2006

Estimating and forming confidence intervals for extrema of random polynomials

Sandra DeSousa
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

Recommended Citation

DeSousa, Sandra, "Estimating and forming confidence intervals for extrema of random polynomials" (2006). *Master's Theses*. 2876.
DOI: <https://doi.org/10.31979/etd.qw29-sbbh>
https://scholarworks.sjsu.edu/etd_theses/2876

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

ESTIMATING AND FORMING
CONFIDENCE INTERVALS FOR
EXTREMA OF RANDOM POLYNOMIALS

A Thesis

Presented to

The Faculty of the Department of Mathematics

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Masters of Arts

by

Sandra DeSousa

May 2006

UMI Number: 1436902

Copyright 2006 by
DeSousa, Sandra

All rights reserved.

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform 1436902

Copyright 2006 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

© 2006

Sandra DeSousa

ALL RIGHTS RESERVED

APPROVED FOR THE DEPARTMENT OF MATHEMATICS

Dr. Steven Crunk

Dr. Leslie Foster

Dr. Bee Leng Lee

APPROVED FOR THE UNIVERSITY

04/06/06

ABSTRACT

ESTIMATING AND FORMING CONFIDENCE INTERVALS FOR EXTREMA OF RANDOM POLYNOMIALS

by Sandra DeSousa

Much research has been done on random polynomials. Topics of past investigation include estimating the number of zeros, finding the roots (and associated distributions), and confidence intervals for the roots of random polynomials.

Research regarding the extrema of random polynomials and their associated confidence intervals is lacking. Fairley performed a study on a method for forming the confidence intervals for the roots of random polynomials. This thesis expands upon Fairley's results by forming confidence intervals for the abscissa and ordinate of the extrema of random polynomials. Three methods to calculate the confidence intervals are compared: the Fairley method, the delta method, and bootstrapping. It was determined that all three methods produced accurate confidence intervals that statistically were not significantly different between the methods. An application of the theoretical work was implemented using data provided by the NASA Ames Research Center, associated with the possibility of a runaway greenhouse effect.

ACKNOWLEDGEMENTS

I dedicate this to
Crystal, Kyle, Joelle, and Logan,
my wonderful children,
for their unconditional love,
patience, understanding, and support.

I would like to thank my thesis advisor,
Dr. Steve Crunk,
for all of his knowledge and guidance with this endeavor.

Table of Contents

Table of Contents.....	vi
List of Tables.....	vii
List of Figures.....	ix
Chapter 1: Introduction.....	1
Chapter 2: Methods.....	7
Fairley Method.....	7
Delta Method.....	11
Bootstrapping.....	16
Chapter 3: Implementation.....	18
Simulations.....	18
Results - Confidence Interval Accuracy.....	27
Results - Confidence Interval Lengths.....	31
Results - Overall.....	33
Chapter 4: Empirical Application.....	36
Chapter 5: Conclusions.....	40
References.....	45
Appendix: S-Plus® Code.....	46

List of Tables

Table 3.1. Percentages of times that the true abscissa lies within the calculated confidence limits for a second degree random polynomial.....	28
Table 3.2. Percentages of times that the true ordinate lies within the calculated confidence limits for a second degree random polynomial.....	28
Table 3.3. Percentages of times that the true abscissa lies within the calculated confidence limits for a third degree random polynomial.....	29
Table 3.4. Percentages of times that the true ordinate lies within the calculated confidence limits for a third degree random polynomial.....	29
Table 3.5. Percentages of times that the true abscissa lies within the calculated confidence limits for a fourth degree random polynomial.....	30
Table 3.6. Percentages of times that the true ordinate lies within the calculated confidence limits for a fourth degree random polynomial.....	30
Table 3.7. Average lengths of the abscissa confidence intervals for a second degree random polynomial.....	31
Table 3.8. Average lengths of the ordinate confidence intervals for a second degree random polynomial.....	31
Table 3.9. Average lengths of the abscissa confidence intervals for a third degree random polynomial.....	32
Table 3.10. Average lengths of the ordinate confidence intervals for a third degree random polynomial.....	32
Table 3.11. Average lengths of the abscissa confidence intervals for a fourth degree random polynomial.....	33

List of Tables (cont'd)

Table 3.12. Average lengths of the ordinate confidence intervals for a fourth degree random polynomial.....	33
Table 4.1. Calculated abscissa and ordinate confidence intervals for the maximum of sea surface temperature vs. outgoing flux.....	39

List of Figures

Figure 2.1. Illustration of nonparametric bootstrap.....	17
Figure 3.1. One example of a second degree random polynomial with normal distribution noise and associated confidence intervals calculated via the delta method.....	21
Figure 3.2. Close up of example of a second degree random polynomial with normal distribution noise and associated confidence intervals.....	22
Figure 3.3. One example of a third degree random polynomial with exponential distribution noise and associated confidence intervals calculated via bootstrapping method.....	23
Figure 3.4. One example of a fourth degree random polynomial with t distribution noise and associated confidence intervals calculated via the Fairley method.....	25
Figure 3.5. Plot of $(\hat{x}_m, \hat{y}_m)$ for all simulations of the third degree polynomial with normal noise.....	26
Figure 4.1. Graph of Sea Surface Temperature vs Outgoing Flux with estimated random polynomial and associated maximum.....	38

Chapter 1

Introduction

The topic of regression analysis is widely used in statistics.

According to Myers (1990), "The term regression analysis describes a collection of statistical techniques that serve as a basis for drawing inferences about relationships among quantities in a scientific system."

Regression analysis allows scientists to make predictions, perform variable screening, parameter estimation, and model specification. The most common uses of regression analysis are predicting response values for given inputs and measuring the importance of a variable (e.g., its influence on the response).

When given a set of input (regression variable, x) and response (y) data, regression is often used to fit a curve, $p(x)$, to the data. If the form of the data is assumed to be $m(x) = y = \beta_0 + \beta_1 x + \beta_2 x^2 + \dots + \beta_k x^k$ (a true but unknown polynomial), then a curve fit to a set of data is referred to as a random algebraic polynomial, such as $p(x) = \hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + \dots + \hat{\beta}_k x^k$. Here, k is the degree of the polynomial, $\beta_0, \beta_1, \dots, \beta_k$ are unknown constants known as regression coefficients and $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_k$ are estimates of these coefficients.

As an example of a typical regression, there are a set of n observations $\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_n, y_n)\}$, and we are interested in finding a curve, $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + \dots + \hat{\beta}_k x^k$ that is the best representation for the set of data. Each point in the set of data (x_i, y_i) can be represented as $y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \dots + \beta_k x_i^k + \varepsilon_i$ where we often assume that the errors, ε_i , are normally distributed with mean zero and standard deviation of σ (which is unknown). In matrix form, we have a system of

$$\text{linear equations: } \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^k \\ 1 & x_2 & x_2^2 & \cdots & x_2^k \\ 1 & x_3 & x_3^2 & \cdots & x_3^k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^k \end{pmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \vdots \\ \varepsilon_n \end{bmatrix} \quad (1.1)$$

or $\underline{Y} = \underline{X} \underline{\beta} + \underline{\varepsilon}$. There are several ways to estimate the coefficients, $\beta_0, \beta_1, \dots, \beta_k$, of the polynomial. The most common are the method of least squares, maximum likelihood estimation, and projection. Under the assumption of normal errors, each of these methods leads to the same solution. Using linear algebra to solve the system of linear equations above we get $\hat{\underline{\beta}} = (\underline{X}' \underline{X})^{-1} \underline{X}' \underline{Y}$, where $\underline{X}'$ indicates the transpose of the $\underline{X}$ matrix. So, $\hat{\underline{\beta}}$ is a linear combination of the elements of $\underline{Y}$ which are normal, as the $\underline{X}$ matrix is assumed to be fixed. Therefore, as the ε_i are

assumed normal, the y_i are also normal and it follows that $\hat{\beta}$ follows a normal distribution.

In the study of random polynomials, of particular interest are the roots (or zeros) of the equation: $p(x) = \hat{y} = 0$. Much research has been conducted in estimating the number of real roots along with distributions of the number of real roots. Additional investigation has been conducted in estimating the roots of random polynomials and their associated distributions. Here, we are going to focus on finding confidence intervals for the abscissa (x value) and ordinate (y value, or $m(x)$) of the extrema of the true, but unknown, polynomials. The abscissa is the root of the derivative of the random polynomial and the ordinate is the $y=m(x)$ value associated with this abscissa value. The distributions of the estimated extrema (minimum/maximum) of random polynomials are based on the random coefficients generated from the regression fit of the data. In most cases, the random variable coefficients are assumed normally distributed, as described above.

To establish a foundation for the rest of this thesis, we need to introduce some notation. Let (x_m, y_m) be the true abscissa and ordinate for the minimum or maximum of interest of the true, unknown polynomial, $m(x)$. Similarly, let $(\hat{x}_m, \hat{y}_m)$ be the abscissa and ordinate of the extrema of the estimated random polynomial, $p(x)$ (i.e., $\hat{x}_m$ is the

solution to the equation $p'(\hat{x}_m)=0$ and $\hat{y}_m = p(\hat{x}_m)$. From Wackerly, Mendenhall, and Scheaffer (2002), we can define (X_m, Y_m) as random variables associated with the abscissa and ordinate since they vary depending on the results of the regression.

A confidence interval is defined as an interval that contains a random value of interest with some specified level of confidence (e.g., $100(1-\alpha)\%$). For instance, the probability statement for a normally distributed random variable X_m is

$$P\left(x_m - z_{\alpha/2} * \sqrt{Var(X_m)} \leq X_m \leq x_m + z_{\alpha/2} * \sqrt{Var(X_m)}\right) = 1 - \alpha, \text{ where } z_{\alpha/2} \text{ is the } \alpha/2$$

quantile of the normal distribution. After some algebra, this becomes

$$P\left(X_m - z_{\alpha/2} * \sqrt{Var(X_m)} \leq x_m \leq X_m + z_{\alpha/2} * \sqrt{Var(X_m)}\right) = 1 - \alpha, \text{ where, although it}$$

looks like a probability statement for x_m (since it is in the center), this is still a probability statement with respect to the random variable X_m .

Upon replacing the random variable X_m with a value $\hat{x}_m$, estimated from a set of data, we form a confidence interval, $\hat{x}_m \pm z_{\alpha/2} * \sqrt{Var(X_m)}$, for the true value x_m (Myers, 1990).

Just as we define a confidence interval for the true abscissa (x value), similar statements can be made for the ordinate (y value) of the extrema. The confidence intervals for the abscissa and ordinate values, together, create a confidence region for the true extrema of the random

polynomial. Assuming the estimated abscissa and ordinate are uncorrelated, which we have found through simulation to be approximately true, the confidence region would be expected to be an oval. However, we simply present confidence regions that look like rectangular boxes that are independent confidence intervals for each of the abscissa and ordinate of the extrema, not a joint confidence interval for (x_m, y_m) , the true location of the extrema.

The goal here is to study, prototype, and test three different methods of finding confidence intervals for the extrema of these random polynomials. The three methods investigated included the Fairley method (based on Fieller's Theorem), the delta method (based on Taylor series expansions), and bootstrapping (based on repeated sampling with replacement). Each method is described in detail, and is shown along with simulations used to test each method. An empirical application using data from the NASA Ames Research Center, associated with the possibility of a runaway greenhouse effect, is also presented.

To test each of the methods, data from three different known polynomials (degrees 2, 3, and 4) were simulated. For each known polynomial, three different types of random noise were added to the known polynomials to imitate a set of input and random response data. The random noise came from the normal, exponential, and t

distributions. The details of the polynomials used and noise added will be discussed further in the following chapters.

Chapter 2 gives the mathematical details of each of the confidence interval methods used, as well as theorems and proofs regarding these confidence interval methods. In Chapter 3, we discuss the simulation processes and state the numerical results of the simulations. An empirical application using data from the NASA Ames Research Center is presented in Chapter 4. Finally, Chapter 5 draws conclusions based on the results in Chapter 3 and gives recommendation for areas of future research in this field.

Chapter 2

Methods

Fairley Method

Fairley (1968) expands upon a ratio, initially presented by Fieller (1954), and implies that for any degree random polynomial, the ratio between the estimated polynomial squared and the variance of the polynomial evaluated at the root follows an F -distribution with 1 numerator degree of freedom and $n-(k+1)$ denominator degrees of

freedom (i.e., $\frac{(p(x_0))^2}{\sigma(p(x_0))^2} \sim F_{1, n-(k+1)}$). As mentioned earlier, the variables n

and k represent the sample size of the data (number of observations) and the order of the fitted polynomial respectively. If one notes the difference between Fieller and Fairley with respect to the distributions of the ratios, the rationale for one using the t -distribution while the other uses the F is that the F -distribution with 1 numerator degree of freedom is the equivalent to the square of a t -distribution (Casella and Berger, 2002, p. 255).

Fairley develops the use of the ratio method initially presented by Fieller for the linear and quadratic cases. In fact, Fairley (1968) gives a

definition of the confidence interval for the root (x value at which $m(x)=0$) of a random polynomial as follows: The region on the x-axis where

$$\frac{(p(x_i))^2}{\sigma(p(x_i))^2} < F_{1,n-(k+1)}(\alpha), \quad (2.1)$$

(where $F_{1,n-(k+1)}(\alpha)$ denotes the upper α point of the $F_{1,n-(k+1)}$ distribution), defines a confidence region for the root of the polynomial with confidence coefficient $1 - \alpha$ (p.125). To compute $\sigma(p(x_i))^2$ in Equation (2.1), we start

with $\sigma(p(x_i))^2 = V(p(x_i)) = V(\hat{y}_i) = V(\hat{x}_i' \hat{\beta})$, where $\hat{x}_i = (1, x_i, x_i^2, \dots, x_i^k)'$, and

$\hat{\beta} = (\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_k)$. Now, $V(\hat{x}_i' \hat{\beta}) = \hat{x}_i' V(\hat{\beta}) \hat{x}_i$ and $V(\hat{\beta}) = V((\hat{X}\hat{X})^{-1} \hat{X}'\hat{Y})$ since

$\hat{\beta} = (\hat{X}\hat{X})^{-1} \hat{X}'\hat{Y}$ as discussed in the Introduction. From here,

$$V((\hat{X}\hat{X})^{-1} \hat{X}'\hat{Y}) = ((\hat{X}\hat{X})^{-1} \hat{X}') V(\hat{Y}) ((\hat{X}\hat{X})^{-1} \hat{X}')' \text{ where } V(\hat{Y}) = V(\hat{X}\hat{\beta} + \hat{\epsilon}) = \sigma^2 I$$

since $\hat{\epsilon}$ is a vector of independent and identically distributed random errors with constant variance σ^2 . Thus,

$$\begin{aligned} V(\hat{\beta}) &= V((\hat{X}\hat{X})^{-1} \hat{X}'\hat{Y}) \\ &= ((\hat{X}\hat{X})^{-1} \hat{X}') V(\hat{Y}) (\hat{X}(\hat{X}\hat{X})^{-1}) \\ &= ((\hat{X}\hat{X})^{-1} \hat{X}') \sigma^2 I (\hat{X}(\hat{X}\hat{X})^{-1}) \\ &= \sigma^2 ((\hat{X}\hat{X})^{-1} \hat{X}') (\hat{X}(\hat{X}\hat{X})^{-1}) \\ &= \sigma^2 (\hat{X}\hat{X})^{-1} \end{aligned}$$

and $V(\hat{\beta}) = \sigma^2 (X'X)^{-1} = \begin{pmatrix} V(\hat{\beta}_0) & \dots & \text{cov}(\hat{\beta}_0, \hat{\beta}_k) \\ \vdots & \ddots & \vdots \\ \text{cov}(\hat{\beta}_k, \hat{\beta}_0) & \dots & V(\hat{\beta}_k) \end{pmatrix}$. Consequently,

$$\hat{V}(\hat{\beta}) = \hat{\sigma}^2 (X'X)^{-1} = \begin{pmatrix} \hat{V}(\hat{\beta}_0) & \dots & \widehat{\text{cov}}(\hat{\beta}_0, \hat{\beta}_k) \\ \vdots & \ddots & \vdots \\ \widehat{\text{cov}}(\hat{\beta}_k, \hat{\beta}_0) & \dots & \hat{V}(\hat{\beta}_k) \end{pmatrix} \quad (2.2)$$

where $\hat{\sigma}^2$, an estimate of σ^2 , is the mean square error (MSE) of the regression. Therefore, $\widehat{\sigma(p(x_i))}^2 = \hat{\sigma}^2 (x_i' (X'X)^{-1} x_i)$. These formulas apply to any degree random polynomial.

Theorem 2.1: A $100(1-\alpha)\%$ confidence interval for x_m , the abscissa of the extrema of a random polynomial, is the set of x_i such that

$$z_{\alpha/2} < \frac{(p'(x_i))}{\widehat{\sigma(p'(x_i))}} < z_{1-\alpha/2}, \text{ where } z \text{ is the appropriate percentile of the}$$

standard normal distribution, $p'(x_i) = \hat{\beta}_1 + 2\hat{\beta}_2 x_i + 3\hat{\beta}_3 x_i^2 + \dots + k\hat{\beta}_k x_i^{k-1}$,

$$\widehat{\sigma(p'(x_i))} = \sqrt{(x_i^\Delta)' \hat{V}(\hat{\beta}^\Delta) x_i^\Delta}, \quad x_i^\Delta = (1, 2x_i, 3x_i^2, \dots, kx_i^{k-1})' \text{ and}$$

$$\hat{V}(\hat{\beta}^\Delta) = \hat{V}(\hat{\beta})_{2:k+1, 2:k+1} = \begin{pmatrix} \hat{V}(\hat{\beta}_1) & \dots & \widehat{\text{cov}}(\hat{\beta}_1, \hat{\beta}_k) \\ \vdots & \ddots & \vdots \\ \widehat{\text{cov}}(\hat{\beta}_k, \hat{\beta}_1) & \dots & \hat{V}(\hat{\beta}_k) \end{pmatrix} \text{ is the } 2:k+1, 2:k+1 \text{ portion}$$

of $\hat{V}(\hat{\beta})$ as described in Equation (2.2).

Proof: In this case, we are interested in calculating confidence intervals for the abscissa of the extrema, or the root of the derivative, of the

random polynomial. Since the derivative of a random polynomial is another random polynomial, we simply replace $p(x)$ in the Equation (2.1) above (from Fairley) with $p'(x)$. Then, the region on the x -axis as described above defines a confidence interval for the abscissa of the extrema of the random polynomial. Equation (2.1) is equivalent to

$$t_{\alpha/2, n-(k+1)} < \frac{(p'(x_i))}{\sigma(p'(x_i))} < t_{1-\alpha/2, n-(k+1)} \text{ where } t_{n-(k+1)} \text{ represents the } t\text{-distribution}$$

with $n-(k+1)$ degrees of freedom (since the F -distribution with 1 numerator degree of freedom corresponds to the square of the t -distribution as mentioned earlier). Since the t -distribution converges to

the normal distribution, this is equivalent to $z_{\alpha/2} < \frac{(p'(x_i))}{\sigma(p'(x_i))} < z_{1-\alpha/2}$, where

$\widehat{\sigma(p'(x_i))}^2 = ((\hat{x}_i^\Delta)'(\hat{V}(\hat{\beta}^\Delta))\hat{x}_i^\Delta)$, as outlined above. Therefore, a confidence

interval for x_m is the set $\{x_i \mid z_{\alpha/2} < \frac{(p'(x_i))}{\sigma(p'(x_i))} < z_{1-\alpha/2}\}$. $\square$

Theorem 2.2: A $100(1-\alpha)\%$ confidence interval for y_m , the ordinate of the

extrema of a random polynomial, is given by $\hat{y}(\hat{x}_m) \pm z_{\alpha/2} \hat{\sigma} \sqrt{\hat{x}_m' (\hat{X}'\hat{X})^{-1} \hat{x}_m}$,

where z is the appropriate percentile of the standard normal distribution,

$\hat{x}_m = (1, \hat{x}_m, \hat{x}_m^2, \dots, \hat{x}_m^k)'$ and matrix $\hat{X}$ is as described in and around

Equation (1.1).

Proof: The confidence interval for the ordinate of the extrema, y_m , is calculated using ordinary methods from regression analysis. As in Myers (1990), assuming normal errors, $100(1-\alpha)\%$ confidence bounds for

$E(y|x=x_0)$ are given by $\hat{y}(x_0) \pm t_{\alpha/2, n-p} s \sqrt{x_0' (X'X)^{-1} x_0}$ (p.112). $\square$

Delta Method

The delta method is based on using Taylor series expansions to approximate variances and covariances of functions of parameter estimators. Many references exist for the delta method, for example, details can be found in Meeker and Escobar (1998). The following describes the details regarding the use of the delta method to find confidence intervals for the extrema of a random quadratic polynomial.

To calculate the true abscissa, x_m , of the extrema of, for example, a quadratic equation, $m(x) = \beta_0 + \beta_1 x + \beta_2 x^2$, set the derivative equal to zero

(i.e., $m'(x) = \beta_1 + 2\beta_2 x = 0$). Solving this equation for x gives $x_m = \frac{-\beta_1}{2\beta_2}$ which

we define to be $g_1(\underline{\beta})$, where $\underline{\beta} = (\beta_0, \beta_1, \beta_2)'$ are the true values of the parameters. Now that the true abscissa of the extrema has been calculated, the true ordinate (y value) of the extrema can be computed. Recall that, $y_m = \beta_0 + \beta_1 x_m + \beta_2 x_m^2$ and we are interested in calculating y_m

when $x_m = \frac{-\beta_1}{2\beta_2}$. Upon substituting $\frac{-\beta_1}{2\beta_2}$ in place of x_m in y_m , we get

$$y_m = \beta_0 + \beta_1 \left(\frac{-\beta_1}{2\beta_2} \right) + \beta_2 \left(\frac{-\beta_1}{2\beta_2} \right)^2 = \beta_0 - \frac{\beta_1^2}{4\beta_2} =: g_2(\beta). \text{ Define } \mathbf{g}(\beta) = (g_1(\beta), g_2(\beta)).$$

The estimated random quadratic equation is $p(x) = \hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2$ whose derivative is $p'(x) = \hat{y}' = \hat{\beta}_1 + 2\hat{\beta}_2 x$. Similar to what was done above for the true abscissa, to calculate the estimated abscissa ($\hat{x}_m$) of the extrema, set the derivative of the estimated random polynomial equal to zero (i.e., $p'(x) = \hat{y}' = \hat{\beta}_1 + 2\hat{\beta}_2 x = 0$). Solving for x gives $\hat{x}_m = \frac{-\hat{\beta}_1}{2\hat{\beta}_2}$. Then,

$$g_1(\hat{\beta}) := \hat{x}_m = \frac{-\hat{\beta}_1}{2\hat{\beta}_2} \text{ where } \hat{\beta} = (\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2)' \text{ are estimates of the parameters.}$$

Now, compute the estimated ordinate (y value) of the extrema. Recall

that, $\hat{y}_m = \hat{\beta}_0 + \hat{\beta}_1 \hat{x}_m + \hat{\beta}_2 \hat{x}_m^2$ and calculate $\hat{y}_m$ when $\hat{x}_m = \frac{-\hat{\beta}_1}{2\hat{\beta}_2}$. Substitute

$\frac{-\hat{\beta}_1}{2\hat{\beta}_2}$ in place of $\hat{x}_m$ in $\hat{y}_m$, to get

$$\hat{y}_m = \hat{\beta}_0 + \hat{\beta}_1 \left(\frac{-\hat{\beta}_1}{2\hat{\beta}_2} \right) + \hat{\beta}_2 \left(\frac{-\hat{\beta}_1}{2\hat{\beta}_2} \right)^2 = \hat{\beta}_0 - \frac{\hat{\beta}_1^2}{4\hat{\beta}_2} =: g_2(\hat{\beta}). \text{ Similarly, define}$$

$$\mathbf{g}(\hat{\beta}) = (g_1(\hat{\beta}), g_2(\hat{\beta})).$$

From Meeker and Escobar (1998),

$$V[\mathbf{g}(\hat{\beta})] = \begin{pmatrix} V(g_1(\hat{\beta})) & \text{cov}(g_1(\hat{\beta}), g_2(\hat{\beta})) \\ \text{cov}(g_2(\hat{\beta}), g_1(\hat{\beta})) & V(g_2(\hat{\beta})) \end{pmatrix} = \begin{pmatrix} V(X_m) & \text{cov}(X_m, Y_m) \\ \text{cov}(Y_m, X_m) & V(Y_m) \end{pmatrix} \quad (2.3)$$

and is calculated as $V[\mathbf{g}(\hat{\beta})] \approx \left[\frac{\partial \mathbf{g}(\beta)}{\partial \beta} \right]'_{\beta=\hat{\beta}} V(\hat{\beta}) \left[\frac{\partial \mathbf{g}(\beta)}{\partial \beta} \right]_{\beta=\hat{\beta}}$,

$$\text{where } \left[\frac{\partial \mathbf{g}(\beta)}{\partial \beta} \right]_{\beta=\hat{\beta}} = \begin{pmatrix} \frac{\partial g_1(\beta)}{\partial \beta_0} & \frac{\partial g_2(\beta)}{\partial \beta_0} \\ \frac{\partial g_1(\beta)}{\partial \beta_1} & \frac{\partial g_2(\beta)}{\partial \beta_1} \\ \frac{\partial g_1(\beta)}{\partial \beta_2} & \frac{\partial g_2(\beta)}{\partial \beta_2} \end{pmatrix}_{\beta=\hat{\beta}} = \begin{pmatrix} 0 & 1 \\ \frac{-1}{2\hat{\beta}_2} & \frac{-\hat{\beta}_1}{2\hat{\beta}_2} \\ \frac{\hat{\beta}_1}{2\hat{\beta}_2^2} & \frac{\hat{\beta}_1^2}{4\hat{\beta}_2^3} \end{pmatrix}. \quad V(\hat{\beta}) \text{ is estimated}$$

from the Fisher Information matrix, I_β . Since data is available, one can

compute the observed information matrix $\hat{I}_\beta := \left(-\frac{\partial^2 \ell(\beta)}{\partial \beta \partial \beta'} \right)^{-1}$, where $\ell(\beta)$ is

the log likelihood for the specified model. Then, an estimate of the

variance of $\hat{\beta}$ is $\hat{V}(\hat{\beta}) = (\hat{I}_\beta)^{-1}$. In our case, $\hat{V}(\hat{\beta})$ is calculated as in the

Fairley method previously discussed.

From here, $\widehat{V}(X_m) = \hat{V}(g_1(\hat{\beta}))$ can be used to calculate the confidence interval for the abscissa, assuming X_m to be normally distributed, as

$\hat{x}_m \pm z_{\alpha/2} \sqrt{\widehat{V}(X_m)}$ and $\widehat{V}(Y_m) = \hat{V}(g_2(\hat{\beta}))$ can be used to compute

the confidence interval for the ordinate, as $\hat{y}_m \pm z_{\alpha/2} \sqrt{\widehat{V}(Y_m)}$.

One should note when using the delta method, that the [2,1] element of $V[\hat{g}(\hat{\beta})]$ is the covariance between the random variables associated with abscissa and ordinate of the extrema of the random polynomial. The covariance is a measure of the linear dependence between the abscissa and ordinate. The larger the absolute value of the covariance, the greater the linear dependence between the abscissa and ordinate. Once the covariance is calculated, the correlation can be computed as $\rho = \frac{\text{cov}(X_m, Y_m)}{\sigma_{X_m} \sigma_{Y_m}}$. The correlation gives a standardized value for the linear dependence between the estimated abscissa and ordinate values (Wackerly, et al., 2002, p.250).

Although the details are omitted here, the same steps were followed for a random cubic polynomial. However, the complexity of the partial derivatives generally makes the use of this procedure for fourth order and higher degree polynomials too complicated to calculate and program.

Theorem 2.3: A $100(1-\alpha)\%$ confidence interval for x_m is $\hat{x}_m \pm z_{\alpha/2} \sqrt{\widehat{V}(X_m)}$, where z is the appropriate percentile of the standard normal distribution. Here, $\widehat{V}(X_m) = \hat{V}(g_1(\hat{\beta}))$ is the estimated value of the variance of the random variable X_m .

Proof: For a normal random variable W , the form of a $100(1-\alpha)\%$ confidence interval is $\hat{w} \pm z_{\alpha/2} \sqrt{\widehat{V(W)}}$. Bharucha-Reid and Sambandham (1986) show that the real roots of random polynomials with normally distributed coefficients are also approximately normally distributed. Since X_m is a real root of a random polynomial, and the coefficients, $\hat{\beta}_{\sim}$, are normally distributed, X_m is a normal random variable. Therefore, a $100(1-\alpha)\%$ confidence interval for x_m is $\hat{x}_m \pm z_{\alpha/2} \sqrt{\widehat{V(X_m)}}$, where $\widehat{V(X_m)} = \hat{V}(g_1(\hat{\beta}_{\sim}))$ and is calculated as the $[1,1]$ element of $\hat{V}[g(\hat{\beta}_{\sim})]$ in Equation (2.3). $\square$

Theorem 2.4: A $100(1-\alpha)\%$ confidence interval for y_m is $\hat{y}_m \pm z_{\alpha/2} \sqrt{\widehat{V(Y_m)}}$, where z is the appropriate percentile of the standard normal distribution. Here, $\widehat{V(Y_m)} = \hat{V}(g_2(\hat{\beta}_{\sim}))$ is the estimated value of the variance of the random variable Y_m .

Proof: For a normal random variable W , the form of a $100(1-\alpha)\%$ confidence interval is $\hat{w} \pm z_{\alpha/2} \sqrt{\widehat{V(W)}}$. From ordinary regression theory, Y_m is a normal random variable (Myers, 1990). Therefore, a $100(1-\alpha)\%$ confidence interval for y_m is $\hat{y}_m \pm z_{\alpha/2} \sqrt{\widehat{V(Y_m)}}$, where $\widehat{V(Y_m)} = \hat{V}(g_2(\hat{\beta}_{\sim}))$ and is calculated as the $[2,2]$ element of $\hat{V}[g(\hat{\beta}_{\sim})]$ in Equation (2.3). $\square$

Bootstrapping

As Meeker and Escobar (1998, p. 205) describe, "The idea of bootstrap sampling is to simulate the repeated sampling process and use the information from the distribution of appropriate statistics in the bootstrap samples to compute the needed confidence interval (or intervals), reducing the reliance on large-sample approximations."

The values of interest are the abscissa and ordinate values for the minimum or maximum of a random polynomial. Therefore, we want to take an observed data set $\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_n, y_n)\}$, and draw B samples (with replacement), each of size n . For the i^{th} random sample, calculate the extrema $(\hat{x}_{m_i}, \hat{y}_{m_i})$ of the random polynomial that was generated from the regression fit of the sampled data set. To accomplish this, first estimate the coefficients of the random polynomial via the method of least squares. Then, estimate the abscissa of the extrema by setting the derivative of the estimated random polynomial, $p(x)$, equal to zero and solve for x (i.e., find $\hat{x}_{m_i}$ such that $p'(\hat{x}_{m_i}) = 0$). Once the abscissa is calculated, the ordinate follows as $\hat{y}_{m_i} = p(\hat{x}_{m_i})$. After B samples have been generated, their associated extrema calculated and saved, the data is sorted in ascending order ($\hat{x}_{m_i}$ and $\hat{y}_{m_i}$ independently). The abscissa confidence intervals are computed by taking the upper and lower $\alpha/2$

quantiles of the sorted $\{\hat{x}_{m_1}, \hat{x}_{m_2}, \dots, \hat{x}_{m_B}\}$ bootstrapped estimates. Similarly, the ordinate confidence intervals are obtained by taking the appropriate quantiles of the sorted $\hat{y}_{m_i}$ bootstrapped estimates.

Figure 2.1. Illustration of nonparametric bootstrap

Chapter 3

Implementation

Simulations

For simplicity in comparing the three methods, polynomials were chosen which had a single unique maximum. The confidence intervals for the one maximum were calculated. The same can be done for the minimums of random polynomials or in situations with multiple extrema (assuming it is known in what region the extrema of interest is located).

Programs were written in S-Plus® Version 7.0 to implement each algorithm (the Fairley method, the delta method, and bootstrapping). Additionally, simulation and testing functions were written. For each method, each degree polynomial, and each type of random noise (normal, exponential, and t -distribution), the following steps were carried out:

- Start with a known polynomial of specified degree,
 $m(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \dots + \beta_k x^k$, with a known real-valued maximum
- For a domain of interest (chosen to surround the maximum) specify a set of $n=100$ x values $\{x_1, x_2, x_3, \dots, x_n\}$ at which the polynomial is evaluated
- Evaluate the polynomial at each of the specified x values
- Repeat these steps for N simulations:
 - Add random noise (normal, exponential or t -distribution) to simulate response data and get $\{y_1, y_2, y_3, \dots, y_n\}$

- Using the simulated data $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, fit an algebraic polynomial of specified degree (i.e., we are assuming the degree of the polynomial is known),

$$p(x) = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + \dots + \hat{\beta}_k x^k$$
using the least squares estimation regression function from S-Plus®
- Obtain regression model parameters from S-Plus® such as the standard deviation of the residuals ($\hat{\sigma}$), and covariance matrix for the parameter estimates, $\hat{V}(\hat{\beta})$
- Calculate the derivative of the random polynomial,

$$p'(x) = \hat{\beta}_1 + 2\hat{\beta}_2 x + \dots + k\hat{\beta}_k x^{k-1}$$
- Estimate the root of the derivative $\hat{x}_m$ of the random polynomial using numerical techniques from S-Plus®
- Calculate the associated y value of the maximum,

$$\hat{y}_m = p(\hat{x}_m) = \hat{\beta}_0 + \hat{\beta}_1 \hat{x}_m + \hat{\beta}_2 \hat{x}_m^2 + \dots + \hat{\beta}_p \hat{x}_m^p$$
- Calculate the upper and lower 95% confidence limits for (x_m, y_m) using the specified method
- Determine the accuracy of the method by counting the number of times (out of N) that the true maximum, (x_m, y_m) , lies within the calculated limits

The second degree polynomial used for simulation was

$m(x) = y = -x^2 + 4x + 3$ with $m'(x) = y' = -2x + 4$. Thus, the only root of the derivative is at $x = 2$. This polynomial has a maximum at $(2, 7)$. The domain used for simulation was $[0, 5]$. As noted earlier, for each degree polynomial, three different types of random noise were added. For the

normal distribution noise, a mean of zero and a standard deviation of 1 was used. The standard deviation of the noise for each known polynomial was chosen to be between 10 and 20 percent of the range of the y values, which is large enough to give good variation, but small enough to correctly estimate the random polynomial.

For the exponential distribution, a scale of 1 was used to determine the effect of a skew distribution with a standard deviation of 1. The scale value of 1 was then subtracted to maintain a zero mean (i.e., if $\exp_i(1)$ is the i^{th} exponential random value with a scale value of 1, the i^{th} noise term was $noise_i = \exp_i(1) - 1$ so that $E(noise_i) = 0$ and $Var(noise_i) = 1$), as in the normal distribution.

For the t -distribution, we used $\nu = 3$ degrees of freedom in order to examine the effect of a long tailed distribution. However, in order to maintain consistency, this random quantity was divided by the square root of 3 (i.e., if $t_{3,i}$ is a random value from the t -distribution with 3 degrees of freedom, then the i^{th} added noise term was $noise_i = t_{3,i} / \sqrt{3}$ so that $E(noise_i) = 0$ and $Var(noise_i) = 1$, as it was for the other distributions).

Figure 3.1, below, shows an example of a simulation for a second degree polynomial with noise from the normal distribution. The small solid circles are the n observations, the solid line is the known polynomial, $m(x)$, the dotted line is the estimated random polynomial,

$p(x)$, and the box represents the confidence intervals for the abscissa and ordinate as computed using the delta method.

Figure 3.1. One example of a second degree random polynomial with normally distributed noise and associated confidence intervals calculated via the delta method

Figure 3.2 is a close up of Figure 3.1 and shows the true and estimated polynomials, the true maximum, (x_m, y_m) , the estimated maximum, $(\hat{x}_m, \hat{y}_m)$, as well as the confidence region obtained using the delta method.

Figure 3.2. Close up of example of a second degree random polynomial with normally distributed noise and associated confidence intervals

For the third degree polynomial we used $m(x) = y = x^3 - 7.5x^2 + 12x$ with $m(x) = y' = (3x - 3)(x - 4)$. The roots of the derivative are at $x = 1, 4$. This polynomial has a maximum at $(1, 5.5)$. The domain used for simulation was $[-0.5, 5.5]$. Again, three types of random noise were added. For the normal distribution noise, a mean of zero and a standard deviation of 2.5 was used. For the exponential distribution, a scale of 2.5 was used, then subtracted to maintain a zero mean. The t -distribution used 3 degrees of

freedom and was then divided by $\sqrt{3}/2.5$ so that the standard deviation was 2.5 as it was for the other two distributions. See Figure 3.3, below, for an example of the third degree polynomial simulation.

Figure 3.3. One example of a third degree random polynomial with exponentially distributed noise and associated confidence intervals calculated via bootstrapping method

The fourth degree polynomial that was used was

$$m(x) = y = -\frac{1}{60}x^4 + \frac{1}{9}x^3 - \frac{16}{15}x^2 + \frac{78}{15}x \text{ with}$$

$m'(x) = y' = -\frac{1}{15}(x-3)(x-(1-5i))(x-(1+5i))$. The roots of the derivative are at $x=3, 1+5i, 1-5i$. This polynomial has a maximum at $(3, 7.65)$. The domain used for simulation was $[0,5]$. Again, three types of random noise were added. For the normal distribution noise, a mean of zero and a standard deviation of 1.5 was used. For the exponential distribution, a scale of 1.5 was used, then subtracted off to maintain a zero mean. The t -distribution used 3 degrees of freedom and was then divided by $\sqrt{3}/1.5$ so that the standard deviation was 1.5 as it was for the other two distributions. See Figure 3.4, below, for an example of a fourth degree polynomial simulation.

Figure 3.4. One example of a fourth degree random polynomial with t -distribution noise and associated confidence intervals calculated via the Fairley method

As stated in the introduction, the abscissa and ordinate of the extrema of the random polynomial are assumed uncorrelated. Figure 3.5 below, shows an example of one set of simulations where the estimated abscissa, $\hat{x}_m$, is plotted against the estimated ordinate, $\hat{y}_m$ for all 2500 simulations of the third order polynomial with normal distribution noise. The sample correlation calculated between the estimated abscissa and ordinate for this case was $\hat{\rho} = 0.065$. To test for absence of correlation,

which is the null hypothesis, we used $t = \frac{\hat{\rho}\sqrt{n-2}}{\sqrt{1-\hat{\rho}^2}} = 0.646$ (Wackerly, Mendenhall, and Scheaffer, 2002). This corresponds to a P -value of 0.74, which implies that we retain the null hypothesis of uncorrelated abscissa and ordinate values.

Figure 3.5. Plot of $(\hat{x}_m, \hat{y}_m)$ for all simulations of the third degree polynomial with normally distributed noise

Results - Confidence Interval Accuracy

The following tables give the accuracy of each method (Fairley, delta, and bootstrap) for each of the three choices of noise (normal, exponential, and t -distribution). Each table specifies the results for the abscissa or ordinate and is based on the degree of the original simulated polynomial. We would like to see the percentage of times that the true value lies within the calculated confidence interval to be approximately $100(1-\alpha)$ (i.e., $100(\frac{\text{\# times true value lies within CI}}{N}) \approx 100(1-\alpha)$), where N is the number of simulations). In our case, $N=2500$ and 95% ($\alpha=.05$) confidence intervals were calculated. Therefore, if the percentage is close to 95%, which is the nominal level of the confidence interval, then the method is deemed successful. The numbers in bold indicate which method was the closest to the nominal level of confidence for each set of simulations run. If the values for two different methods (for a particular degree polynomial and noise) are less than about 1.2% apart, there is no statistically significant difference between the accuracy of the methods. The number 1.2% is approximately two standard errors of the difference of the proportions. With very few exceptions, the three methods do not have statistically significant differences in the accuracy results.

Table 3.1. Percentages of time that the true **abscissa** lies within the calculated confidence limits for a **second** degree random polynomial

	Normal	Exponential	<i>t</i>
Fairley Method	94.20	94.28	93.96
Delta Method	94.88	94.92	94.60
Bootstrapping	94.20	93.96	94.12

Table 3.2. Percentages of time that the true **ordinate** lies within the calculated confidence limits for a **second** degree random polynomial

	Normal	Exponential	<i>t</i>
Fairley Method	94.20	94.80	94.80
Delta Method	93.80	94.88	94.96
Bootstrapping	94.72	93.76	93.20

For the second degree polynomial abscissa, it seems that the delta method gives the best results (closest to the nominal level) for all choices of noise. However, for the second degree polynomial ordinate, bootstrapping was the best for the normal distribution noise; the delta method was just barely better than the Fairley method for both the exponential and *t*-distribution noise.

Table 3.3. Percentages of time that the true **abscissa** lies within the calculated confidence limits for a **third** degree random polynomial

	Normal	Exponential	<i>t</i>
Fairley Method	94.80	94.64	94.60
Delta Method	94.48	94.20	94.08
Bootstrapping	94.00	92.76	93.24

Table 3.4. Percentages of time that the true **ordinate** lies within the calculated confidence limits for a **third** degree random polynomial

	Normal	Exponential	<i>t</i>
Fairley Method	95.16	93.68	95.20
Delta Method	94.92	94.72	94.84
Bootstrapping	94.44	93.00	93.12

For the third order polynomial, the Fairley method was the closest to nominal for the abscissa and ordinate for almost all choices of noise. The only exception was for the ordinate when exponential noise was added, where the delta method provided closer to nominal results. Bootstrapping was the farthest from the nominal confidence levels for all third degree simulations run.

Table 3.5. Percentages of time that the true **abscissa** lies within the calculated confidence limits for a **fourth** degree random polynomial

	Normal	Exponential	<i>t</i>
Fairley Method	94.92	94.72	95.16
Bootstrapping	94.20	94.32	94.72

Table 3.6. Percentages of time that the true **ordinate** lies within the calculated confidence limits for a **fourth** degree random polynomial

	Normal	Exponential	<i>t</i>
Fairley Method	94.12	94.40	95.08
Bootstrapping	93.48	95.28	93.52

For the fourth degree polynomial, only the Fairley method and bootstrapping were used to calculate the confidence intervals. As mentioned earlier, the complexity of the partial derivatives required for the delta method made it too complicated to use. Again, the results are close, but mixed. The Fairley method percentages were closest to the nominal level for both the abscissa and ordinate when normal and *t*-distributed noise were used. However, when the noise came from the exponential distribution, the Fairley method was closer to nominal for the abscissa whereas bootstrapping was closer to nominal for the ordinate value.

Results - Confidence Interval Lengths

The following tables give the average length of the confidence intervals calculated for each method (Fairley, delta, and bootstrap) for each of the three variations of noise (normal, exponential, and t -distribution). Each table specifies the results for the abscissa or ordinate and is based on the degree of the original polynomial simulated. We would like to see the length of the confidence interval to be as small as possible, assuming the method is sufficiently accurate. Therefore, if the method is accurate and the width is small, then the method is optimal. The numbers in bold indicate the smallest average confidence interval length for each set of simulations run.

Table 3.7. Average lengths of the **abscissa** confidence intervals for a **second** degree random polynomial

	Normal	Exponential	t
Fairley Method	.170	.169	.165
Delta Method	.147	.183	.148
Bootstrapping	.169	.170	.162

Table 3.8. Average lengths of the **ordinate** confidence intervals for a **second** degree random polynomial

	Normal	Exponential	t
Fairley Method	.565	.563	.546
Delta Method	.529	.644	.499
Bootstrapping	.562	.555	.540

Looking at Tables 3.7 and 3.8 there is no clear method that

provides the smallest confidence intervals. The delta method provided the smallest average confidence interval length when normal and t -distribution noise were used. When exponentially distributed noise was used for the simulations, the Fairley method offers the smallest average confidence interval length for the abscissa, whereas bootstrapping gave the smallest average confidence interval length for the ordinate value.

Table 3.9. Average lengths of the **abscissa** confidence intervals for a **third** degree random polynomial

	Normal	Exponential	t
Fairley Method	.139	.139	.135
Delta Method	.135	.138	.133
Bootstrapping	.139	.137	.134

Table 3.10. Average lengths of the **ordinate** confidence intervals for a **third** degree random polynomial

	Normal	Exponential	t
Fairley Method	1.745	1.732	1.679
Delta Method	1.600	1.732	1.675
Bootstrapping	1.731	1.699	1.650

For the third degree polynomial, the average confidence interval length results were similar to those for the second degree. Once again, the delta method had the smallest average confidence interval length for the normal distribution noise. However, bootstrapping provided the smallest average confidence interval length for both the abscissa and ordinate when exponentially distributed noise was used. The results were mixed for the t -distribution noise.

Table 3.11. Average lengths of the **abscissa** confidence intervals for a **fourth** degree random polynomial

	Normal	Exponential	<i>t</i>
Fairley Method	.656	.653	.510
Bootstrapping	.653	.642	.612

Table 3.12. Average lengths of the **ordinate** confidence intervals for a **fourth** degree random polynomial

	Normal	Exponential	<i>t</i>
Fairley Method	1.037	1.028	.792
Bootstrapping	1.027	1.060	.979

Just as with the second and third degree polynomials, there is no method that clearly gives the smallest confidence interval lengths for fourth degree polynomials. Bootstrapping has the smallest average confidence interval length (in both abscissa and ordinate) for the normally distributed noise. For the *t*-distributed noise, the Fairley method provided the smallest average confidence intervals. However, for the exponential distribution, the results are mixed.

Results - Overall

Looking at the accuracy data (percentages of times the true abscissa or ordinate lies within the calculated confidence intervals) combined with the length of the confidence intervals, we can determine if there is an optimal method for computing these confidence intervals.

For the second degree polynomials simulated, the delta method for the abscissa had accuracy results that were the closest to the nominal level and the smallest average confidence interval lengths for the normal and t -distribution noise. Although the delta method had the smallest average confidence interval length for the ordinate for the normally distributed noise, the bootstrapping method had the best accuracy. For the second degree exponentially distributed noise simulations, the delta method had the best accuracy, but not the smallest average confidence interval lengths for either the abscissa or ordinate.

The third degree polynomial simulations did not indicate an optimal method. The Fairley method was the closest to nominal for all simulations except the exponential noise ordinate where the delta method was closer to nominal. However, the delta method and bootstrapping had the smallest average confidence interval lengths for both the abscissa and ordinate with all choices of noise.

The results for the fourth degree polynomial simulations, again, do not indicate a single optimal method. For both the abscissa and ordinate, the Fairley method provided the most accurate confidence intervals for normally distributed noise, but bootstrapping had the smallest average confidence interval lengths. For the cases where t -distribution noise was added, the Fairley method had the closest to nominal percentages, and the smallest average confidence interval

lengths for both the abscissa and ordinate values.

When using the Fairley method for fourth degree polynomial simulations with t -distributed noise, caution need be taken. There are cases using a long tailed distribution where the Fairley method creates an indefinite length confidence interval. If the estimated variance of $p'(x_i)$, $(\widehat{\sigma(p'(x_i))})^2$ increases as x_i increases, then the ratio $\frac{p'(x_i)}{\widehat{\sigma(p'(x_i))}}$ may remain in the interval $(z_{\alpha/2}, z_{1-\alpha/2})$ indefinitely. Similarly, if $\widehat{\sigma(p'(x_i))}$ increases as x_i decreases, this can also cause the ratio to remain in the interval as described above. Either of these cases can cause the Fairley method to create an infinite length confidence interval.

Over the 2500 simulations of the fourth order polynomial with t -distribution noise, there were four instances of the infinite confidence intervals using the Fairley method. Each of these four instances was caused by an outlier whose distance from the estimated curve to the outlier, in terms of number of standard deviations away, was 36.3, 45.0, 67.2, and 141.1. When the outliers were removed, the confidence interval calculations were successful and appropriate. One should note that these outliers are reasonable values to occur in $n*N$ simulations with a t -distribution with 3 degrees of freedom.

Chapter 4

Empirical Application

Data from NASA Ames Research Center was gathered over a region of the Pacific Ocean to investigate the possibility of a runaway greenhouse effect. A runaway greenhouse effect occurs when the amount of solar radiation absorbed exceeds the amount reflected or released (i.e., when planetary heat loss begins to decrease as surface temperature rises). The data included measurements of sea surface temperature in degrees Kelvin and clear sky upward long-wave flux (also known as outgoing flux) measured in watts per meter squared at numerous latitudes and longitudes over the Pacific Ocean. Outgoing flux is a measure of the rate of flow of heat back up out of the atmosphere. Of particular interest was to model the relationship between the sea surface temperature and outgoing flux, determine its maximum, and calculate confidence intervals for this maximum.

The data that was used for this study included weekly observations over a one-year time span from March 1, 2000 through February 28, 2001 (53 weeks). For each week, ten different latitudes and thirty-six different longitudes at which the measurements were observed were included. This gives 19,080 observations total.

Several regressions were run to determine which order polynomial best represented the data. It was determined that an eighth degree polynomial best fit the data. The order of the polynomial to best fit the data was determined by attempting various degree regressions and choosing the one with the smallest mean square error (MSE). Once the random polynomial was estimated, the estimated maximum was computed. The estimated maximum for this set of data was (299.975, 291.6639). This implies that there is evidence of a runaway greenhouse effect occurring in specific regions of the Pacific Ocean, when planetary heat loss begins to decrease, as surface temperature continues to rise, (i.e., as outgoing flux reaches a maximum then begins to decrease while sea surface temperature continues to increase). After the maximum was estimated, confidence intervals were calculated using the Fairley method and bootstrapping. Figure 4.1 shows the data, estimated polynomial, and estimated maximum.

Figure 4.1. Graph of Sea Surface Temperature vs. Outgoing Flux with estimated random polynomial and associated maximum

The confidence intervals calculated via the Fairley method and bootstrapping are given in Table 4.1. It is interesting to note that the Fairley method has a slightly wider confidence interval for the abscissa than the bootstrapping method. Conversely, the bootstrapping method confidence interval for the ordinate is slightly wider than the corresponding confidence interval as calculated by the Fairley method. It should be noted that both methods of calculating confidence intervals contain the estimated maximum as would be expected.

Table 4.1. Calculated abscissa and ordinate confidence intervals for the maximum of sea surface temperature vs. outgoing flux

	Abscissa Confidence Intervals	Ordinate Confidence Intervals
Fairley	(299.9093, 300.0423)	(291.4298, 291.8980)
Bootstrap	(299.9144, 300.0348)	(291.3218, 291.9251)

Chapter 5

Conclusions

The topic of estimating the abscissa and ordinate of an extrema, and their associated confidence intervals, for random polynomials was studied. Three methods of interest for calculating the confidence intervals were tested; these included the Fairley method, delta method, and bootstrapping.

The Fairley method is based on a ratio discussed by Fieller. Fairley (1968) indicated that the region on the x axis where the ratio of the estimated polynomial and the variance of the polynomial is less than an F -distribution with $n-(k+1)$ degrees of freedom defines a confidence region for the root of the random polynomial. This method was modified to find the confidence interval for the root of the derivative of a random polynomial, therefore giving a confidence interval for the abscissa of the extrema of the random polynomial. A standard confidence interval technique from regression analysis was used to compute the confidence bounds for the ordinate of the extrema of the random polynomial.

The delta method, using Taylor series approximations, was used for second and third degree polynomials only. The complexity of the partial derivatives that are required for the use of this method for fourth and higher degree polynomials are too complicated. This method proved

to be the fastest computationally, of the three methods that were investigated. For example, to run one hundred simulations for the delta method only took a matter of seconds, whereas the Fairley method may take an hour and bootstrapping could take two hours. Both the Fairley method and bootstrapping have internal loops that require a significant amount of processing time but the delta method has only simple calculations at each simulation, after the initial partial derivatives are computed.

Nonparametric bootstrapping, based on repeated sampling with replacement, although by far the easiest to implement, only needing to compute the maximums and take quantiles, was the most time consuming (computer intensive) process to run because for each single simulation, there were $B=2000$ bootstrap iterations. Literature recommends using between 2000 and 10,000 bootstrap iterations to achieve reasonably accurate results (Meeker and Escobar, 1998, p. 206).

To fully investigate each of the methods, known polynomials of degree 2, 3, and 4 were used as bases for simulations using random noise from the normal, exponential, and t distributions. It should be noted that in all of the regressions that were run for the simulations, we are assuming that the degree of the polynomial is known. However, this is rarely the case. One should study information about model selection to determine the correct degree polynomial to be fit for a given set of

input and response data. See Myers (1990) for more information on model selection.

In general, all three methods seemed to be roughly equally accurate and provided approximately the nominal level of confidences desired. Of the three methods investigated, the Fairley method is the best overall choice for computing confidence intervals for the abscissa and ordinate of the extrema of random polynomials in the absence of outliers creating infinite confidence intervals. In any case, such outliers would likely be detected and removed prior to regression analysis, thus obviating this difficulty. This method is the most versatile (applicable for all degree polynomials), the accuracy provides approximately the nominal level of confidence, the average confidence interval lengths were similar to the other two methods, and the computational times, although longer than the delta method, were far better than the bootstrapping method.

Using the data from NASA Ames Research Center proved to be a powerful application of the use of finding and calculating confidence intervals for the extrema of random polynomials. Since an eighth degree polynomial was fit to the data, only the Fairley method and bootstrapping were used to compute the confidence regions. Both methods gave similar results.

Some areas for additional research could include finding other methods for computing the confidence regions for the extrema of random

polynomials and investigating joint confidence intervals (instead of independent abscissa and ordinate confidence bounds). The joint confidence regions would give oval shaped areas (instead of rectangular boxes), and would give a slightly better indication of where the true extrema would lie if the estimated ordinate and abscissa are correlated.

The simulations for this thesis were limited to normal, exponential, and t -distribution noise, with relatively small variances. Another area of future research could be to continue investigation with additional non-normal errors. Similarly, higher order polynomials could be simulated, and the results compared. Also, different sample sizes could be used to test the methods. Additionally, these methods were only applied to single variate polynomials (i.e., $p(x)$); expanding the use of these methods to multivariate polynomials (i.e., $p(x_1, x_2, \dots, x_s)$, a random polynomial in s variables) could be useful.

Of particular interest would be working out the details and the programming of the delta method for higher order polynomials, since the method is so computationally efficient. With all of the mathematical software packages available, the partial derivatives are computable. Thus, reducing the complexity of the derivative and associated calculations for the delta method would be a worthwhile endeavor. Such software could be written and made publicly available.

Lastly, since the bootstrapping method is so computationally intensive, it may be interesting to see if there are techniques readily available to optimize the programming of the method to reduce computational times via software enhancements. This would allow either decreased computational time or a higher number of bootstrap samples to be used, which would tend to make the method more accurate.

References

- Bharucha-Reid, A. T., and Sambandham, M. (1986) *Random Polynomials*. Orlando: Academic Press.
- Casella, G. and Berger, R. L. (2002). *Statistical Inference* (2nd ed.). Pacific Grove: Duxbury.
- Fairley, W. B. (1968). Roots of Random Polynomials (Ph.D. thesis Harvard University, 1968).
- Fieller, E. C. (1954). Some Problems in Interval Estimation. *Journal of the Royal Statistical Society*. Series B 16, 175-185.
- Insightful Corp. (2005). S-Plus 7.0 for Windows, Enterprise Developer [Computer Software]. Seattle: Insightful.
- Meeker, W. and Escobar, L. (1998). *Statistical Methods for Reliability Data*. New York: John Wiley & Sons, Inc.
- Myers, R. H. (1990). *Classical and Modern Regression with Applications*. Belmont: PWS-KENT Publishing Company.
- Wackerly, D., Mendenhall III, W., and Scheaffer, R. (2002). *Mathematical Statistics with Applications* (6th ed.). Pacific Grove: Duxbury.

Appendix: S-Plus® Code

Index

Functions:

PolyMatrix.Normal.Order4.....	47
PolyMatrix.Exp.Order4.....	48
PolyMatrix.Tdist.Order4.....	49
PolyMatrix.Normal.Order3.....	50
PolyMatrix.Exp.Order3.....	50
PolyMatrix.Tdist.Order3.....	51
PolyMatrix.Normal.Order2.....	51
PolyMatrix.Exp.Order2.....	52
PolyMatrix.Tdist.Order2.....	52
FindMax.Order4.....	53
FindMax.Order3.....	54
FindMax.Order2.....	55

Simulations/Methods:

Fairley method for second degree polynomials.....	56
Delta method for second degree polynomials.....	58
Bootstrapping for second degree polynomials.....	60
Fairley method for third degree polynomials.....	61
Delta method for third degree polynomials.....	63
Bootstrapping for third degree polynomials.....	65
Fairley method for fourth degree polynomials.....	66
Bootstrapping for fourth degree polynomials.....	69

Graph Production:

Second degree example.....	70
Close up of second degree example.....	72
Third degree example.....	73
Fourth degree example.....	75

```
#####
## This function takes the polynomial  $y' = -(x-3)(x-(1-5i))(x-(1+5i))$  and creates
##      a matrix with the x and y values over a given interval
## Return:
##  A := matrix of polynomial values [y,x]
## Written by: Sandy DeSousa  July 16, 2005
#####
PolyMatrix.Normal.Order4 <- function()
{
  ## Generate a sequence that spans the values and compute y values
  x.vec<-seq.default(from=0.0, to=4.95, by=0.05)
  y.vec<-vector("numeric", 100)

  noise <- rnorm(100, 0, 1.5)

  for (i in 1:100)
    y.vec[i]<-(-(1/60)*x.vec[i]^4 + (1/9)*x.vec[i]^3 +
              - (16/15)*x.vec[i]^2 + (78/15)*x.vec[i] + noise[i])

  A<-matrix(nrow=100, ncol=2)
  A[1:100,1]<-Re(y.vec)
  A[1:100,2]<-Re(x.vec)

  ## result is the A matrix
  result<-A
  result
} ## PolyMatrix.Normal.Order4
```

```
#####
## This function takes the polynomial  $y' = -(x-3)(x-(1-5i))(x-(1+5i))$  and creates
##          a matrix with the x and y values over a given interval
## Return:
##  A := matrix of polynomial values [y,x]
## Written by: Sandy DeSousa July 16, 2005
#####
PolyMatrix.Exp.Order4 <- function()
{
  ## Generate a sequence that spans the values and compute y values
  x.vec<-seq.default(from=0.0, to=4.95, by=0.05)
  y.vec<-vector("numeric", 100)

  noise <- (rexp(100,scale = 1.5) - 1.5)

  for (i in 1:100)
    y.vec[i]<-(-(1/60)*x.vec[i]^4 + (1/9)*x.vec[i]^3 +
              - (16/15)*x.vec[i]^2 + (78/15)*x.vec[i] + noise[i])

  A<-matrix(nrow=100, ncol=2)
  A[1:100,1]<-Re(y.vec)
  A[1:100,2]<-Re(x.vec)

  ## result is the A matrix
  result<-A
  result
} ## PolyMatrix.Exp.Order4
```

```
#####
## This function takes the polynomial  $y' = -(x-3)(x-(1-5i))(x-(1+5i))$  and creates
##      a matrix with the x and y values over a given interval
## Return:
## A := matrix of polynomial values [y,x]
## Written by: Sandy DeSousa July 16, 2005
#####
PolyMatrix.Tdist.Order4 <- function()
{
  ## Generate a sequence that spans the values and compute y values
  x.vec<-seq.default(from=0.0, to=4.95, by=0.05)
  y.vec<-vector("numeric", 100)

  noise <- (rt(100,df = 3) / (sqrt(3)/1.3))

  for (i in 1:100)
    y.vec[i]<-(-(1/60)*x.vec[i]^4 + (1/9)*x.vec[i]^3 +
      - (16/15)*x.vec[i]^2 + (78/15)*x.vec[i] + noise[i])

  A<-matrix(nrow=100, ncol=2)
  A[1:100,1]<-Re(y.vec)
  A[1:100,2]<-Re(x.vec)

  ## result is the A matrix
  result<-A
  result
} ## PolyMatrix.Tdist.Order4
```

```
#####
## This function takes the polynomial  $y'=(3x-3)(x-4)$  and creates
##          a matrix with the x and y values over a given interval
## Return:
## A := matrix of polynomial values [y,x]
## Written by: Sandy DeSousa Sept 16, 2005
#####
PolyMatrix.Normal.Order3 <- function()
{
  ## Generate a sequence that spans the values and compute y values
  A<-matrix(nrow=100, ncol=2)
  A[1:100,2]<-seq.default(from=-0.5, to=5.44, by=0.06)
  noise <- rnorm(100,0,2.5)

  for (i in 1:100)
    A[i,1]<-(A[i,2]^3 - (7.5)*A[i,2]^2 + 12*A[i,2] + noise[i])

  ## result is the A matrix
  result<-A
  result
} ## PolyMatrix.Normal.Order3
```

```
#####
## This function takes the polynomial  $y'=(3x-3)(x-4)$  and creates
##          a matrix with the x and y values over a given interval
## Return:
## A := matrix of polynomial values [y,x]
## Written by: Sandy DeSousa Sept 16, 2005
#####
PolyMatrix.Exp.Order3 <- function()
{
  ## Generate a sequence that spans the values and compute y values
  A<-matrix(nrow=100, ncol=2)
  A[1:100,2]<-seq.default(from=-0.5, to=5.44, by=0.06)
  noise <- (rexp(100,scale = 2.5) - 2.5)

  for (i in 1:100)
    A[i,1]<-(A[i,2]^3 - (7.5)*A[i,2]^2 + 12*A[i,2] + noise[i])

  ## result is the A matrix
  result<-A
  result
} ## PolyMatrix.Normal.Order3
```

```
#####
## This function takes the polynomial  $y'=(3x-3)(x-4)$  and creates
##      a matrix with the x and y values over a given interval
## Return:
## A := matrix of polynomial values [y,x]
## Written by: Sandy DeSousa Sept 16, 2005
#####
PolyMatrix.Tdist.Order3 <- function()
{
  ## Generate a sequence that spans the values and compute y values
  A<-matrix(nrow=100, ncol=2)
  A[1:100,2]<-seq.default(from=-0.5, to=5.44, by=0.06)
  noise <- (rt(100,df = 3) / (sqrt(3)/2.5))

  for (i in 1:100)
    A[i,1]<-(A[i,2]^3 - (7.5)*A[i,2]^2 + 12*A[i,2] + noise[i])

  ## result is the A matrix
  result<-A
  result
} ## PolyMatrix.Tdist.Order3
```

```
#####
## This function takes the polynomial  $y'=-2x + 4$  and creates
##      a matrix with the x and y values over a given interval
## Return:
## A := matrix of polynomial values [y,x]
## Written by: Sandy DeSousa Sept 16, 2005
#####
PolyMatrix.Normal.Order2 <- function()
{
  ## Generate a sequence that spans the values and compute y values
  A<-matrix(nrow=100, ncol=2)
  A[1:100,2]<-seq.default(from=0, to=4.95, by=0.05)
  noise <- rnorm(100,0,1)

  for (i in 1:100)
    A[i,1]<-(-A[i,2]^2 + (4)*A[i,2] + 3 + noise[i])

  ## result is the A matrix
  result<-A
  result
} ## PolyMatrix.Normal.Order2
```

```
#####
## This function takes the polynomial  $y' = -2x + 4$  and creates
##          a matrix with the x and y values over a given interval
## Return:
## A := matrix of polynomial values [y,x]
## Written by: Sandy DeSousa July 16, 2005
#####
PolyMatrix.Exp.Order2 <- function()
{
  ## Generate a sequence that spans the values and compute y values
  A<-matrix(nrow=100, ncol=2)
  A[1:100,2]<-seq.default(from=0, to=4.95, by=0.05) # x values
  y.vec<-vector("numeric", 100)
  noise <- ( rexp(100, scale = 1) - 1 )

  for (i in 1:100)
    A[i,1]<-Re(-A[i,2]^2 + (4)*A[i,2] + 3 + noise[i])

  ## result is the A matrix
  result<-A
  result
} ## PolyMatrix.Exp.Order2
```

```
#####
## This function takes the polynomial  $y' = -2x + 4$  and creates
##          a matrix with the x and y values over a given interval
## Return:
## A := matrix of polynomial values [y,x]
## Written by: Sandy DeSousa Sept 16, 2005
#####
PolyMatrix.Tdist.Order2 <- function()
{
  ## Generate a sequence that spans the values and compute y values
  A<-matrix(nrow=100, ncol=2)
  A[1:100,2]<-seq.default(from=0, to=4.95, by=0.05)
  noise <- (rt(100,df = 3) / sqrt(3))

  for (i in 1:100)
    A[i,1]<-(-A[i,2]^2 + (4)*A[i,2] + 3 + noise[i])

  ## result is the A matrix
  result<-A
  result
} ## PolyMatrix.Normal.Order2
```

```
#####
## This function takes a matrix (of y and x coords) as input and y returns the
##   x, y coordinate of the maximum
## Input:
##   A: matrix of y and x coordinates
##   my.fit: data object returned from lsfit (contains coefficients)
## Return:
##   max := x and y coordinates of maximum
## Written by: Sandy DeSousa July 16, 2005
#####
```

```
FindMax.Order4 <- function(A, my.fit)
{
  ## the derivative is going to be the x coef, 2*x^2 coef, 3*x^3 coef, 4*x^4 coef
  derivative<-vector("numeric", 4)
  derivative[1] <- my.fit$coef[2]
  derivative[2] <- 2*my.fit$coef[3]
  derivative[3] <- 3*my.fit$coef[4]
  derivative[4] <- 4*my.fit$coef[5]

  # Pick the one real root, closest to 3, not the complex conjugate
  xmax<-polyroot(derivative)
  my.xmax <- Re(xmax[1])
  for(i in 2:3)
  {
    if(Im(xmax[i]) < 10^-10 ) # not a complex number
    {
      x.distance <- abs(my.xmax-3)
      for(j in 2:3)
      {
        if(abs(xmax[j]-3) < x.distance)
        {
          my.xmax <- Re(xmax[j])
          x.distance_<-abs(xmax[j]-3)
        }
      }
    }
  }
  ymax<-sum(my.fit$coef*c(1, my.xmax,my.xmax**2,my.xmax**3,my.xmax**4))

  ## result is the x,y coords of the max
  max<-vector("numeric", 2)
  max[1]<-Re(my.xmax)
  max[2]<-Re(ymax)
  result<-max
  result
} ## FindMax.Order4
```



```
#####
## This function takes a matrix (of y and x coords) as input and y returns the
##   x, y coordinate of the maximum
## Input:
##   A: matrix of y and x coordinates
##   my.fit: data object returned from lsfit (contains coefficients)
## Return:
##   max := x and y coordinates of maximum
## Written by: Sandy DeSousa  Sept 16, 2005
#####
```

```
FindMax.Order3 <- function(A, my.fit)
{
  ## the derivative is going to be the x coef, 2*x^2 coef, 3*x^3 coef
  derivative<-vector("numeric", 3)
  derivative[1] <- my.fit$coef[2]
  derivative[2] <- 2*my.fit$coef[3]
  derivative[3] <- 3*my.fit$coef[4]

  xmax<-polyroot(derivative)

  # get the right root should be approximately x=1

  for (i in 1:length(xmax))
  {
    if ( Re(xmax[i]) > -1 && Re(xmax[i]) < 3)
    {
      my.xmax <- xmax[i]
    }
  }
  ymax<-sum(my.fit$coef*c(1, my.xmax, my.xmax**2, my.xmax**3))

  ## result is the x,y coords of the max
  max<-vector("numeric", 2)
  max[1]<-Re(my.xmax)
  max[2]<-Re(ymax)
  result<-max
  result
} ## FindMax.Order3
```

```
#####
## This function takes a matrix (of y and x coords) as input and y returns the
## x, y coordinate of the maximum
## Input:
## A: matrix of y and x coordinates
## my.fit: data object returned from lsfit (contains coefficients)
## Return:
## max := x and y coordinates of maximum
## Written by: Sandy DeSousa Sept 16, 2005
#####
```

```
FindMax.Order2 <- function(A, my.fit)
{
  ## create derivative funtion
  ## the derivative is going to be the x coef, 2*x^2 coef
  derivative<-vector("numeric", 2)
  derivative[1] <- my.fit$coef[2]
  derivative[2] <- 2*my.fit$coef[3]

  xmax<-polyroot(derivative)
  ymax<-sum(my.fit$coef*c(1, xmax, xmax**2))

  ## result is the x,y coords of the max
  max<-vector("numeric", 2)
  max[1]<-Re(xmax)
  max[2]<-Re(ymax)
  result<-max
  result
} ## FindMax.Order2
```

```
#####
## The following code computes abscissa (x) and ordinate (y)
## confidence intervals via the Fairley method for
## second degree polynomials
#####
my.iter <- 2500
xy.max<-matrix(nrow=my.iter, ncol=2)
x.normal.ci<-matrix(nrow=my.iter, ncol=2)
y.normal1.ci<-matrix(nrow=my.iter, ncol=2)
y.normal2.ci<-matrix(nrow=my.iter, ncol=2)

my.xvec2<-vector("numeric", 2)
my.xvec3<-vector("numeric", 3)

for (i in 1:my.iter)
{
  A<-PolyMatrix.Tdist.Order2()

  ## find and return x,y coord of max
  my.fit <- lsfit(cbind(A[,2], A[,2]^2), A[,1])
  my.diag <- ls.diag(my.fit)

  my.max<-FindMax.Order2(A, my.fit)

  # Save the x and y values of the max for this iteration
  xy.max[i,1]<- my.max[1] # x value
  xy.max[i,2]<- my.max[2] # y value

  my.x.max = xy.max[i,1]

  my.vars <- my.diag$std.dev^2*my.diag$cov.unscaled[2:3,2:3]

  #march along x values to determine which ones include yprime(x)=0
  my.continue <- TRUE
  j <- 1
  while(my.continue)
  {
    my.xval <- my.x.max + (0.0001 * j)
    my.xvec2[1]<-1
    my.xvec2[2]<-2*my.xval
    my.yprime.hat <- (my.fit$coef[2] + 2*my.fit$coef[3]*my.xval)
    my.sigmahat <- sqrt(my.xvec2%*%my.vars%*%my.xvec2)
    my.ymax <- my.yprime.hat + 1.96 * my.sigmahat
    my.ymin <- my.yprime.hat - 1.96 * my.sigmahat

    if (my.ymin < 0 && my.ymax > 0)
    {
```

```

        x.normal.ci[i,2] <- my.xval
        j <- j+1
    }
    else
        my.continue <- FALSE
}

my.continue <- TRUE
j <- 1
while (my.continue)
{
    my.xval <- my.x.max - (0.0001 * j)
    my.xvec2[1]<-1
    my.xvec2[2]<-2*my.xval
    my.yprime.hat <- (my.fit$coef[2] + 2*my.fit$coef[3]*my.xval)
    my.sigmahat <- sqrt(my.xvec2%*%my.vars%*%my.xvec2)
    my.ymax <- my.yprime.hat + 1.96 * my.sigmahat
    my.ymin <- my.yprime.hat - 1.96 * my.sigmahat

    if (my.ymin < 0 && my.ymax > 0)
    {
        x.normal.ci[i,1] <- my.xval
        j <- j+1
    }
    else
        my.continue <- FALSE
}

# Now we need to compute the CI for the y value

my.vars <- my.diag$std.dev^2*my.diag$cov.unscaled[1:3,1:3]

my.xvec3[1]<-1
my.xvec3[2]<-my.x.max
my.xvec3[3]<-my.x.max**2
y.hat <- sum(my.fit$coef*my.xvec3)
my.sigmahat <- sqrt(my.xvec3%*%my.vars%*%my.xvec3)
y.normal1.ci[i,2] <- y.hat + 1.96 * my.sigmahat
y.normal1.ci[i,1] <- y.hat - 1.96 * my.sigmahat
}

```

```
#####
## The following code computes abscissa (x) and ordinate (y) confidence
## intervals via the delta method for second degree polynomials
#####
my.iter <- 2500
xy.max<-matrix(nrow=my.iter, ncol=2)
x.delta.ci<-matrix(nrow=my.iter, ncol=2)
y.delta.ci<-matrix(nrow=my.iter, ncol=2)
cov.xhat.yhat<-vector("numeric",my.iter)
cor.xhat.yhat<-vector("numeric",my.iter)

dg<-matrix(nrow=3, ncol=2)
var.matrix<-matrix(nrow=3, ncol=3)

for (i in 1:my.iter)
{
  A<-PolyMatrix.Normal.Order2()

  my.fit <- lsfit(cbind(A[,2], A[,2]^2), A[,1])
  my.diag <- ls.diag(my.fit)
  my.max<-FindMax.Order2(A, my.fit)

  # Save the x and y values of the max for this iteration
  xy.max[i,1]<- my.max[1] # x value
  xy.max[i,2]<- my.max[2] # y value

  var.matrix <- my.diag$std.dev^2*my.diag$cov.unscaled

  beta.0 = my.fit$coef[1]
  beta.1 = my.fit$coef[2]
  beta.2 = my.fit$coef[3]

  dg[1,1] <- 0
  dg[2,1] <- (-1/(2*beta.2))
  dg[3,1] <- (beta.1 / (2 * beta.2^2))
  dg[1,2] <- 1
  dg[2,2] <- (-1 * beta.1/(2*beta.2))
  dg[3,2] <- (beta.1^2 / (4 * beta.2^2))

  Var.g<- (t(dg)%*%var.matrix)%*%dg

  std.dev.xhat <- sqrt(Var.g[1,1])
  x.delta.ci[i,1] <- xy.max[i,1] - 1.96 * std.dev.xhat
  x.delta.ci[i,2] <- xy.max[i,1] + 1.96 * std.dev.xhat

  std.dev.yhat <- sqrt(Var.g[2,2])
  y.delta.ci[i,1] <- xy.max[i,2] - 1.96 * std.dev.yhat
  y.delta.ci[i,2] <- xy.max[i,2] + 1.96 * std.dev.yhat
}
```

```
y.delta.ci[i,2] <- xy.max[i,2] + 1.96 * std.dev.yhat  
  
cov.xhat.yhat[i] <- (Var.g[2,1])  
cor.xhat.yhat[i] <- (cov.xhat.yhat[i] / sqrt( std.dev.xhat^2 + std.dev.yhat^2))  
  
}
```

```
#####
## The following code computes abscissa (x) and ordinate (y) confidence
## intervals via bootstrapping for second degree polynomials
#####

my.iter <- 2500
my.bs.iter <- 2000
xy.max.bs <- matrix(nrow=my.bs.iter, ncol=2)
x.bs.ci <- matrix(nrow=my.iter, ncol=2)
y.bs.ci <- matrix(nrow=my.iter, ncol=2)

for (i in 1:my.iter)
{
  A <- PolyMatrix.Normal.Order2()

  for (j in 1:my.bs.iter)
  {
    which.pairs <- sample(x=1:100, size=100, replace=T)
    B <- A[which.pairs, ]

    ## find and return x,y coord of max
    my.fit <- lsfit(cbind(B[,2], B[,2]^2), B[,1])

    my.max <- FindMax.Order2(B, my.fit)

    # Save the x and y values of the max for this iteration
    xy.max.bs[j,1] <- my.max[1]
    xy.max.bs[j,2] <- my.max[2]
  }
  x.bs.ci[i, ] <- quantile(xy.max.bs[,1], probs = c(.025, .975))
  y.bs.ci[i, ] <- quantile(xy.max.bs[,2], probs = c(.025, .975))
}

```

```
#####
## The following code computes abscissa (x) and ordinate (y)
## confidence intervals via the Fairley method for third degree polynomials
#####
my.iter <- 2500
xy.max<-matrix(nrow=my.iter, ncol=2)
x.normal.ci<-matrix(nrow=my.iter, ncol=2)
y.normal1.ci<-matrix(nrow=my.iter, ncol=2)

my.xvec4<-vector("numeric", 4)
my.xvec3<-vector("numeric", 3)

for (i in 1:my.iter)
{
  A<-PolyMatrix.Normal.Order3()

  ## find and return x,y coord of max
  my.fit <- lsfit(cbind(A[,2], A[,2]^2, A[,2]^3), A[,1])
  my.diag <- ls.diag(my.fit)

  my.max<-FindMax.Order3(A, my.fit)

  # Save the x and y values of the max for this iteration
  xy.max[i,1]<- my.max[1] # x value
  xy.max[i,2]<- my.max[2] # y value

  my.x.max = xy.max[i,1]

  my.vars <- my.diag$std.dev^2*my.diag$cov.unscaled[2:4,2:4]

  #march along x values to determine which ones include yprime(x)=0
  my.continue <- TRUE
  j <- 1
  while(my.continue)
  {
    my.xval <- my.x.max + (0.0001 * j)
    my.xvec3[1]<-1
    my.xvec3[2]<-2*my.xval
    my.xvec3[3]<-3*my.xval**2
    my.yprime.hat <- (my.fit$coef[2] + 2*my.fit$coef[3]*my.xval
                     + 3*my.fit$coef[4]*my.xval**2 )
    my.sigmahat <- sqrt(my.xvec3%*%my.vars%*%my.xvec3)
    my.ymax <- my.yprime.hat + 1.96 * my.sigmahat
    my.ymin <- my.yprime.hat - 1.96 * my.sigmahat

    if (my.ymin < 0 && my.ymax > 0)
    {
```



```

        x.normal.ci[i,2] <- my.xval
        j <- j+1
    }
    else
        my.continue <- FALSE
}

my.continue <- TRUE
j <- 1
while (my.continue)
{
    my.xval <- my.x.max - (0.0001 * j)
    my.xvec3[1]<-1
    my.xvec3[2]<-2*my.xval
    my.xvec3[3]<-3*my.xval**2
    my.yprime.hat <- (my.fit$coef[2] + 2*my.fit$coef[3]*my.xval
                     + 3*my.fit$coef[4]*my.xval**2 )
    my.sigmahat <- sqrt(my.xvec3%*%my.vars%*%my.xvec3)
    my.ymax <- my.yprime.hat + 1.96 * my.sigmahat
    my.ymin <- my.yprime.hat - 1.96 * my.sigmahat

    if (my.ymin < 0 && my.ymax > 0)
    {
        x.normal.ci[i,1] <- my.xval
        j <- j+1
    }
    else
        my.continue <- FALSE
}

# Now we need to compute the CI for the y value

my.vars <- my.diag$std.dev^2*my.diag$cov.unscaled[1:4,1:4]

my.xvec4[1]<-1
my.xvec4[2]<-my.x.max
my.xvec4[3]<-my.x.max**2
my.xvec4[4]<-my.x.max**3
y.hat <- sum(my.fit$coef*my.xvec4)
my.sigmahat <- sqrt(my.xvec4%*%my.vars%*%my.xvec4)
y.normal1.ci[i,2] <- y.hat + 1.96 * my.sigmahat
y.normal1.ci[i,1] <- y.hat - 1.96 * my.sigmahat
}

```

```
#####
## The following code computes abscissa (x) and ordinate (y) confidence
## intervals via the delta method for third degree polynomials
#####
my.iter <- 2500
xy.max<-matrix(nrow=my.iter, ncol=2)
x.delta.ci<-matrix(nrow=my.iter, ncol=2)
y.delta.ci<-matrix(nrow=my.iter, ncol=2)
cov.xhat.yhat<-vector("numeric",my.iter)
cor.xhat.yhat<-vector("numeric",my.iter)

dg<-matrix(nrow=4, ncol=2)
var.matrix<-matrix(nrow=4, ncol=4)

for (i in 1:my.iter)
{
  A<-PolyMatrix.Normal.Order3()

  ## find and return x,y coord of max
  my.fit <- lsfit(cbind(A[,2], A[,2]^2, A[,2]^3), A[,1])
  my.diag <- ls.diag(my.fit)

  my.max<-FindMax.Order3(A, my.fit)

  # Save the x and y values of the max for this iteration
  xy.max[i,1]<- my.max[1] # x value
  xy.max[i,2]<- my.max[2] # y value

  var.matrix <- my.diag$std.dev^2*my.diag$cov.unscaled

  beta.0 = my.fit$coef[1]
  beta.1 = my.fit$coef[2]
  beta.2 = my.fit$coef[3]
  beta.3 = my.fit$coef[4]

  u <- sqrt(4*beta.2^2 - 12 * beta.1 * beta.3)
  v <- ((4 * beta.2) / u)
  w <- (( -2 * beta.2) - u)

  dg[1,1] <- 0
  dg[2,1] <- (1/u)
  dg[3,1] <- ((-2 - v) / (6 * beta.3))
  dg[4,1] <- (( beta.1 / (beta.3 * u)) - (w / (6 * beta.3^2)))
  dg[1,2] <- 1
  dg[2,2] <- ( (w/(6*beta.3)) + (beta.1 / u) + (((1/3)*beta.2*w) / (beta.3*u))
               + (((1/12) * w^2) / (beta.3 * u)) )
}
```

```

dg[3,2] <- ( ((beta.1 * (-2 - v)) / (6 * beta.3)) + ( w^2 / (36 * beta.3^2))
             + (((beta.2 * w) * (-2 - v)) / ( 18 * beta.3^2))
             + ((w^2 * (-2 - v)) / (72 * beta.3^2)) )

dg[4,2] <- ( (beta.1^2 / (beta.3 * u)) - ((beta.1 * w) / (6 * beta.3^2))
             + ((1/3) * (beta.1*beta.2*w) / ( beta.3^2 * u))
             - ((beta.2 * w^2) / ( 18 * beta.3^3))
             - ( w^3 / (108 * beta.3^3))
             + (( (1/12) * beta.1 * w^2) / ( beta.3^2 * u)) )

Var.g<- (t(dg)%*%var.matrix%*%dg)

std.dev.xhat <- sqrt(Var.g[1,1])

x.delta.ci[i,1] <- xy.max[i,1] - 1.96 * std.dev.xhat
x.delta.ci[i,2] <- xy.max[i,1] + 1.96 * std.dev.xhat

std.dev.yhat <- sqrt(Var.g[2,2])

y.delta.ci[i,1] <- xy.max[i,2] - 1.96 * std.dev.yhat
y.delta.ci[i,2] <- xy.max[i,2] + 1.96 * std.dev.yhat

cov.xhat.yhat[i] <- (Var.g[2,1])
cor.xhat.yhat[i] <- (cov.xhat.yhat[i] /
                    sqrt( std.dev.xhat^2 + std.dev.yhat^2))
}

```

```
#####
## The following code computes abscissa (x) and ordinate (y)
## confidence intervals via bootstrapping for third degree polynomials
#####
my.iter <- 2500
my.bs.iter <- 2000
xy.max.bs<-matrix(nrow=my.bs.iter, ncol=2)
x.bs.ci<-matrix(nrow=my.iter, ncol=2)
y.bs.ci<-matrix(nrow=my.iter, ncol=2)

for (i in 1:my.iter)
{

  A<-PolyMatrix.Exp.Order3()

  # Bootstrapping
  for (j in 1:my.bs.iter)
  {
    which.pairs <- sample(x=1:100, size=100, replace=T)
    B <- A[which.pairs, ]

    ## find and return x,y coord of max
    my.fit <- lsfit(cbind(B[,2], B[,2]^2, B[,2]^3), B[,1])

    my.max<-FindMax.Order3(B, my.fit)

    # Save the x and y values of the max for this iteration
    xy.max.bs[j,1]<- my.max[1]
    xy.max.bs[j,2]<- my.max[2]
  }
  x.bs.ci[i, ] <- quantile(xy.max.bs[,1], probs = c(.025, .975))
  y.bs.ci[i, ] <- quantile(xy.max.bs[,2], probs = c(.025, .975))
}

```

```
#####
## The following code computes abscissa (x) and ordinate (y) confidence
## intervals via the Fairley method for fourth degree polynomials
#####
my.iter <- 2500
bad.count <- 0
xy.max<-matrix(nrow=my.iter, ncol=2)
x.normal.ci<-matrix(nrow=my.iter, ncol=2)
y.normal1.ci<-matrix(nrow=my.iter, ncol=2)

my.xvec4<-vector("numeric", 4)
my.xvec5<-vector("numeric", 5)

for (i in 1:my.iter)
{
  A<-PolyMatrix.Tdist.Order4()

  ## find and return x,y coord of max
  my.fit <- lsfit(cbind(A[,2], A[,2]^2, A[,2]^3, A[,2]^4), A[,1])
  my.diag <- ls.diag(my.fit)

  my.max<-FindMax.Order4(A, my.fit)

  # Save the x and y values of the max for this iteration
  xy.max[i,1]<- my.max[1] # x value
  xy.max[i,2]<- my.max[2] # y value
  my.x.max = my.max[1]

  my.vars <- my.diag$std.dev^2*my.diag$cov.unscaled[2:5,2:5]

  #march along x values to determine which ones include y=prime = 0
  skip <- FALSE
  my.continue <- TRUE
  j <- 1
  while(my.continue)
  {
    my.xval <- my.x.max + (0.001 * j)
    my.xvec4[1]<-1
    my.xvec4[2]<-2*my.xval
    my.xvec4[3]<-3*my.xval^2
    my.xvec4[4]<-4*my.xval^3
    my.yprime.hat <- (my.fit$coef[2] + 2*my.fit$coef[3]*my.xval +
      3*my.fit$coef[4]*my.xval^2 + 4*my.fit$coef[5]*my.xval^3)
    var <- my.xvec4%*%my.vars%*%my.xvec4
    my.sigmahat <- sqrt(var)
    my.ymax <- my.yprime.hat + 1.96 * my.sigmahat
    my.ymin <- my.yprime.hat - 1.96 * my.sigmahat
  }
}
```

```

    if (my.ymin < 0 && my.ymax > 0)
    {
        x.normal.ci[i,2] <- my.xval
        j <- j+1
    }
    else
        my.continue <- FALSE

# Need to be able to get out if there is a runaway effect
if (j > 2000)
{
    my.continue <- FALSE
    skip <- TRUE
    bad.count <- bad.count + 1
}
}

if (skip != TRUE) # no runaway effect yet
{
    my.continue <- TRUE
    j <- 1
    while (my.continue)
    {
        my.xval <- my.x.max - (0.001 * j)
        my.xvec4[1] <- 1
        my.xvec4[2] <- 2*my.xval
        my.xvec4[3] <- 3*my.xval^2
        my.xvec4[4] <- 4*my.xval^3
        my.yprime.hat <- (my.fit$coef[2] + 2*my.fit$coef[3]*my.xval +
                        3*my.fit$coef[4]*my.xval^2 + 4*my.fit$coef[5]*my.xval^3)
        var <- my.xvec4%%my.vars%%my.xvec4
        my.sigmahat <- sqrt(var)
        my.ymax <- my.yprime.hat + 1.96 * my.sigmahat
        my.ymin <- my.yprime.hat - 1.96 * my.sigmahat

        if (my.ymin < 0 && my.ymax > 0)
        {
            x.normal.ci[j,1] <- my.xval
            j <- j+1
        }
        else
            my.continue <- FALSE

# Need to be able to get out if there is a runaway effect
if (j > 2000)
{

```

```

        my.continue <- FALSE
        skip <- TRUE
        bad.count <- bad.count + 1
    }
}

# Now we need to compute the CI for the y value - if x was successful
if (skip != TRUE)
{
    my.xvec5[1]<-1
    my.xvec5[2]<-my.x.max
    my.xvec5[3]<-my.x.max^2
    my.xvec5[4]<-my.x.max^3
    my.xvec5[5]<-my.x.max^4
    y.hat <- sum(my.fit$coef*my.xvec5)
    my.vars <- my.diag$std.dev^2*my.diag$cov.unscaled[1:5,1:5]
    var <- my.xvec5%*%my.vars%*%my.xvec5
    my.sigmahat <- sqrt(var)
    y.normal1.ci[i,2] <- y.hat + 1.96 * my.sigmahat
    y.normal1.ci[i,1] <- y.hat - 1.96 * my.sigmahat
}
}

```

```
#####
## The following code computes abscissa (x) and ordinate (y)
## confidence intervals via bootstrapping for
## fourth degree polynomials
#####
my.iter <- 2500
my.bs.iter <- 2000
xy.max.bs <- matrix(nrow=my.bs.iter, ncol=2)
x.bs.ci <- matrix(nrow=my.iter, ncol=2)
y.bs.ci <- matrix(nrow=my.iter, ncol=2)

# Bootstrapping method
for (i in 1:my.iter)
{
  A <- PolyMatrix.Tdist.Order4()

  for (j in 1:my.bs.iter)
  {
    which.pairs <- sample(x=1:100, size=100, replace=T)
    B <- A[which.pairs, ]

    ## find and return x,y coord of max
    my.fit <- lsfit(cbind(B[,2], B[,2]^2, B[,2]^3, B[,2]^4), B[,1])
    my.diag <- ls.diag(my.fit)

    my.max <- FindMax.Order4(B, my.fit)

    # Save the x and y values of the max for this iteration
    xy.max.bs[j,1] <- my.max[1]
    xy.max.bs[j,2] <- my.max[2]
  }
  x.bs.ci[i, ] <- quantile(xy.max.bs[,1], probs = c(.025, .975))
  y.bs.ci[i, ] <- quantile(xy.max.bs[,2], probs = c(.025, .975))
}
}
```



```
#####
## The following code creates a plot, as in Figure 3.1, which is
##   an example of a second degree random polynomial with
##   normally distributed noise and associated confidence
##   intervals as calculated via the delta method
#####
# order 2 - Normal Noise and Delta Method

A<-PolyMatrix.Normal.Order2()

# First plot the simulated data

plot(A[1:100,2], A[1:100,1], type='n',
      xlab="X", ylab="Y")
points(A[1:100,2], A[1:100,1], pch=16, cex=0.5)

legend(c(1, 3), c(0, 1.5), legend=c('True Polynomial',
  'Estimated Polynomial'), lty=1:2)

#Calculate Confidence Intervals via the delta method
x.delta.ci<-vector("numeric", 2)
y.delta.ci<-vector("numeric", 2)
dg<-matrix(nrow=3, ncol=2)
var.matrix<-matrix(nrow=3, ncol=3)
my.fit <- lsfit(cbind(A[,2], A[,2]^2), A[,1])
my.diag <- ls.diag(my.fit)
my.max<-FindMax.Order2(A, my.fit)

# Save the x and y values of the max for this iteration
x.max<- my.max[1] # x value
y.max<- my.max[2] # y value

var.matrix <- my.diag$std.dev^2*my.diag$cov.unscaled

beta.0 = my.fit$coef[1]
beta.1 = my.fit$coef[2]
beta.2 = my.fit$coef[3]

dg[1,1] <- 0
dg[2,1] <- (-1/(2*beta.2))
dg[3,1] <- (beta.1 / (2 * beta.2^2))
dg[1,2] <- 1
dg[2,2] <- (-1 * beta.1/(2*beta.2))
dg[3,2] <- (beta.1^2 / (4 * beta.2^2))

Var.g<- (t(dg)%*%var.matrix)%*%dg)
```

```

std.dev.xhat <- sqrt(Var.g[1,1])

x.delta.ci[1] <- x.max - 1.96 * std.dev.xhat
x.delta.ci[2] <- x.max + 1.96 * std.dev.xhat

std.dev.yhat <- sqrt(Var.g[2,2])

y.delta.ci[1] <- y.max - 1.96 * std.dev.yhat
y.delta.ci[2] <- y.max + 1.96 * std.dev.yhat

# First we will add the true polynomial line

x.values.true <- seq.default(from=0, to=4.95, by=0.05)
y.values.true <- -x.values.true^2 + 4 * x.values.true + 3
lines(x.values.true, y.values.true, lty=1)

# Next we need to add the estimated polynomial line

x.values.est <- seq.default(from=0, to=4.95, by=0.05)
y.values.est<- vector("numeric", 100)
my.xvec3<-vector("numeric", 3)
for (i in 1:100)
{
  my.xvec3[1]<-1
  my.xvec3[2]<-x.values.est[i]
  my.xvec3[3]<-x.values.est[i]**2

  y.values.est[i] <- sum(my.fit$coef*my.xvec3)
}

lines(x.values.est, y.values.est, lty=2, lwd=3)

#Now graph confidence intervals
x.values <- seq.default(from=x.delta.ci[1], to=x.delta.ci[2], by=0.001)
y.values <- seq.default(from=y.delta.ci[1], to=y.delta.ci[2], by=0.001)

#Need 4 line segments to create box

lines(x.values,rep(y.delta.ci[1],length(x.values)))
lines(x.values,rep(y.delta.ci[2],length(x.values)))
lines(rep(x.delta.ci[1],length(y.values)), y.values)
lines(rep(x.delta.ci[2],length(y.values)), y.values)

```

```
#####
## The following code creates a zoomed plot (including the true and
##     estimated maximum (as in Figure 3.2) which is an example
##     of a second degree random polynomial with normally
##     distributed noise and associated confidence intervals
##     as calculated via the delta method
#####
#### Now we want a zoomed in version of the same graph

plot(A[30:50,2], A[30:50,1], type='n' ,
      xlab="X", ylab="Y", cex=1)
points(A[30:50,2], A[30:50,1], pch=16, cex=0.5)
legend(c(2.0, 2.4), c(4.2, 5), legend=c('True Max',
    'Estimated Max'), marks=c(4,3), cex=1)

# true and estimated max
points(x.max, y.max, pch=3, cex=2)
points(2,7, pch = 4, cex=2, cex=2)

# true polynomial
lines(x.values.true[30:50], y.values.true[30:50], lty=1)

# estimated polynomial
lines(x.values.est[30:50], y.values.est[30:50], lty=2, lwd=2)

# confidence intervals
lines(x.values,rep(y.delta.ci[1],length(x.values)))
lines(x.values,rep(y.delta.ci[2],length(x.values)))
lines(rep(x.delta.ci[1],length(y.values)), y.values)
lines(rep(x.delta.ci[2],length(y.values)), y.values)
```

```
#####
## The following code creates a plot, as in Figure 3.3, which is
##   an example of a third degree random polynomial with
##   exponentially distributed noise and associated confidence
##   intervals as calculated via bootstrapping
#####
# order 3 - Exponential Noise and Bootstrapping method

A<-PolyMatrix.Exp.Order3()

# First plot the simulated data

plot(A[1:100,2], A[1:100,1], type='n' ,
      xlab="X", ylab="Y")
points(A[1:100,2], A[1:100,1], pch=16, cex=0.5)

legend(c(3, 5.5), c(5, 8), legend=c('True Polynomial',
  'Estimated Polynomial'), lty=1:2)

x.bs.ci.pix<-vector("numeric", 2)
y.bs.ci.pix<-vector("numeric", 2)

# Bootstrapping confidence interval calculations
for (j in 1:2000)
{
  which.pairs <- sample(x=1:100, size=100, replace=T)
  B <- A[which.pairs, ]

  ## find and return x,y coord of max
  my.fit <- lsfit(cbind(B[,2], B[,2]^2, B[,2]^3), B[,1])

  my.max<-FindMax.Order3(B, my.fit)

  # Save the x and y values of the max for this iteration
  xy.max.bs[j,1]<- my.max[1]
  xy.max.bs[j,2]<- my.max[2]
}
x.bs.ci.pix[ ] <- quantile(xy.max.bs[ ,1], probs = c(.025, .975))
y.bs.ci.pix[ ] <- quantile(xy.max.bs[ ,2], probs = c(.025, .975))

# First we will add the true polynomial line
x.values.true <- seq.default(from=-0.5, to=5.44, by=0.06)
y.values.true <- x.values.true^3 - 7.5 * x.values.true^2 + 12 * x.values.true
lines(x.values.true, y.values.true, lty=1)

# Next we need to add the estimated polynomial line
```

```

my.fit <- lsfit(cbind(A[,2], A[,2]^2, A[,2]^3), A[,1])
my.diag <- ls.diag(my.fit)
my.max<-FindMax.Order3(A, my.fit)

x.values.est <- seq.default(from=-0.5, to=5.44, by=0.06)
my.xvec4<-vector("numeric", 4)
for (i in 1:100)
{
  my.xvec4[1]<-1
  my.xvec4[2]<-x.values.est[i]
  my.xvec4[3]<-x.values.est[i]^2
  my.xvec4[4]<-x.values.est[i]^3

  y.values.est[i] <- sum(my.fit$coef*my.xvec4)
}

lines(x.values.est, y.values.est, lty=2, lwd=3)

#Now graph confidence intervals
x.values <- seq.default(from=x.bs.ci.pix[1], to=x.bs.ci.pix[2], by=0.001)
y.values <- seq.default(from=y.bs.ci.pix[1], to=y.bs.ci.pix[2], by=0.001)

#Need 4 line segments to create box

lines(x.values,rep(y.bs.ci.pix[1],length(x.values)))
lines(x.values,rep(y.bs.ci.pix[2],length(x.values)))
lines(rep(x.bs.ci.pix[1],length(y.values)), y.values)
lines(rep(x.bs.ci.pix[2],length(y.values)), y.values)

```

```
#####
## The following code creates a plot, as in Figure 3.4, which is
##   an example of a fourth degree random polynomial with
##   t-distributed noise and associated confidence
##   intervals as calculated via the Fairley method
#####
# order 4 - T-dist Noise and Fairleys method
```

```
A<-PolyMatrix.Tdist.Order4()
```

```
# First plot the simulated data
```

```
plot(A[1:100,2], A[1:100,1], type='n' ,
      xlab="X", ylab="Y")
points(A[1:100,2], A[1:100,1], pch=16, cex=0.5)
```

```
legend(c(2, 4), c(-1, 1), legend=c('True Polynomial',
  'Estimated Polynomial'), lty=1:2)
```

```
x.fairley.ci<-vector("numeric", 2)
y.fairley.ci<-vector("numeric", 2)
```

```
# Compute the confidence intervals
```

```
my.xvec4<-vector("numeric", 4)
my.xvec5<-vector("numeric", 5)
```

```
## find and return x,y coord of max
my.fit <- lsfit(cbind(A[,2], A[,2]^2, A[,2]^3, A[,2]^4), A[,1])
my.diag <- ls.diag(my.fit)
```

```
my.max<-FindMax.Order4(A, my.fit)
# Save the x and y values of the max for this iteration
xy.max[1]<- my.max[1] # x value
xy.max[2]<- my.max[2] # y value
```

```
my.vars <- my.diag$std.dev^2*my.diag$cov.unscaled[2:5,2:5]
```

```
#march along x values to determine which ones include 0
```

```
my.continue <- TRUE
```

```
j <- 1
```

```
while(my.continue)
```

```
{
  my.xval <- xy.max[1] + (0.003 * j)
  my.xvec4[1]<-1
  my.xvec4[2]<-2*my.xval
  my.xvec4[3]<-3*my.xval**2
  my.xvec4[4]<-4*my.xval**3
```

```

my.yprime.hat <- (my.fit$coef[2] + 2*my.fit$coef[3]*my.xval +
  3*my.fit$coef[4]*my.xval**2 + 4*my.fit$coef[5]*my.xval**3)
var <- my.xvec4%%my.vars%%my.xvec4
my.sigmahat <- sqrt(var)
my.ymax <- my.yprime.hat + 1.96 * my.sigmahat
my.ymin <- my.yprime.hat - 1.96 * my.sigmahat

if (my.ymin < 0 && my.ymax > 0)
{
  x.fairley.ci[2] <- my.xval
  j <- j+1
}
else
  my.continue <- FALSE
}

my.continue <- TRUE
j <- 1
while (my.continue)
{
  my.xval <- xy.max[1] - (0.003 * j)
  my.xvec4[1]<-1
  my.xvec4[2]<-2*my.xval
  my.xvec4[3]<-3*my.xval**2
  my.xvec4[4]<-4*my.xval**3
  my.yprime.hat <- (my.fit$coef[2] + 2*my.fit$coef[3]*my.xval +
    3*my.fit$coef[4]*my.xval**2 + 4*my.fit$coef[5]*my.xval**3)
  var <- my.xvec4%%my.vars%%my.xvec4
  my.sigmahat <- sqrt(var)
  my.ymax <- my.yprime.hat + 1.96 * my.sigmahat
  my.ymin <- my.yprime.hat - 1.96 * my.sigmahat

  if (my.ymin < 0 && my.ymax > 0)
  {
    x.fairley.ci[1] <- my.xval
    j <- j+1
  }
  else
    my.continue <- FALSE
}

# Now we need to compute the CI for the y value

my.xvec5[1]<-1
my.xvec5[2]<-xy.max[1]
my.xvec5[3]<-xy.max[1]**2
my.xvec5[4]<-xy.max[1]**3

```

```

my.xvec5[5]<-xy.max[1]**4
y.hat <- sum(my.fit$coef*my.xvec5)
my.vars <- my.diag$std.dev^2*my.diag$cov.unscaled[1:5,1:5]
var <- my.xvec5%*%my.vars%*%my.xvec5
my.sigmahat <- sqrt(var)
y.fairley.ci[2] <- y.hat + 1.96 * my.sigmahat
y.fairley.ci[1] <- y.hat - 1.96 * my.sigmahat

# First we will add the true polynomial line

x.values.true <- seq.default(from=0, to=4.95, by=0.05)
y.values.true <- ((1/15) * (-(1/4)*x.values.true ^4 + (5/3)*x.values.true ^3 +
- 16*x.values.true ^2 + 78*x.values.true ))
lines(x.values.true , y.values.true , lty=1)

# Next we need to add the estimated polynomial line

x.values.est <- seq.default(from=0, to=4.95, by=0.05)
my.xvec5<-vector("numeric", 5)
for (i in 1:100)
{
  my.xvec5[1]<-1
  my.xvec5[2]<-x.values.est[i]
  my.xvec5[3]<-x.values.est[i]^2
  my.xvec5[4]<-x.values.est[i]^3
  my.xvec5[5]<-x.values.est[i]^4

  y.values.est[i] <- sum(my.fit$coef*my.xvec5)
}

lines(x.values.est, y.values.est, lty=2, lwd=3)

#Now graph confidence intervals
x.values <- seq.default(from=x.fairley.ci[1], to=x.fairley.ci[2], by=0.01)
y.values <- seq.default(from=y.fairley.ci[1], to=y.fairley.ci[2], by=0.01)

#Need 4 line segments to create box

lines(x.values,rep(y.fairley.ci[1],length(x.values)))
lines(x.values,rep(y.fairley.ci[2],length(x.values)))
lines(rep(x.fairley.ci[1],length(y.values)), y.values)
lines(rep(x.fairley.ci[2],length(y.values)), y.values)

```