proliferation of predators in the presence of prey, respectively.

The Lotka-Volterra predator-prey model is a classic population model. However, it is often too simple to capture the complex dynamics of interacting species. One classic example in which the model describes the interaction fairly well is the interaction between lynx and hares, as measured by the number of pelts collected by the Hudson Bay Company from 1900–1920. The data from the Hudson Bay Company is given in Table 1.5.

Year	Lynx	Hare	Year	Lynx	Hare	Year	Lynx	Hare
1900	4.0	30.0	1907	13.0	21.4	1914	45.7	52.3
1901	6.1	47.2	1908	8.3	22.0	1915	51.1	19.5
1902	9.8	70.2	1909	9.1	25.4	1916	29.7	11.2
1903	35.2	77.4	1910	7.4	27.1	1917	15.8	7.6
1904	59.4	36.3	1911	8.0	40.3	1918	9.7	14.6
1905	41.7	20.6	1912	12.3	57.0	1919	10.1	16.2
1906	19.0	18.1	1913	19.5	76.6	1920	8.6	24.7

Table 1.5: Pelts collected by the Hudson Bay Company (in thousands).

Our goal is to estimate values of a_1, b_1, a_2, and b_2 without finding an exact solution of the differential equations. Assuming the predator population is nonzero, the differential equation for y can be written as

$$\frac{1}{y}\frac{dy}{dt} = b_2 x - a_2.$$

By treating the expression $\frac{1}{y}\frac{dy}{dt}$ as a single variable, we see that b_2 and a_2 are the slope and intercept of a line, respectively. Thus, we would like to plot the values of $\frac{1}{y}\frac{dy}{dt}$ versus x and fit a line through the data. Since we have the values of x and y, the only difficulty is in finding the values of $\frac{dy}{dt}$. In order to do this, recall the definition of a derivative

$$\frac{dy}{dt} = \lim_{h\to 0} \frac{y(t+h) - y(t)}{h}.$$

Following the idea behind Euler's method for numerically solving differential equations, we conclude that for h sufficiently small,

$$\frac{dy}{dt} \approx \frac{y(t+h) - y(t)}{h},$$

which is called the *forward difference* approximation.

How good is this approximation? To answer this question, recall that the Taylor series for a function $f(x)$ centered at $x = a$ is given by

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots.$$

Setting $x = t + h$ and $a = t$, we have the expansion

$$f(t+h) = f(t) + f'(t)h + \frac{f''(t)}{2}h^2 + \frac{f'''(t)}{3!}h^3 + \cdots.$$

Finally, subtracting $f(t)$ from both sides and dividing by h gives the approximation

$$\frac{f(t+h) - f(t)}{h} = f'(t) + \frac{f''(t)}{2}h + \frac{f'''(t)}{3!}h^2 + \cdots.$$

It follows that the error in our approximation is proportional to h. Thus, the forward difference approximation is an *order one* approximation. This can be written as

$$f'(t) = \frac{f(t+h) - f(t)}{h} + \mathcal{O}(h),$$

where $g(h) = \mathcal{O}(h)$ means that $|\frac{g(h)}{h}|$ remains bounded as $h \to 0$.

Can we do better than this approximation? Consider the *central difference* approximation

$$\frac{dy}{dt} \approx \frac{y(t+h) - y(t-h)}{2h}.$$

It is not difficult to show that this is a *second order* approximation. That is,

$$f'(t) = \frac{f(t+h) - f(t-h)}{2h} + \mathcal{O}(h^2).$$

Therefore, we will use the central difference approximation for the data.

Returning to the data, we find that $h = 1$ for our approximation. This value for h is not particularly small. However, our goal is to estimate the parameters, and we can check the validity of our estimates by checking the model against our data. Since we cannot compute a central difference approximation for the first year of data, we begin in 1901 and compute

$$\begin{aligned}\frac{1}{y}\frac{dy}{dt} &\approx \frac{1}{y}\left(\frac{y(t+h)-y(t-h)}{2h}\right)\\ &= \frac{1}{6.1}\left(\frac{9.8-4.0}{2}\right)\\ &= b_2(47.2)-a_2.\end{aligned}$$

Repeating this for each year, up to 1919, we obtain a system of equations which can be solved by regression. This can be done by writing a script M-file in MATLAB.

```
% Glenn Lahodny Jr.
% Math 442 - Mathematical Modeling

% Data
Hare=[30 47.2 70.2 77.4 36.3 20.6 18.1 21.4 22 25.4 27.1 40.3 57 76.6...
    52.3 19.5 11.2 7.6 14.6 16.2 24.7];
Lynx=[4 6.1 9.8 35.2 59.4 41.7 19 13 8.3 9.1 7.4 8 12.3 19.5...
    45.7 51.1 29.7 15.8 9.7 10.1 8.6];

% Central Difference Approximation
for k=1:19
    y(k)=(1/Lynx(k+1))*(Lynx(k+2)-Lynx(k))/2;
    x(k)=Hare(k+1);
end

% Linear Fit
P=polyfit(x,y,1);
L=P(1)*x+P(2);

% Plot Information
plot(x,y,'ro',x,L,'b-','linewidth',1.5)
title('Linear Regression for Predators')
xlabel('x')
ylabel('(1/y)(dy/dt)')
legend('Data','Linear Fit','location','Northwest')
```

The data from the central difference approximation is plotted in Figure 1.7. The best fit line for this data is $y = 0.0234x - 0.7646$. Therefore, we have estimates of $b_2 = 0.0234$ and $a_2 = 0.7646$.

Similarly, we approximate the prey equation using a central difference approximation to obtain the data in Figure 1.8. The best fit Line for this data is $y = -0.0240x + 0.4732$. Therefore, we find estimates of $a_1 = 0.4732$ and $b_1 = 0.024$.

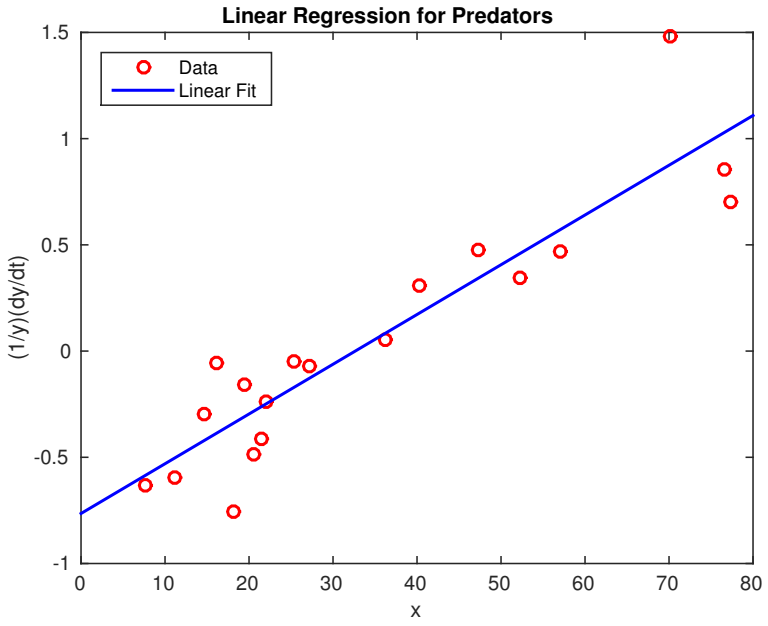

Figure 1.7: Linear fit for predator equation.

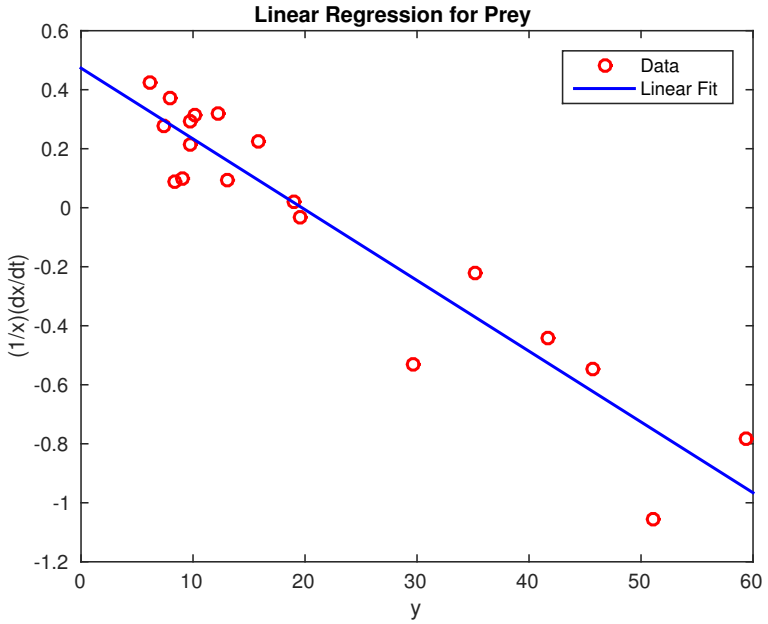

Figure 1.8: Linear fit for prey equation.

Finally, we would like to check our parameter estimates for the model by plotting time series of the numerical solution of the model against the Hudson Bay Company data. To do this, we first define the Lotka-Volterra predator-prey equations with a function M-file.

```
% Glenn Lahodny Jr.
% Math 442 - Mathematical Modeling

% This function defines the Lotka-Volterra predator-prey model.

function dx = LotVolt1(t,x)

% Parameters
a1=0.4732;
a2=0.7646;
b1=0.024;
b2=0.0234;

% ODEs (Here, x1=x and x2=y)
dx=zeros(2,1);
dx(1)=(a1-b1*x(2))*x(1);
dx(2)=(-a2+b2*x(1))*x(2);
```

Then we solve the model using the built-in MATLAB ODE solver *ode45* with the following script M-file.

```
% Glenn Lahodny Jr.
% Math 442 - Mathematical Modeling

% This program solves the Lotka-Volterra predator-prey model defined by
% the function LotVolt1.m using the ODE solver ode45 and plots the
% solutions and data over time.

% Time Vector
Year=1900+0:1:20;

% Maximum Time
tmax=20;

% Time Span
tspan=[0 tmax];

% Initial Conditions
x0=30;
y0=4;
Init=[x0,y0];
```

```
% Solves ODEs
[t,x]=ode45('LotVolt1',tspan,Init);

% Plot Information
subplot(2,1,1)
plot(Year,Hare,'ro',1900+t,x(:,1),'b-','linewidth',1.5)
title('Prey (Hare)')
xlabel('Year')
ylabel('Population (in Thousands)')
legend('Data','Model')

subplot(2,1,2)
plot(Year,Lynx,'ro',1900+t,x(:,1),'b-','linewidth',1.5)
title('Predator (Lynx)')
xlabel('Year')
ylabel('Population (in Thousands)')
legend('Data','Model')
```

The data, along with the numerical solution, are illustrated in Figure 1.9

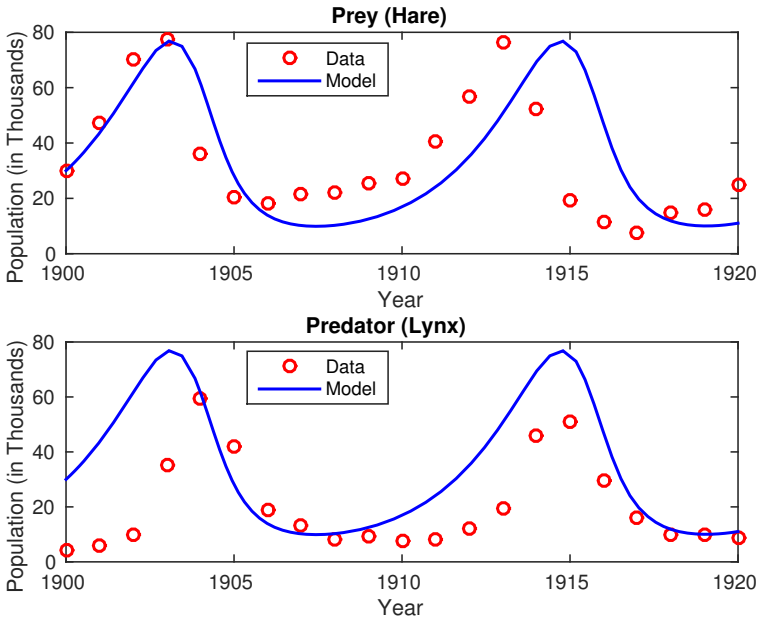

Figure 1.9: Data and model for lynx-hare system.

References

[1] P. Howard, *Modeling Basics*, Lecture Notes for Math 442 - Mathematical Modeling, 2009, `http://www.math.tamu.edu~/phoward/M442.html`.