MICROCOMPUTER IMPLEMENTATION OF ROBUST REGRESSION TECHNIQUES

by

Dana Detwiler

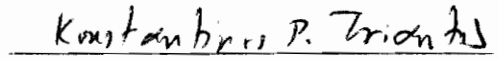
Report submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

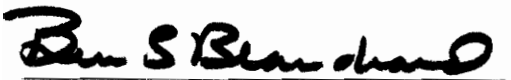
in

Systems Engineering

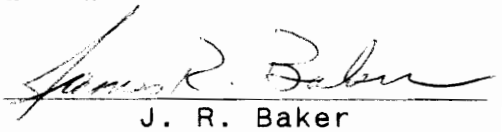
APPROVED:



K. P. Tiantis, Chairman



B. S. Blanchard



J. R. Baker

November, 1993

Blacksburg VA

MICROCOMPUTER IMPLEMENTATION OF ROBUST REGRESSION TECHNIQUES

by

Dana Detwiler

Committee Chairman: Kostos P. Triantis
Industrial and Systems Engineering

(ABSTRACT)

Systems Engineering requires that relationships between variables under control (independent variables) and variables which are to be controlled (dependent variables) be quantified. The most common method for characterizing this relationship is to use least squares regression. This method is inadequate in cases where some of the observed relationships between variables is contaminated with aberrant data.

This project provides two microcomputer based tools that will aid the systems engineer in identifying such instances of aberrant observations. Although mainframe implementations of these tools exist, microcomputer implementations of these tools are needed to accommodate many engineers which find microcomputers more accessible than mainframe computers.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure

LIST OF TABLES

Table

INTRODUCTION

The purpose of this project is to develop a tool to aid the systems engineer in isolating instances of aberrant system behavior. The tool that has been developed for this project uses robust regression techniques to isolate instances of aberrant system behavior. Although there are several robust regression software packages designed for implementation on computer mainframes, there is a shortage of such software designed to run on microcomputers. Because of advances in the capabilities of microcomputers, distributed client/server computer architecture is becoming more cost effective than mainframe centered architecture for many applications. This trend will continue as microcomputers become more powerful and advances are made in the development of client/server software. Thus, the need for a microcomputer based robust regression implementation will become more acute as mainframe computers are replaced by microcomputers.

The goal of the systems engineering process is to develop a system that will meet the user's need at a minimum cost. This goal is achieved in two ways: through system design and system improvement. The major thrust of systems engineering is to meet the needs of the system user by expending necessary resources during

the system design phase to ensure that the system will be adequate when it is implemented. The premise of this philosophy is that the earlier in the system life-cycle a problem is identified and solved, the cheaper the solution will be. It is most cost effective to identify and solve problems early in the system life-cycle. One technique used to identify problems early in the life-cycle is to model the system. The model may be a physical model of the system or a mathematical model of the characteristics of the system. In the formation of mathematical models, it is essential to identify the relationships between the parameters under the control of the systems engineer and the aspects of the system that are to be affected by these parameters.

In addition to determining system design parameters, the systems engineer must provide adjustments to optimize system performance after delivery. Even if the systems engineer were entirely successful in developing a system design that met the user's need, adjustments to the system would be necessary to adapt the system to changes in the system environment or changes in the needs of the system users. When it is determined that some adjustment is necessary during the operation of a system, a feedback loop is designed into the system. This feedback loop uses

information regarding the performance of the system to initiate a change in system parameters when necessary. This concept of a feedback loop used to control system performance can be expanded to a feedback loop designed to improve system performance. The system improvement feedback loop is initiated by analysis of system performance data followed by the necessary corrective action. One goal of this analysis is to isolate instances of aberrant system behavior. Instances of aberrant system behavior are often called outliers, because they lie outside the boundaries of normal system behavior. Seaver and Triantis (1994) define outliers as "an observation (or subset of observations) which appears to be inconsistent with the remainder of the data". Once these instances of aberrant behavior are isolated, investigation is initiated to hypothesize the cause of the aberrant behavior. The hypothesis can then be tested either through experimentation or from the collection and analysis of more data. Before action can be taken to improve the system, the relationship between the aberrant system behavior and the cause must be quantified. Several alternatives are available once the relationship between the cause and the aberrant behavior is established. The cause can be eliminated, ignored, or monitored. The choice of alternative is based on the

impact of the aberration on the capability of the system, the feasibility of eliminating the cause, the feasibility of monitoring the cause, and the frequency of the aberration. After an alternative is selected an implementation plan can be developed. This plan is subjected to the same rigor as any other plan to change the system architecture. The implementation of the system improvement plan is subject to all configuration control procedures and design reviews. The purpose of these controls and reviews is to ensure that in solving the identified problem, another problem is not created. At this point the system improvement feedback loop closes, and system performance data is monitored to ensure that the improvement implementation was successful, and to detect subsequent aberrant system behavior.

In each of the systems engineering methods mentioned above: design, adjustment, and process improvement, regression analysis often plays a crucial role. During system design, it is necessary to quantify the relationship between the design parameters, and cost and system performance. In some instances, these relationships are straightforward mathematical relationships. In these cases, the optimal solution can be obtained through differentiating the mathematical

relationships. In other cases the relationships are complex and may involve random variation. These relationships are often modeled using regression analysis. Feedback loops that initiate changes may also be based on regression analysis. Data may be collected during system testing to determine the relationship between system parameters and operational performance. This data may be summarized through regression analysis and the results of this analysis used in the implementation of the feedback loop. The system improvement process is based on the premise that abnormal system behavior can be identified. If random variations in system performance are inherent in the system, robust regression techniques can be used to separate aberrant instances from those which are due to normal system fluctuation.

Classic regression techniques model the relationship between a dependent variable y and a group of n independent variables x_i (i=1,2,...n) as a linear function of form $p_i x_i = y$. If there are m observations of n independent variables and m corresponding dependent variables, and we let x_{ij} be the jth observation of the ith observation and let y_j be the dependent variable associated with the jth observation, then classic regression will compute coefficients, p_i which minimize

the quantity:

$$\sum_{j=1}^{m}\left(\left(\left(\sum_{i=1}^{n} p_i x_{ij}\right)-y_j\right)^2\right)$$

This approach to modeling the relationships between dependent and independent variables is widely used because the estimates of the regression coefficients are unbiased and consistent. An estimator is unbiased if its expected value is equal to the value of the parameter that is being estimated and consistent if it reduces the error associated with the estimate as more data is incorporated into the model. A detailed description of unbiased and consistent estimators can be found in Stark and Woods (1986). The weakness of this method is that it is sometimes impossible to determine from analysis of the regression results that some of the data is aberrant. Robust techniques provide a means of isolating cases in which the variation of the relationship between the dependent and independent variables is due to some cause other than random variation. This ability to isolate aberrant cases is achieved at a cost in computational complexity. Robust regression techniques require more complex algorithms and more computational resources to achieve results than classic regression techniques.

The robust techniques implemented in this project select coefficients of the observations (p_is) to fit the data that is not aberrant. Since aberrant data is ignored, the aberrant observations can be isolated as those with very large residuals. In system design and adjustment, the computational overhead required for robust techniques may not be justified; however, since it is essential to isolate aberrant cases in the system improvement process, robust regression techniques are an indispensible method used in system improvement.

The first section of this report is a review of literature available on the topic of robust regression methods and algorithms. This section summarizes the characteristics of robust regression methods, and focuses on comparing various robust regression algorithms. The second section discusses the implementation of the software and focuses on the capabilities and limitations of the software. The third section contains the test results used to verify that the algorithms are implemented correctly. The software developed for this project has been benchmarked against mainframe implementations of robust regression techniques that have undergone extensive testing. The final section in this report outlines the conclusions of the project and focuses on the lessons learned during

the project and ways in which the robust regression techniques implemented in this project could be improved. The final section also contains suggestions for further research. Appendixes A through F contain details of the implementation of this project. Appendix G contains the presentation charts which were used in the defense of this project.

REVIEW OF LITERATURE

Rousseeuw and Leroy (1987) provide an excellent explanation of the impact that outliers can have on regression analysis and why these outliers are sometimes difficult to detect using classic regression results. They also provide a history of techniques designed to deal with the problem and compare the merits and shortfalls of each technique.

Rousseeuw and Leroy show why it is sometimes possible to identify a data point as an outlier from analysis of classic regression results and other times it is not. If the values of the independent variables for an aberrant observation are not significantly different than those of the other observations, then the residual for the aberrant observation will be significantly larger than the other residuals. If, however, the value of the independent variables of the aberrant data are significantly different than the those of the other observations, the largest residual will not belong to the aberrant data. This can be seen in Figures 1 and 2 from Rousseeuw and Leroy (reprinted with permission of John Wiley & Sons, Inc. Copyright 1987). Figure 1a represents the least squares solution to a group of x-y pairs that, for the most part, lie on the same line. Figure 1b represents the same line with a

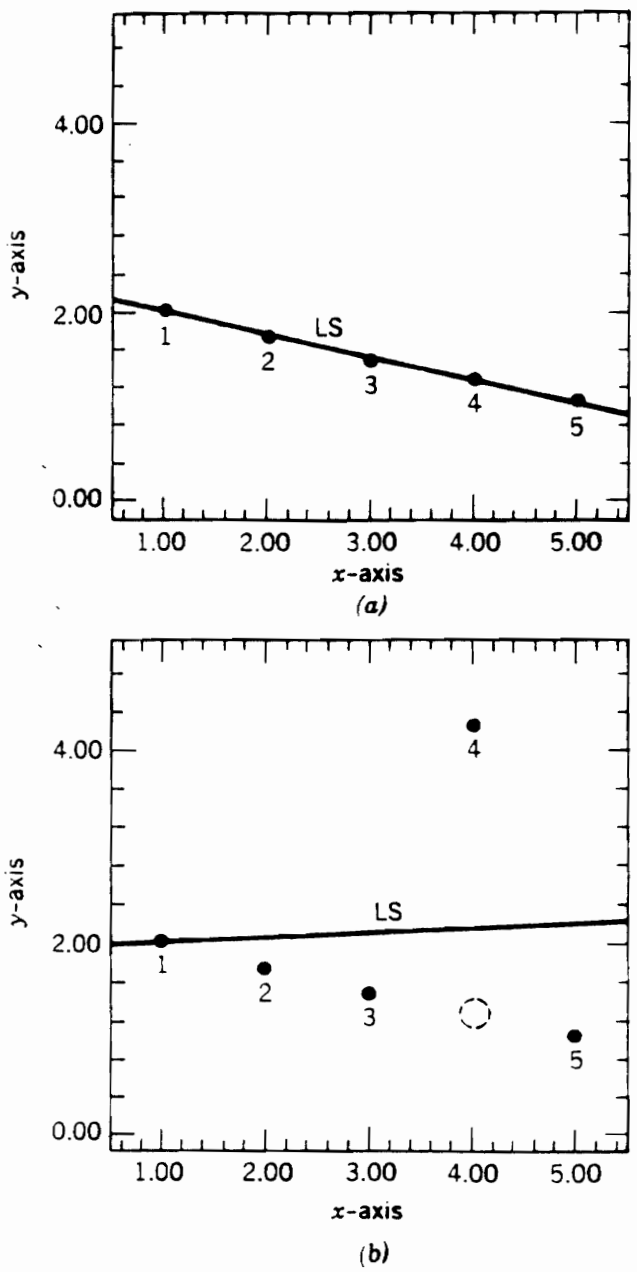


Figure 1

Effect of Dependent Variable Displacement

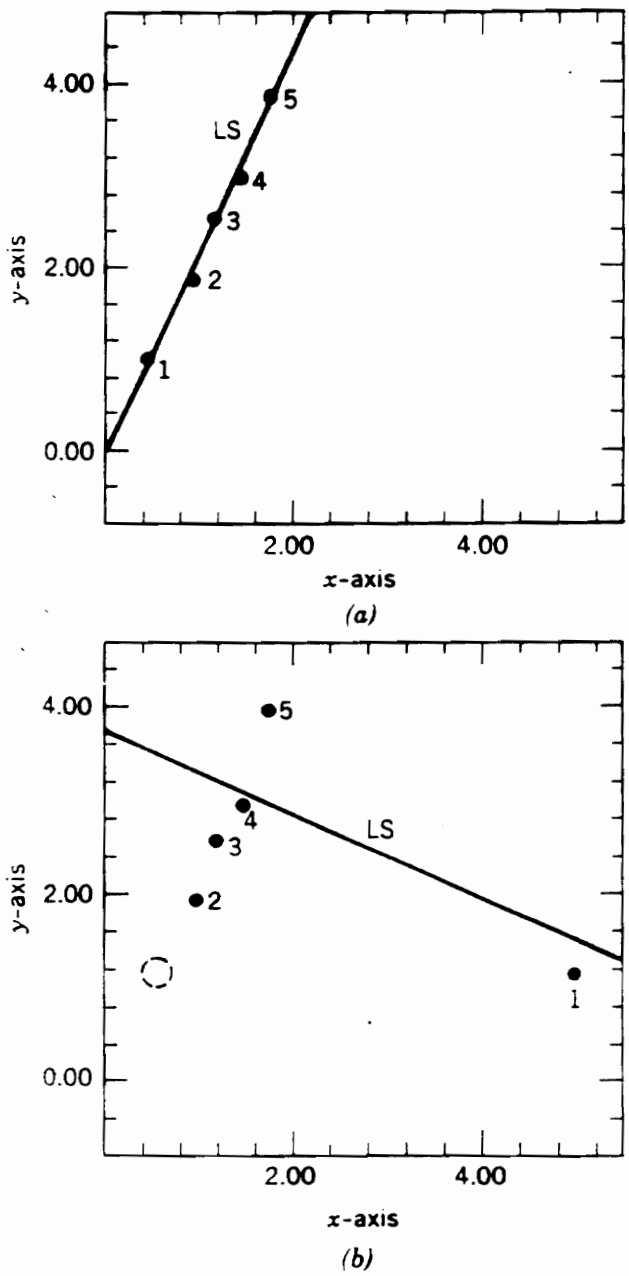


Figure 2

Effect of Independent Variable Displacement

single data point perturbed in the y (dependent variable) direction. It can be seen that the largest residual is associated with the aberrant data point. It is conceivable that through analysis of regression results we would be able to identify the aberrant datum since it has the largest residual. Figure 2a also shows a set of points that, for the most part, lie on the same line. In contrast to 1b, however, 2b shows one of the data points has been perturbed in the x (independent variable) direction. This figure shows that the aberrant data point does not yield the largest residual, in fact, it is one of the smaller residuals. Because the aberrant data has a small residual, it is difficult to identify through analysis of the residuals of the least squares solution. Although Figures 1 and 2 illustrate cases with a single independent variable, the concept can be extended to cases with multiple independent variables.

Rousseeuw and Leroy designate observations with independent variables that differ significantly from the other observations under consideration in a regression model as leverage points. They go on to point out that while a leverage point has more impact on the regression solution than the other observations under consideration, the leverage point is not necessarily

aberrant and should not be disregarded simply because it is a leverage point. Aberrant observations have a relationship between their independent variables and dependent variables which is inconsistent with the remainder of the observations; leverage points have independent variables which take on values inconsistent with the remainder of the observations. Since leverage points have the potential for skewing regression results, the impact that these points have on the regression solution must be identified. If the regression results would not be grossly affected by omitting the leverage points, these leverage points can be retained as valid data and used to improve the estimate of the regression coefficients. If, however, the regression solution differs significantly when the leverage points are omitted from the regression model, it is not appropriate to use the leverage points as observations to a regression model that will be used to characterize data across the entire span of independent variable values. Aberrant data is always discarded when forming a regression model. The problem of identifying leverage points becomes more difficult as the number of independent variables increases. In an application with n independent variables, the determination of leverage points is accomplished in n dimensional space.

Rousseeuw and Leroy further define the "breakdown point" of a regression model as the percent of aberrant observations that will not impact regression results. The definition of the breakdown point is based on the number of aberrant observations that could cause the regression solution to differ significantly from what would be expected without aberrant data. In some cases, a regression model's capability to yield results insensitive to aberrant data is dependent on the nature of the corruption. The definition of a breakdown point, however, states that if a regression model cannot accommodate some percentage of aberrant data regardless of the nature of the corruption, the model has reached its breakdown point. Applying this definition to the classic regression model, the breakdown point is 1/m (m being the number of observations) because in some instances the classic regression model cannot detect a single outlier. Thus, as the number of observations approaches infinity, the breakdown point approaches 0%. The first attempts to develop robust regression techniques identified cases which contained outlying dependent variables. The basis for these techniques was to make the influence of a observation a function of its corresponding residual. If a observation had a large residual in the classic regression solution, it received

a reduced weight in the objective function. These techniques did not improve the breakdown point of classic regression, because they did not address the problem of aberrant leverage points.

The next advance in the field of robust regression came as the result of weighting schemes based not only on the residuals of a the observations with respect to a classic regression solution, but also based on their position in n space relative to the other observations (n being the number of independent variables). This scheme was accomplished in two steps. First, an estimate of the center of the independent variable space spanned by all of the observations was obtained. Second, a weight was assigned to each observation based on its distances from the computed center point. Observations closer to the center point received higher weight than those observations which were far from the center point. Since these techniques could accommodate aberrant leverage points, they were the first techniques to exceed the 1/m breakdown point. The problem with these techniques was that the breakdown point was a function of the number of independent variables, not the the number of observations under consideration. As a result, the breakdown point approached 0% as the number of observations under consideration became large.

Rousseeuw and Leroy conclude their discussion of the history of robust estimation by introducing the two robust methods implemented in this project: least trimmed squares and least median squares. Both of these methods have breakdown points that approach 50% as the collection of observations becomes large and both methods can be applied to linear transformations of the independent variables.

The objective of the least trimmed squares technique is to find the h observations that will produce the minimum sum of squared residuals, where h is a subset of the m available observations. The least trimmed squares technique implements a method that will converge to a local minimum when applied to a combination of h observations. By using this method, the least trimmed squares solution can generally be found with less computational effort than the exact least median squares solution. The drawback to this approach is that the optimal choice for h is dependent on the number of aberrant observations. Since the goal of robust regression is to identify aberrant observations, it is difficult to select an appropriate h. Because the optimum value for h is not known, the course that is generally taken is to compute least trimmed results for a range of h values. Another

disadvantage of the least trimmed squares technique is that the procedure used to find the local minimum may not identify the global minimum. The only way to be certain that the best set of h observations has been chosen is to evaluate every h combination of m observations.

The objective of the exact least median squares is to minimize the median of the squared residuals. This algorithm is computationally intensive, but in contrast to the least trimmed squares, the optimal parameters of the objective function are not dependent on the amount of aberrant data.

Rousseeuw and Leroy give a description of a software package called PROGRESS (which derives its name from Program for RObust reGRESSion), which is a robust regression analysis tool. PROGRESS results can be used to test statistical hypothesis concerning robust regression coefficients and provide a plot of the data when there is a single independent variable.

To illustrate the application of robust techniques, Rousseeuw and Leroy apply PROGRESS to specific cases. The robust results obtained using PROGRESS are compared to results obtained using classic regression. The comparisons show that in addition to corrupting the estimates of regression coefficients,

aberrant data can corrupt the estimate of the statistical properties of the coefficients. This corruption can be so severe that the coefficients for the non-aberrant data can actually lie outside of the confidence interval established by the classic regression model.

Analysis of regression data is not limited to determining the regression coefficients. It is often useful to analyze the independent variables to determine if a linear relationship exists between two or more of those variables. This analysis is useful in determining whether a single independent variable can be changed to effect a change on the dependent variable. If several pairs of independent variables in the model have high correlation coefficients, it becomes impossible to isolate the relationship between a single independent variable and the dependent variable. Although high correlations between sets of independent variables makes analysis of the regression model difficult, analysis is even more difficult when the estimate of correlation among the independent variables is corrupted by aberrant data. A single aberrant observation can significantly distort the value of the correlation coefficient between a pair of independent variables. This distortion may either mask a correlation of independent variables or

indicate is a high correlation when, in fact, none exists.

Rousseeuw and Leroy characterize the least trimmed squares and least median squares in terms of their properties and offer proofs of these properties. The most significant properties for the purpose of this project are the properties of the breakdown points for these techniques. It is shown that for both techniques, the breakdown point approaches 50% as the number of observations approaches infinity.

Although Rousseeuw and Leroy introduce the concept of a minimum volume ellipsoid, a better explanation of its definition and application can be found in Hawkins and Simonoff (1992). The purpose of the minimum volume ellipsoid is to characterize the independent variables in a regression model and help identify leverage points. The minimum volume ellipsoid is defined as a ellipsoid in n space that encompasses h independent observations in the minimum possible volume, where n is the number of independent variables in each observation. Although the software developed for this project does not compute the parameters for this ellipsoid, Hawkins and Simonoff suggest that analysis based on the distance of each observation from the center of the minimum volume ellipsoid can be useful in identifying leverage points.

Hawkins and Simonoff advocate using an approximate least median squares technique to reduce the amount of computations necessary for regression coefficients. The number of calculations necessary for the computation of the exact least median squares parameters is a function of the possible number of combinations of m items taken p at a time, where m is the number of observations in the regression model and p is the number of independent variables in each observation. For a case with 25 observations of 4 independent variables, the number of combinations needed to calculate the exact least median squares parameters exceeds six million. While mainframe computers can accomplish the task of doing the exact least median squares computations, very good approximations to these parameters can be obtained using fewer computations. The approximate least median squares technique is based on the premise that the regression coefficients and residuals for any non-aberrant subset of data will not differ substantially from the coefficients and residuals obtained using an exact least median squares technique. The collection of observations is sampled p observations at a time with the expectation that at least one of the samples will contain non-aberrant data. Rousseeuw and Leroy advocate estimating the percentage of observations

that are corrupted (based on a priori information) to help determine the number of samples which will be necessary to reasonably assure that at least one of the observation samples will not contain aberrant data. If many of the observations are thought to contain aberrant data, the number of samples needed will be large; if the data is thought to contain only a few aberrant observations, only a few samples will be needed.

The least median squares approximation technique developed by Hawkins and Simonoff is different from an exact median squares algorithm in another respect. Their approximate technique is based on the premise that the parameters of the least median squares results can be approximated by the linear solution to some p observations. Thus, even if every p combination of the m observations were considered, the model would only produce an approximate least median squares solution.

The least median squares technique can be thought of as a specific application of a general algorithm that minimizes the qth largest residual. In the case of least median squares, q happens to be m/2 so that half of the squared residuals will be greater than q and half will be less than q. Hawkins and Cook (1990) argue that the general solution for various values of q be computed and analyzed. If the regression coefficients vary

radically as the value of q is changed, it can be concluded that a linear regression model may not be adequate to characterize the data (assuming that the range of qs chosen does not include aberrant data). It should be noted that the time required to compute regression values for a range of qs is not much greater than computing the regression value for a single q. The bulk of the computer time used by the algorithm is devoted to computing the ordinary least squares solution to each set of p observations under consideration. Once this solution is obtained, it is compared with the last optimal solution for q. Since the time necessary for a comparison is negligible compared to the time required to perform an ordinary least squares solution, making a comparison for each value of q does not significantly increase the time needed to compute regression results for multiple values of q.

Hawkins and Simonoff indicate that an approximate least median squares algorithm can be significantly improved by adjusting the intercept value for each elemental solution. By adjusting each elemental solution, the exact median squares coefficients can be found in cases where there are two independent variables. Results produced by this algorithm were compared to results of PROGRESS which does not adjust

the intercept. In some cases, the adjustment of the intercept significantly reduced the value of the qth residual.

Stromberg (1992) outlines the exact least median squares algorithm that has been implemented in this project. The algorithm used to compute the exact least median squares is much like the one used to compute the approximate least median squares. The primary difference is that the approximate least median squares objective is to minimize the sum of the squares of the residuals for some subset of the data; while the objective of the exact least median squares algorithm is to minimize the maximum residual for some subset of data.

The exact least median squares algorithm is based on two theorems. The first theorem states that the solution that minimizes the maximum residual for all of the observations will also minimize the maximum residual to some combination of p observations. The second theorem provides a means of computing the solution for a p+1 set of observations which will minimize the maximum residual. Thus, by considering the regression solution that minimizes the maximum residual for every combination of p+1 points, we will eventually happen upon the the solution which will minimize our qth

largest residual. It is necessary to compute the solution for p+1 points rather than p points, because the solution to any p points will yield a residual of zero for all p observations, while a solution to p+1 observations will usually yield non-zero residuals for each observation.

First the matrix X is selected. X is an (p+1)xn matrix whose rows are sets of independent variables. The least squares solution θ_{LS} is then computed for X.

$$\hat{\theta}_{LS} = MY$$

where $M=(X^T X)^{-1} X^T$ and Y is the vector of dependent variables. The residual for each observation, $r_i(\theta_{LS})$, is then computed by multiplying the ith row of X by the vector $\hat{\theta}_{LS}$ and subtracting this result from the ith element of Y. These residuals are then used to compute epsilon:

$$\epsilon = \frac{\sum_{i=1}^{p+1} r_i^2(\theta_{LS})}{\sum_{i=1}^{p+1} |r_i(\theta_{LS})|}$$

S is computed as the p dimensional vector with each element $s_i = \mathrm{sgn}(r_i(\theta_{LS}))$. Finally, the regression coefficients which minimize the maximum residual θ_c is computed:

$$\theta_c = M(Y - \epsilon S)$$

This set of regression coefficients is evaluated to determine whether it reduces the qth largest residual for each q of interest. If the set does reduce the qth largest residual, it is stored as a potential solution. This process continues until each p+1 combinations of observations is considered.

Stromberg also advocates comparing the regression coefficients for various values of q. As in the approximate least median squares algorithm, the computation required for multiple qs is not much greater than for a single q. In addition to evaluating each combination of p+1 observations for each value of q, each solution can be analyzed to determine the impact of the ith observation on the final solution. This can be done by storing the best solution that is not based on data from observation i. This requires that each time a solution is developed for a subset of p observations, a comparison must be made to determine whether the best solution that did not contain observation i has been improved. This comparison must be done for each observation that was not included in the p observations under consideration. The implementation of this enhancement involves only comparison and storage of data and does not require any additional computation. Thus,

as was the case for evaluating multiple qs, this can be done without significantly impacting the amount of time necessary to reach a solution. The software that has been written for this project does compute the least median squares solution over a range of qs, but does not compute the best solution with the ith observation deleted. The software was limited in its scope to reduce its complexity. This limited complexity makes the software easier to use and modify.

Stromberg cites specific examples of the exact least median squares solutions differing significantly from the approximate technique presented by Hawkins and Simonoff. It is shown that in one case, the point that provides the basis for the approximate solution is considered an outlier in the exact median squares solution.

Hawkins (1992) describes the least trimmed squares technique used in this project. While considering each p combination of m observations required to obtain an exact least squares estimate may be difficult, considering h such combinations is infeasible even for large mainframes, where h is the number of non-trimmed observations. Unlike the exact least median squares solutions, combinations for multiple hs can only be accomplished by repeating the steps needed to arrive at

a solution for a single h. Since it is desirable to evaluate the least trimmed solution for many values of h, it is infeasible to accomplish a rigorous solution for each value of h.

The algorithm described by Hawkins finds a local minimum for the sum of the squared residuals for a set of h observations. Since there exist some combinations of h observations whose local minimum is also the global minimum, it is only necessary to find the minimum for one of these combinations to find the global minimum. Unfortunately, there is no way to determine what observations will lead to a global minimum nor is there any way to know that a global minimum has been achieved. Hawkins advocates starting with h observations selected at random and computing the local minimum for these observations. This process is then repeated several times, each time storing local minimum results. The global minimum is assumed to be the least local minimum.

The algorithm first determines the least sum of squares solution for the h observations selected randomly. This solution is then evaluated to determine the effect of exchanging one trimmed observation for one of the non-trimmed observations. The effect of the swap is measured in terms of the difference in the sum of squares of the residuals of the solution to the h cases.

It is possible to compute the effect of a swap without actually computing the regression solution for the swapped pair. Every possible pair of trimmed and non-trimmed cases are considered as candidates for a swap. After every possible swap has been considered, the best improvement in the solution is examined. If the best improvement that can be obtained by a swap of a trimmed case for a non-trimmed case leads to an increase in the sum of squares of the residuals, then a local minimum has been found. If, however, the best swap will reduce the sum of squares of the residuals, that swap is made. After the swap is made, the regression solution for the new combination of h observations is computed and evaluated for yet another swap. This process continues until a local minimum is reached. After each minimum is reached, another combination of h observations is randomly selected and the process repeats. The process concludes when a specified number of random starts have been attempted.

The algorithm is only feasible if there are few local minima. If the astronomical number of possible combinations of h observations have an astronomical number of minima, this algorithm is not of much use. Hawkins computes the global minimum for several observations by considering each h possible combinations

of m observations. He then compares this rigorous solution to the one achieved through sampling and local minima. This is done for 5 cases with m ranging from 16 to 28. In every case, it is shown that if 100 random starts are considered, the probability of considering a combination that will converge to a global minimum is greater than 95%. If these observations can be considered a representative sample of all observations, then considering 100 local minima is quite feasible.

The notation used in the description of the least trimmed squares algorithm below is consistent with the notation used in Hawkins from which it was derived. The notation used in the description of the algorithm for the exact least median squares is also consistent with its source. The reader should be aware that these papers differ in their notation, particularly in the symbol used to describe the value of the ith residual.

X_J is an hxn matrix whose rows are the h independent variables of the observations under consideration. The least sum of squares solution for the h observations is computed. This solution is used to determine the residuals for all observations including the observations that were trimmed. These residuals, e_{iJ}, will be used to compute delta S (The notation for the residuals was changed from $r_i(\theta_{LS})$ to

e_{rJ} so that the notation for both the LTS and exact LMS would be consistent with the works from which they were derived). Let x_r be a 1xn vector of the independent variables for the rth observation. We form an mxm matrix H and designate the element in the rth row and sth column as h_{rs}. For each combination of r (which take on values from 1 to m) and s (which take on values from 1 to m) we compute h_{rs}:

$$h_{rs} = x_r(X_J^T X_J)^{-1} x_s^T$$

We compute delta S (the change in the sum of the squares of the residuals) that would occur by exchanging a non-trimmed observation (set j) for a trimmed observation (set i):

$$\Delta S_J = \frac{e_{jJ}^2(1-h_{ii}) - e_{iJ}^2(1+h_{jj}) + 2e_{iJ}e_{jJ}h_{ij}}{(1-h_{ii})(1+h_{jj}) + h_{ij}^2}$$

The value of delta S is computed for all possible combinations of i and j (There are m-h observations in set i and h observations in set j). If the least delta S is less than 0, the swap associated with the least delta S will bring us closer to the local minimum. In this case, X_{J+1} is formed by the making the exchange associated with the smallest value of delta S, and the

process is repeated. If all delta Ss are greater than or equal to zero, a local minimum has been found and another random start is chosen.

METHODS

The software that has been developed for this project was compiled using the Microsoft QuickC (version 2.5) compiler. This compiler transforms computer code written in C to machine code that can be run on Intel 8086 series (IBM compatible) microcomputers. Because the software written for this project used the full non-standard graphic capabilities of this compiler, a moderate amount of modification would be necessary to compile the code using some other compiler. Since the compiler costs less than $80, anyone wishing to modify the code should consider buying the Microsoft compiler.

The remainder of this section is devoted to the description of the implementation of the algorithms used in this project. It is broken into two subsections; one for each technique. Each technique is described in terms of the inputs, process, and outputs of the software.

Least Trimmed Squares

Inputs: The software reads inputs from both the command line and an input file. The command line input is used to designate what input file will be used, and the name of the output file. The software is designed to run under Microsoft DOS. At the DOS prompt, the user enters LTS path\filename, where path is the directory in

which the input file resides and filename is the filename of the input file with the extension omitted. The software assumes that the filename has an extension of .dat. Thus, to run the software using the file B:\SYSDAT\CASE1.DAT as the input file, the user would type LTS B:\SYSDAT\CASE1 at the DOS prompt. An example of a properly formatted least trimmed squares input file can be found in Appendix A. This file is divided into two parts. The first part of the file can be considered a run control; it contains parameters that control the execution of the software. The second portion of the input file contains the data. The data are separated from the input file by an indefinite number of header lines. After the run control is read, the first line that contains either a digit or a minus sign as the first non-blank character is considered a data line. Thus, all header information is ignored. Each subsequent line is read to determine whether it is a data line (using the criterion above). If it is not a data line, it is ignored. If the user wished to annotate a observation, he could do so on the preceding line by ensuring that the first character of the annotation was not a digit or a minus sign. Annotations must appear on a line of their own. Since observations are referenced by the order in which they appear in the

input file, it may be useful to precede each observation with "#nn", where the '#' serves to indicate that the line is not a data line (it is neither a digit nor a minus sign) and the nn is the number of the observation. The user should be cautioned that a row of minus signs cannot be used in the header. The software will interpret this line as a data line and will produce an error message indicating that it could not find the number of variables specified in the run control. The user is also cautioned that the lines used to annotate the data are ignored by the software. If the third observation is incorrectly labeled #4, the output will refer to that observation as observation 3. The data line is formatted so that the independent variables are on the left, and the dependent variables are to the right. The software will recognize any combination of spaces and tabs between variables.

Process: A flowchart of the process used in the least trimmed squares software can be found in Figure 3. The source code can be found in Appendix B.

Output: There are three types of output that are produced by the software: run-time output, post-processing screen graphics, and output written to a file. The run-time output is designed to keep the user informed of the progress of the software. It lets the

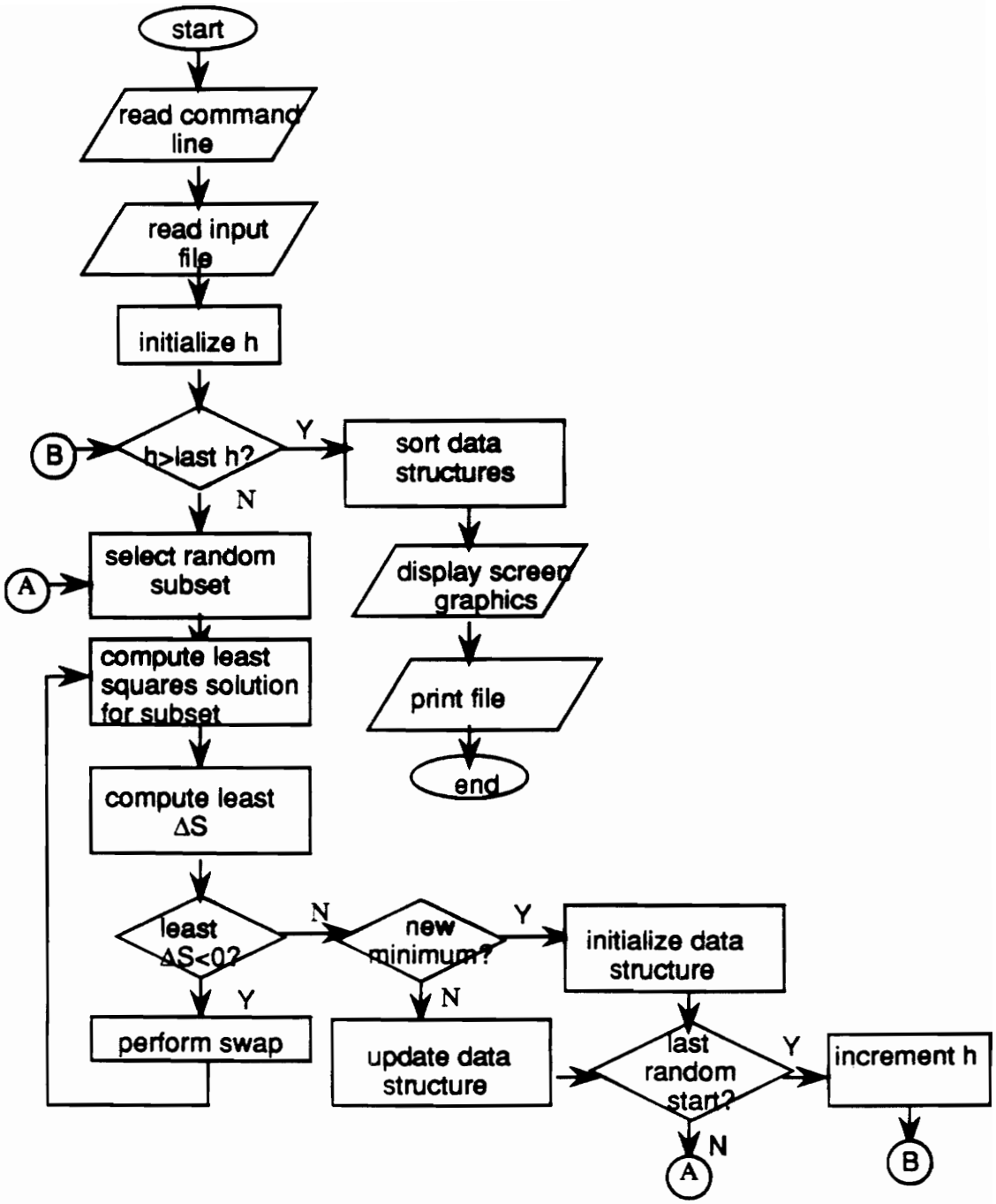


Figure 3
Flowchart for Least Trimmed Squares

user know the current value of h and how many random starts for that value of h have been processed. The purpose of the screen graphics is to show how the regression solution changed as the value of h changed. Each graphic is a line chart that uses a parameter of interest as the y axis and the value of h as the x axis. The parameters that are graphed as a function of h are: the sum of the squared residuals, the median residual, the sum of the squared standardized residuals, the median standardized residual, and the value for each of the regression coefficients. These graphs are useful in making subjective judgments such as how many of the observations are aberrant and which coefficients are unstable over the range chosen for h. A discussion of standardized residuals can be found in Seaver and Triantis. An example output file can be found in Appendix C. This file corresponds to the input file found in Appendix A. Output is produced for a user specified number of local minima for each value of h. These outputs are sorted by ascending values of h. For each value of h, the data associated with each minimum are sorted in ascending order of the value of the objective function. The residuals for each solution are listed in the ascending order of their squares. The case numbers that appear in the list of residuals, refer

to the order in which the observations were arranged in the input file. The beta vector contains the robust estimation of the regression coefficients which correspond to the order of the independent variables as they appear in the input file. If the user requested that an intercept be computed, this intercept value will precede the other regression coefficients. The output file will be written to the same directory and have the same filename as the input file. The output file will differ only in its extension. The output file will have a ".OUT" extension rather than the ".DAT" extension used for the input file.

Exact Median Squares

Inputs: The software reads inputs from both the command line and an input file. The command line input is used to designate what input file will be used, and the name of the output file. The software is designed to run under Microsoft DOS. At the DOS prompt, the user enters LMS path\filename, where path is the directory in which the input file resides and filename is the filename of the input file with the extension omitted. The software assumes that the filename has an extension of .DAT. Thus, to run the software using the file B:\SYSDAT\CASE1.DAT as the input file, the user would type LMS B:\SYSDAT\CASE1 at the DOS prompt. An example

of a properly formatted least median squares input file can be found in Appendix D. The input file is divided into two parts. The first part of the file can be considered a run control; it contains parameters that control the execution of the software. The second portion of the input file contains the data. The data are separated from the input file by an indefinite number of header lines. After the run control is read, the first line that contains either a digit or a minus sign as the first non-blank character is considered a data line. Thus, all header information is ignored. Each line is read to determine whether it is a data line (using the criterion above). All lines which are not data lines are ignored. If the user wishes to annotate a observation, he can do so by ensuring that the first character of the annotation is neither a digit nor a minus sign. Annotations must appear on a line of their own. Since the output file refers to observations in the order in which they appear, it may be useful to precede each observation with "#nn", where the '#' serves to indicate that the line is not a data line (it is neither a digit nor a minus sign) and the nn is the number of the observation. The user should be cautioned that a row of minus signs cannot be used in the header. The software will interpret this line as a data line and

will produce an error message indicating that it could not find the number of variables specified in the run control. The user is also cautioned that the lines used to annotate the data are ignored by the software. If the third observation is incorrectly labeled #4, the output will refer to that observation as observation 3. The data line is formatted so that the independent variables are on the left and the dependent variables are to the right. The software will recognize any combination of spaces and tabs between variables.

Process: A flowchart of the process used in the least median squares software can be found in Figure 4. The source code can be found in Appendix E.

Output: There are three types of output that are produced by the software: run-time output, post-processing screen graphics, and output written to a file. The run-time output is designed to keep the user informed of the progress of the software. It shows the user a bit pattern with each bit representing a observation. Each 1 in the bit pattern indicates that the corresponding observation is under consideration for an elemental solution. Each time that a new combination of observations is considered, the bit mask is updated. Processing ends when all combinations of observations have been considered. The purpose of the screen

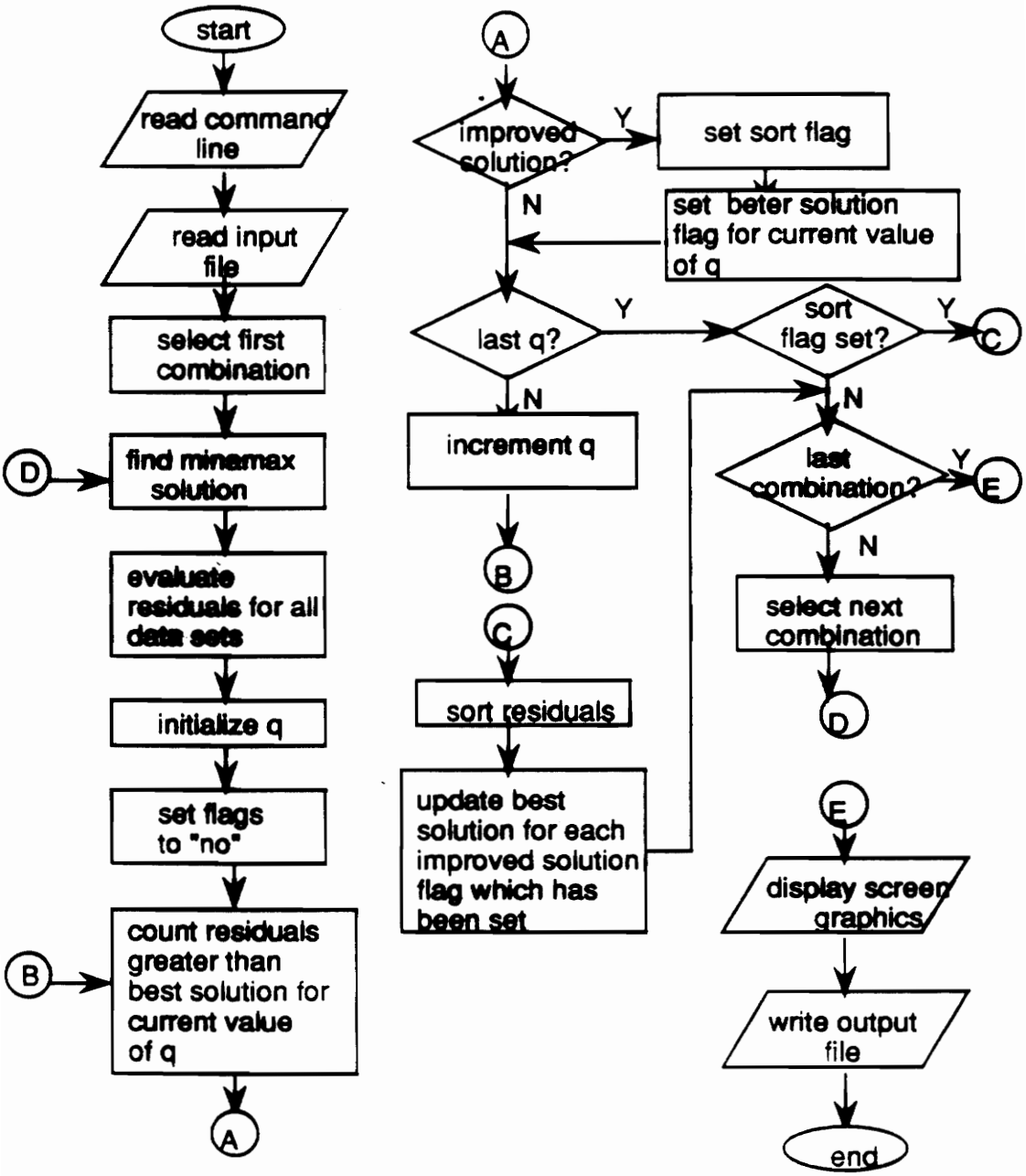


Figure 4
Flowchart for Exact Least Median Squares

graphics is show how the regression solution changed as he value of q changed. Each graphic is a line chart that uses a parameter of interest as the y axis and the value of q as the x axis. The parameters that are graphed as a function of q are: the sum of the squared residuals, the median residual, the sum of the squared standardized residuals, the median standardized residual, and the value for each of the regression coefficients. These graphs are useful in making subjective judgments such as how many of the observations are aberrant and which coefficients are unstable over the range chosen for q. A discussion of standardized residuals can be found in Seaver and Triantis (1993). An example output file can be found in Appendix F. This file corresponds to the input file found in Appendix D. Output is produced for each value of q. These outputs are sorted by ascending values of q. The residuals for each solution are listed in the ascending order of their squares. The case numbers that appear in the list of residuals refer to the order in which the observations were arranged in the input file. The theta vector contains the robust estimation of the regression coefficients which correspond to the order of the independent variables as they appear in the input file. If the user requested that an intercept be

computed, this intercept value will precede the other regression coefficients. The output file will be written to the same directory and will have the same filename as the input file. The output file will differ from the input file only in its extension. The output file will have a ".OUT" extension rather the ".DAT" extension used for the input file.

RESULTS

Tables 1 through 4 contain a comparison of the regression coefficients generated by the software developed in this project to those generated by mainframe implementations. In every case tested, the results obtained using the software developed for this project were identical to those obtained using the mainframe implementations. Case 1 contains 12 observations while Case 2 contains 25 observations. In both cases, the intercept was computed. The results for the intercept are listed as the 0th coefficient. It should be noted that the least trimmed squares software arrives at its solution much quicker than the exact least median squares software.

Table 1

Least Trimmed Squares Results: Case 1

h	Coefficient	Project Result	Mainframe Result
11	0	5.079	5.079
	1	0.177	0.177
	2	0.763	0.763
10	0	5.444	5.444
	1	0.299	0.299
	2	0.539	0.539
9	0	4.934	4.934
	1	0.954	0.954
	2	-0.090	-0.090
8	0	4.899	4.899
	1	0.951	0.951
	2	-0.081	-0.081

Table 2

Least Trimmed Squares Results: Case 2

h	Coefficient	Project Result	Mainframe Result
1	0	-2.65	-2.65
	1	0.0988	0.0988
	2	0.679	0.679
	3	0.256	0.256
	4	0.224	0.224
2	0	-2.87	-2.87
	1	0.103	0.103
	2	0.716	0.716
	3	0.236	0.236
	4	0.183	0.183
3	0	-3.58	-3.58
	1	0.0641	0.0641
	2	0.749	0.749
	3	0.213	0.213
	4	0.308	0.308
4	0	-2.66	-2.66
	1	0.129	0.129
	2	0.694	0.694
	3	0.229	0.229
	4	0.172	0.172

· Table 3

Exact Least Median Squares Results: Case 1

q	Coefficient	Project Result	Mainframe Result
6	0	4.745	4.745
	1	0.904	0.904
	2	0.001	0.001
7	0	4.741	4.741
	1	0.906	0.906
	2	0.000	0.000
8	0	5.175	5.175
	1	1.027	1.027
	2	-0.217	-0.217

Table 4

Exact Least Median Squares Results: Case 2

q	Coefficient	Project Result	Mainframe Result
13	0	0.811	0.811
	1	-0.0552	-0.0552
	2	0.250	0.250
	3	0.697	0.697
	4	0.418	0.418
14	0	1.00	1.00
	1	-0.0728	-0.0728
	2	0.220	0.220
	3	0.732	0.732
	4	0.447	0.447
15	0	0.961	0.961
	1	-0.0651	-0.0651
	2	0.228	0.228
	3	0.733	0.733
	4	0.422	0.422
16	0	1.28	1.28
	1	-0.0581	-0.0581
	2	0.187	0.187
	3	0.752	0.752
	4	0.437	0.437

CONCLUSIONS

This project achieved its stated purpose: to provide the systems engineer with a microcomputer based tool that will aid in the identification of aberrant system behavior. The first step in accomplishing this objective was to research the subject of robust regression. While some of the most advanced robust regression techniques were implemented on computer mainframes, these advanced techniques were not implemented on microcomputers. This project provides the only known microcomputer implementation of these techniques. Because these techniques are now implemented on a microcomputer, they are available to all engineers which have access to a microcomputer. When instances of aberrant system behavior are identified, these instances can be disregarded in modeling true system behavior and thus improve the accuracy of the system model.

The adaptation of the algorithms presented in the literature was straightforward. It was possible to include both algorithms within the scope of the project because very little effort was necessary to convert the detailed descriptions found in the literature to computer code. Although the source code that was implemented on the mainframe was available, its use as a

tool for developing the microcomputer based applications was quickly abandoned. In contrast to the descriptions found in the literature, the mainframe code was not well documented. The mainframe code was written in FORTRAN, and while there are FORTRAN compilers for microcomputers, C compilers are more prevalent. It would have been possible to accomplish a line by line translation of the FORTRAN code into C; however, since the purpose of each line of FORTRAN code was not always apparent, the translated C code would have suffered from the same lack of documentation as the FORTRAN implementations. Debugging the translated C code would have been difficult, since intermediate dumps of the FORTRAN implementation were not available. The code that was developed for this project provided capabilities beyond what was implemented in the mainframe versions. It required less effort to design the additional capabilities into the software than would have been required to retrofit translated C code.

The largest shortfall of the software, from a user's perspective, is that the post-processing screen graphics are available only immediately after the software is run and cannot be replayed without rerunning the software. The exact least median squares algorithm took over 3 hours to produce results. If the user

wanted to graphically display the results of the robust analysis at some time in the future, it would be necessary to execute the software again and tie up a microcomputer for 3 hours. A preferable solution would be to write the information needed to produce the graphics into a file and write separate software to read this file and produce the graphics. Another solution would be to format the graphic results for output on a hardcopy device.

RECOMMENDATIONS

From an academic perspective the purpose of this project is to provide a tool that can be used in further research. Some recommended topics for further research are included below:

1. The current algorithm for the least trimmed squares selects initial combinations of observations in a random fashion. Hawkins has shown that this method will usually converge to the best solution with few initial starts. The probability of converging to the optimum solution may be improved by changing the selection process of the initial observations from a random to a systematic process.

2. Although this paper contains sufficient detail to use the tools developed for this project, it does little to aid the systems engineer in interpreting their results. A user's manual for these tools would be useful. Such a manual should contain guidelines on how to interpret the results of the tools. The best way to accomplish this objective would be to create a simulated data set with known aberrant observations. This simulated data could be used to define the signature of aberrant data and test the sensitivity of these tools.

3. It may be possible to reduce the amount of computational effort required by each of the tools

developed for this project. The amount of computations required by the least trimmed squares algorithm could be reduced if some estimate of the number of starts needed were available. This estimate could either be based on the first few minima achieved by the tool or as a result of an assessment of the characteristics of the observations used in the model. The amount of computations required by the exact median squares algorithm may be reduced by using an approximate solution. Although some approximate least median squares algorithms exist, they are based on elemental solutions with zero of freedom. An approximate solution based on elemental solutions with one degree of freedom may provide an improved technique. The swap assessment used in least trimmed squares may improve an approximate least median squares algorithm based on elemental solutions with one degree of freedom.

4. It would be useful to do hypothesis testing in a statistical sense. The tools in their current form can aid in determining which data is aberrant; however, any such determination is based on subjective judgement. Although the confidence level for hypothesis testing is chosen in a subjective manner, the probability of making an error in identifying an observation as aberrant can be quantified.

5. One method of modeling non-linear systems is to use repeated linear approximations of these systems to converge on a non-linear solution. It would be useful to expand robust linear regression techniques into robust non-linear techniques.

References

Cook, R.D. and Hawkins, D.M. (1990), "Outliers Everywhere" - Comment on "Unmasking Multivariate Outliers and Leverage Points," Journal of the American Statistical Association, 85, 640-644.

Hawkins, D.M. (1992), "The Feasible Solution Algorithm for Least Trimmed Squares Regression," University of Minnesota, Department of Applied Statistics, working paper.

Hawkins, D. M. and Simonoff J.S. (1992), "High Breakdown Regression and Multivariate Estimation," The American Statistician.

Rousseeuw, P.J. and Leroy, A. (1987), Robust Regression and Outlier Detection, New York: John Wiley and Sons.

Seaver, B.L. and Triantis, K.P (1994), "The Impact of Outliers and Leverage Points on Technical Efficiency Measurements Using Robust Procedures", To be published in The American Journal of Management Science

Stark, H. and Woods, J.W (1986), Probability, Random Processes and Estimation Theory for Engineers, Englewood Cliffs, New Jersey: Prentice-Hall

Stromberg, A.J. (1992), "Computing the Exact Value of the Least Median of Squares Estimate in Multiple Linear Regression," University of Minnesota, Department of Applied Statistics, working paper #329.

APPENDIX A

LEAST TRIMMED SQUARES INPUT FILE

```
Number of Cases: 12
Number of Independent variables: 2
Use of intercept (y or n): y
Start trim: 1
End trim: 5
Number Tracked: 1
Number Starts: 10
Display CRT graphics: y

INDEP      INDEP           DEP
VAR 1      VAR 2           VAR
==============================
#1
5.593      5.045           9.806
#2
5.697      5.191           9.900
#3
5.245      4.830           9.4911
#4
5.343      5.507           10.580
#5
5.380      5.100           10.370
#6
5.325      4.380           9.658
#7
5.252      4.393           9.497
#8
5.098      4.676           9.358
#9
5.615      5.313           9.826
#10
6.296      4.386           9.443
#11
5.461      4.792           9.781
#12
6.369      5.387           10.509
```

APPENDIX B

LEAST TRIMMED SQUARES SOURCE CODE

```
/* This program is an adaption of an algorithm written by
   Douglas M. Hawkins.  The copyright for the algorithm belongs
   to him */

#include <conio.h>
#include <stdlib.h>
#include <pgchart.h>
#include <math.h>
#include <malloc.h>
#include <stdio.h>
#include <graph.h>
#include <string.h>
#define INDEP(AR1,AR2) (*(indep+(AR1)*num_idep+(AR2)))
#define DEP(AR1) (*(dep+(AR1)))
#define XTX(AR1,AR2) (*(xtx+(AR1)*num_idep+(AR2)))
#define XTXI(AR1,AR2) (*(xtxi+(AR1)*num_idep+(AR2)))
#define H(AR1,AR2) (*(h+(AR1)*num_cases+(AR2)))
#define XTXIXT(AR1,AR2) (*(xtxixt+(AR1)*tc+(AR2)))
#define XRXTXI(AR1,AR2) (*(xrxtxi+(AR1)*num_idep+(AR2)))
#define BETA(AR1) (*(beta+(AR1)))
#define X(AR1,AR2) (*(x+(AR1)*num_idep+(AR2)))
#define PRINT_SOL(AR1) (*(print_sol+(AR1)))
#define E(AR1) (*(e+(AR1)))
#define SS(AR1) (*(ss+(AR1)))
#define STD_MEDS(AR1) (*(std_meds+(AR1)))
#define STD_SS(AR1) (*(std_ss+(AR1)))
#define MEDS(AR1) (*(meds+(AR1)))
#define C_NUM(AR1) (*(c_num+(AR1)))
#define RES_SORT(AR1,AR2) (((AR1)->res_sort+(AR2)))
#define SQ(AR1) ((AR1)*(AR1))

#define Y(AR1) (*(y+(AR1)))
#define TRIM_LAB(AR1) (*(trim_lab+(AR1)))
#define FIRST_SOL(AR1) (*(first_sol+(AR1)))
#define S_LAB(AR1) (*(s_lab+(AR1)))
#define G_BETA(AR1,AR2) (*(g_beta+(AR1)*(end_trim-start_trim+1)+(AR2)))
   typedef struct res_struct
    {
       double res_v;
       int res_n;
    } res_T;

   typedef struct sol_struct
    {
       struct sol_struct *next_sol_P;
       int tot_swaps;
       int tot_sol;
       double sum_sqrs;
       char *sol_mask;
       double *beta_vec;
```

```
        res_T  *res_sort;
        double std_factor;
        double med;
        double ss;
    } sol_T;
char ibuffer [256];      /* Buffer to store data read from file */
void get_std(sol_T *sol, int num_cases);
int c_rs(res_T *arg1, res_T *arg2);
int comp_dbl(double *arg1, double *arg2);
char *get_valid(FILE *fp_input);
int comp_sol(sol_T **arg1, sol_T **arg2);
char bitchk(char *trim_mask, int pos_chk);
void setbit(char *trim_mask, int bit_num);
int c_r(res_T *arg1, res_T *arg2);

void invert(double *out_mat,double *in_mat,int mat_size);
void flip_bit(char *trim_mask, int pos_clear);
main(int num_args, char *args[2], char **senv)

/* This program accepts the input file name on
   the command line.  The file is assumed to have an extension of ".dat"
   The output from this program will be written to a file with a ".out"
   extension.  The output file will have the same name as the input file,
   and will be written in the same directory.  An example of what is expected
   on the command line is "B:\test" */

{
  sol_T **first_sol, *next_sol, **print_sol, *current_sol, *last_sol;

  char *file_name;        /* Input and Output filename */
  FILE *input_fp;         /* File pointer to input file */
  FILE *output_fp;        /* File pointer to output file */
  int  num_cases, num_idep, dep_start, num_track, num_start, cur_case;
  int  start, m_count, num_trim, exclude, row, col, offset, index;
  int  idep_start, idep_count, xclude, start_trim, end_trim, *c_num, t_index;
  int  tc, bit_search, swaps, trim_search, swap_out, swap_in, num_sols;
  char int_flag, dif_flag;
  char buffer[256], *tst_str;
  double *indep, *dep, *xtx, *xtxi, *xtxixt, *beta, *x, *y, *e, *h, delta_s;
  double *xrxtxi, big_ds, e_sum, ei2, ei2, eiej, std_f, t_sum;
  float  *g_beta;
  float  *ss, *std_ss, *std_meds, *meds;
  char *trim_mask, *swap_mask, do_g;
  chartenv env;
  char _far *_far *trim_lab, _far *_far *s_lab;
  res_T *cur_res;
  srand(2715);
  if(num_args!=2)  /* The user did not enter the correct number of arguments */
  {
     printf("This program expects to see an input file name on the command line.\n");
     printf("The format for this input file is path\\filename without an extension.\n");
```

```
    printf("An extension of .dat is assumed.\n");
    printf("The output file will be written to the same directory as the input file.\n");
    printf("The output file will have an .out extension and will overwrite\n");
    printf("whatever was in filename.out file before the program was run.");
    exit(0);
  }
  file_name=malloc(sizeof(char)*80);
  file_name=strcpy(file_name,args[1]);
  file_name=strcat(file_name,".dat");

  input_fp=fopen(file_name,"rt");/* Open file for reading */

  if(input_fp==NULL) /* There was some problem opening the file */
  {
     printf("Problem opening %s.dat for input",file_name);
     exit(0);
  }
/* Read header information from the file */
  num_cases=atoi(get_valid(input_fp));
  num_idep=atoi(get_valid(input_fp));
  int_flag=*(strtok(get_valid(input_fp)," \n"));
  if(int_flag == 'y' || int_flag == 'Y')  /*  we are to compute an intercept */
     idep_start=1;
  else
     idep_start=0;
  num_idep+=idep_start;
  start_trim=atoi(get_valid(input_fp));
  end_trim=atoi(get_valid(input_fp));
  num_track=atoi(get_valid(input_fp));
  num_start=atoi(get_valid(input_fp));
  do_g=*(strtok(get_valid(input_fp)," \n"));


  tc=num_cases-start_trim;
  indep=malloc(sizeof(double)*num_cases*num_idep);
  dep=malloc(sizeof(double)*num_cases);
  c_num=malloc(sizeof(int)*num_cases);

  for (cur_case=0;cur_case<num_cases;cur_case++)
  {
     C_NUM(cur_case)=cur_case+1;
     do
     {
  if(feof(input_fp)) /* There was not enough data */
  {
     printf("Insufficient data: only %d cases found %d expected\n",cur_case,num_cases);
     printf("Program terminating\n");
     exit(0);
  }
  fgets(buffer,256,input_fp);
```

```
tst_str=strtok(buffer,"\t\n ");
if(tst_str=='\0') tst_str=buffer;
    }while ((*(tst_str) < '0' || *(tst_str) > '9') && *(tst_str)!='-');

    if (idep_start==1) INDEP(cur_case,0)=1;
    for(idep_count=idep_start;idep_count<num_idep;idep_count++)
    {
if (tst_str==NULL) /* There is insufficient data */
{
   printf("data value missing in line %d\n",cur_case+1);
   printf("program terminating\n");
   exit(0);
}
INDEP(cur_case,idep_count)=atof(tst_str);
tst_str=strtok(NULL," \t\n");
    }
    if (tst_str==NULL) /* There is insufficient data */
    {
 printf("data value missing in line %d\n",cur_case+1);
 printf("program terminating\n");
 exit(0);
     }

     DEP(cur_case)=atof(tst_str);
 }
 fclose(input_fp);
 trim_lab=malloc(sizeof(char _far * far)*(end_trim-start_trim+1));
 g_beta=malloc(sizeof(float)*(end_trim-start_trim+1));
 swap_mask=malloc(sizeof(char)*num_cases);
 trim_mask=malloc(sizeof(char)*num_cases);
 xtx=malloc(sizeof(double)*num_idep*num_idep);
 xtxi=malloc(sizeof(double)*num_idep*num_idep);
 xtxixt=malloc(sizeof(double)*num_idep*tc);
 x=malloc(sizeof(double)*tc*tc);
 beta=malloc(sizeof(double)*num_cases);
 y=malloc(sizeof(double)*tc);
 e=malloc(sizeof(double)*num_cases);
 h=malloc(sizeof(double)*num_cases*num_cases);
 xrxtxi=malloc(sizeof(double)*num_idep*num_cases);
 ss=malloc(sizeof(float)*(end_trim-start_trim+1));
 std_ss=malloc(sizeof(float)*(end_trim-start_trim+1));
 std_meds=malloc(sizeof(float)*(end_trim-start_trim+1));
 meds=malloc(sizeof(float)*(end_trim-start_trim+1));
 first_sol=malloc(sizeof(sol_T *)*end_trim-start_trim+1);


 for(num_trim=start_trim;num_trim<=end_trim;num_trim++)
 {
 FIRST_SOL(num_trim-start_trim)=NULL;
 t_index=num_trim-start_trim;
```

```
   tc=num_cases-num_trim;

TRIM_LAB(num_trim-start_trim)= fmalloc(sprintf(buffer,"%d",(int)((float)num_trim/(float)num_cases*100))+1);
    _fstrcpy(TRIM_LAB(num_trim-start_trim),(char _far *)buffer);

   printf("\n\nThe number trimmed is %d\n",num_trim);
   for(start=0;start<num_start;start++)
   {
   printf("This is start %d\n",start+1);
/* Initialize the bit mask */
      for(m_count=0;m_count<num_cases;m_count++)
      {
  *(trim_mask+m_count)='0';
      }
/*  Pick out cases to be trimmed */
      for(cur_case=0;cur_case<num_trim;cur_case++)
      {
      do
  {
     xclude=rand()%num_cases;
  } while (bitchk(trim_mask,xclude)=='1');
      setbit(trim_mask,xclude);
      }
      swaps=0;
/* The do loop below will find OLS solutions and perform swaps until no
   improvement is realized. */

      do
      {
/* Compute an OLS solution for the cases that were not trimmed */
/* Initalize matrixs */
  for(offset=0;offset<num_idep*num_idep;*(xtx+offset++)=0);
  for(offset=0;offset<num_idep;*(beta+offset++)=0);
  for(offset=0;offset<num_cases;*(e+offset++)=0);
  for(offset=0;offset<num_cases*num_idep;*(xrxtxi+offset++)=0);
  for(offset=0;offset<num_cases*num_cases;*(h+offset++)=0);
  e_sum=0;
  t_sum=0;
  bit_search=-1;
  for(offset=0;offset<tc;offset++)
  {

     for(bit_search++;bitchk(trim_mask,bit_search)=='1';bit_search++);
     Y(offset)=DEP(bit_search);

     for(index=0;index<num_idep;index++)
     {
        XTXIXT(index,offset)=0;
        X(offset,index)=INDEP(bit_search,index);
     }
```

```
  }
/* Compute X transpose X  */
  for (row=0;row<num_idep;row++)
  {
     for (col=0;col<num_idep;col++)
     {
        for(index=0;index<tc;index++)
        {
         XTX(row,col)+=X(index,row)*X(index,col);
        }
     }
  }
  invert(xtxi,xtx,num_idep);
/* Compute xtxixt */
  for(row=0;row<num_idep;row++)
  {
     for(col=0;col<tc;col++)
     {
        for(index=0;index<num_idep;index++)
        {
        XTXIXT(row,col)+=XTXI(row,index)*X(col,index);
        }
     }
  }
/* Compute beta vector */
  for(row=0;row<num_idep;row++)
  {
     for(index=0;index<tc;index++)
     {
        BETA(row)+=xTXIXT(row,index)*Y(index);
     }
  }
/* Compute the E vector */
  for(row=0;row<num_cases;row++)
  {
     for(index=0;index<num_idep;index++)
     {
        E(row)+=INDEP(row,index)*BETA(index);
     }
     E(row)=E(row)-DEP(row);
     if(bitchk(trim_mask,row)=='0') e_sum+=E(row)*E(row);
     t_sum+=E(row)*E(row);
  }
/* Compute the H matrix */
  for(row=0;row<num_cases;row++)
  {
     for(col=0;col<num_idep;col++)
     {
        for(index=0;index<num_idep;index++)
        {
```

```
        XRXTXI[row,col]+=INDEP[row,index]*XTXI[col,index];
        }
    }
  }
  for(row=0;row<num_cases;row++)
  {
    for(col=0;col<num_cases;col++)
    {
      for(index=0;index<num_idep;index++)
      {
      H[row,col]+=XRXTXI[row,index]*INDEP[col,index];
      }
    }
  }
  bit_search=-1;
  big_ds=0;
  for(offset=0;offset<num_cases;*(swap_mask+offset++)='0');
/* Compute the effect of a swap for each case which was not chosen
   This effect is characterized by a quantity delta S sub ij which is the
   change that will occur in the sum of squares of the residuals if
   i (a case in the present set of unchanged cases) is exchanged for j
   (a case that was trimmed).  If delta S is negative, a swap would benefit us
*/
  for(offset=0;offset<tc;offset++)
  {
    for(bit_search++;bitchk(trim_mask,bit_search)=='1';bit_search++);
    trim_search=-1;
    for(index=0;index<num_trim;index++)
    {
      for(trim_search++;bitchk(trim_mask,trim_search)=='0';trim_search++);
      ej2=E[trim_search]*E[trim_search]*(1-H[bit_search,bit_search]);
      ei2=E[bit_search]*E[bit_search]*(1+H[trim_search,trim_search]);
      eiej=E[bit_search]*E[trim_search]*H[bit_search,trim_search];


      delta_s=(ej2-ei2+2*eiej)/
           ((1+H[trim_search,trim_search])*(1-H[bit_search,bit_search])+
           H[trim_search,bit_search]*H[trim_search,bit_search]);
      if(delta_s<big_ds) /*  The pair under consideration should be
                  swapped */
      {
      big_ds=delta_s;
      swap_in=trim_search;
      swap_out=bit_search;
      }
    }
  }
  if(big_ds<0) /* There is a benefitical swap */
  {
    flip_bit(trim_mask,swap_out);
    flip_bit(trim_mask,swap_in);
```

```
    swaps++;
  }

    } while(big_ds<0);
/*  When we have reached this portion of the code, we have a potential
    solution.  We check to see if this solution matches any of the solutions
    from any of the previous random starts.  We will know that it matches
    a previous solution if the trim mask is equal to a mask saved from a
    previous local solution */
    if (FIRST_SOL(num_trim-start_trim)==NULL) /* This is our first potential solution */
    {
  FIRST_SOL(t_index)=malloc(sizeof(sol_T));
  FIRST_SOL(t_index)->tot_swaps=swaps;
  FIRST_SOL(t_index)->tot_sol=1;
  FIRST_SOL(t_index)->ss=t_sum;
  FIRST_SOL(t_index)->sum_sqrs=e_sum;
  FIRST_SOL(t_index)->next_sol_P=NULL;
  FIRST_SOL(t_index)->sol_mask=malloc(sizeof(char)*num_cases);
  FIRST_SOL(t_index)->beta_vec=malloc(sizeof(double)*num_idep);
  FIRST_SOL(t_index)->res_sort=malloc(sizeof(res_T)*num_cases);
  for(index=0;index<num_cases;index++)
  {
      RES_SORT(FIRST_SOL(t_index),index)->res_v=E(index);
      RES_SORT(FIRST_SOL(t_index),index)->res_n=C_NUM(index);
  }
  get_std(*(first_sol+t_index),num_cases);


  for(index=0;index<num_idep;*((FIRST_SOL(t_index)->beta_vec)+index)=BETA(index),index++);

for(index=0;index<num_cases;*((FIRST_SOL(t_index)->sol_mask)+index)=*(trim_mask+index),index++);
  last_sol=FIRST_SOL(t_index);
      }
      else
      {
  for(next_sol=FIRST_SOL(t_index);next_sol!=NULL;next_sol=next_sol->next_sol_P)
  {
     dif_flag='f';
     for(index=0;index<num_cases;index++)
     {

      if(*((next_sol->sol_mask)+index)!=*(trim_mask+index))
/* This solution is different than the solution under consideration */
      {
         dif_flag='t';
         break;
      }
     }
     if (dif_flag=='f') /* This solution matches next_sol */
     {
/*  Update next_sol's values */
```

```
            next_sol->tot_swaps+=swaps;
            next_sol->tot_sol++;
            break;
         }
      }
      if(dif_flag=='t') /*  We have a new solution  */
      {
         last_sol->next_sol_P=malloc(sizeof(sol_T));
         current_sol=last_sol->next_sol_P;
         last_sol=current_sol;
         current_sol->tot_swaps=swaps;
         current_sol->tot_sol=1;
         current_sol->ss=t_sum;
         current_sol->sum_sqrs=e_sum;
         current_sol->next_sol_P=NULL;
         current_sol->sol_mask=malloc(sizeof(char)*num_cases);
         current_sol->beta_vec=malloc(sizeof(double)*num_idep);
         current_sol->res_sort=malloc(sizeof(res_T)*num_cases);
         for(index=0;index<num_cases;index++)
         {
          RES_SORT(current_sol,index)->res_v=E(index);
          RES_SORT(current_sol,index)->res_n=C_NUM(index);
         }

         for(index=0;index<num_cases;*((current_sol->sol_mask)+index)=*(trim_mask+index),index++);
         for(index=0;index<num_idep;*((current_sol->beta_vec)+index)=BETA(index),index++);

      }
          }

       } /* This is the end of the random starts loop */
       } /* This is the end of the num_trim loop */
       file_name=strcpy(file_name,args[1]);
       file_name=strcat(file_name,".out");
       output_fp=fopen(file_name,"wt");/* Open file for writing */

       if(output_fp==NULL) /* There was some problem opening the file */
       {
          printf("Problem opening %s.out for output",file_name);
          exit(0);
       }


       for(num_trim=start_trim;num_trim<=end_trim;num_trim++)
       {
       t_index=num_trim-start_trim;

    /* Now we print the results */
    /* Initialize the print_sol array to be an array pointers to all of the
       solutions that we have generated.  We will sort this array */
       print_sol=malloc(sizeof(next_sol)*num_track);
```

```
   for(next_sol=FIRST_SOL(t_index),index=0;index<num_track &&
next_sol!=NULL;next_sol=next_sol->next_sol_P,index++)
   {
      *(print_sol+index)=next_sol;
   }
   num_sols=index;
   qsort(print_sol,num_sols,2,comp_sol);

   fprintf(output_fp,"\n\nNumber of cases trimmed: %d",num_trim);
   for(index=0;index<num_track && index<num_sols;index++)
   {
      get_std(PRINT_SOL(index),num_cases);

      std_f=PRINT_SOL(index)->std_factor;

      if(index==0) /* This is the best solution for the number trimmed */
      {
  SS(t_index)=(float)(PRINT_SOL(index)->ss);
  MEDS(t_index)=(float)(PRINT_SOL(index)->med);
  STD_SS(t_index)=(float)(PRINT_SOL(index)->ss/std_f/std_f);
  STD_MEDS(t_index)=(float)(PRINT_SOL(index)->med/std_f/std_f);
  for(offset=0;offset<num_idep;offset++)
  {
     G_BETA(offset,t_index)=(float)*((PRINT_SOL(index)->beta_vec)+offset);
  }
      }
      fprintf(output_fp,"\n\nSolution %d:\n",index+1);
      fprintf(output_fp,"Percent of starts leading to solution: %f \n",
      (PRINT_SOL(index)->tot_sol/(float)num_start)*100);

      fprintf(output_fp,"Average number of swaps to achieve solution: %f \n",
      (PRINT_SOL(index)->tot_swaps/(float)(PRINT_SOL(index)->tot_sol)));

      fprintf(output_fp,"Sum of squares of the residuals for the non trimmed cases: %f\n",
       (float)((PRINT_SOL(index)->sum_sqrs)));

      fprintf(output_fp,"Sum of a squares for all residuals (non-std, std): %lf, %lf\n",
       PRINT_SOL(index)->ss,PRINT_SOL(index)->ss/std_f/std_f);

      fprintf(output_fp,"Median squared residual (non-std, std): %lf, %lf\n",
       PRINT_SOL(index)->med,PRINT_SOL(index)->med/std_f/std_f);


      fprintf(output_fp,"Cases trimmed:\n");
      for(offset=0;offset<num_cases;offset++)
      {
  if(bitchk(PRINT_SOL(index)->sol_mask,offset)=='1')
/* We have a trimmed case */
  {
     fprintf(output_fp,"%d  ",offset+1);
  }
```

```
      }
      fprintf(output_fp,"\n\nBeta vector:\n");
      for(offset=0;offset<num_ideo;offset++)
      {
  fprintf(output_fp,"%lf\n",*((PRINT_SOL(index)->beta_vec)+offset));
      }
      fprintf(output_fp,"\n\nCase Numbers, Residuals, and Standard Residuals:\n");
      for(offset=0;offset<num_cases;offset++)
      {
  cur_res=(PRINT_SOL(index)->res_sort+offset);
  fprintf(output_fp,"%d,   %lf,
%lf\n",cur_res->res_n,cur_res->res_v,(cur_res->res_v)/(PRINT_SOL(index)->std_factor));
      }

    }

    } /* This is the end of the number trimmed loop */

    fcloseall();
     if (do_g=='n' || do_g=='N') exit(0);
     if( !_setvideomode( _MAXRESMODE ) )
  exit( 1 );

     _pg_initchart();                    /* Initialize chart system    */
     _pg_defaultchart( &env, _PG_LINECHART, _PG_POINTANDLINE );
     strcpy( env.maintitle.title, "SS For Non-Standard Residuals" );
     _pg_chart( &env, (char _far *_far *)trim_lab, (float _far *)ss, end_trim-start_trim+1);

     getch();
     _clearscreen( _GCLEARSCREEN );
     _pg_initchart();                    /* Initialize chart system    */
     _pg_defaultchart( &env, _PG_LINECHART, _PG_POINTANDLINE );

     strcpy( env.maintitle.title, "Median of Non-Standard Squared Residuals" );
     _pg_chart( &env, (char _far *_far *)trim_lab, (float _far *)meds, end_trim-start_trim+1);

     getch();
     _clearscreen( _GCLEARSCREEN );
     _pg_initchart();                    /* Initialize chart system    */
     _pg_defaultchart( &env, _PG_LINECHART, _PG_POINTANDLINE );

     strcpy( env.maintitle.title, "SS For Standard Residuals" );
     _pg_chart( &env, (char _far *_far *)trim_lab, (float _far *)std_ss, end_trim-start_trim+1);

     getch();
     _clearscreen( _GCLEARSCREEN );
     _pg_initchart();                    /* Initialize chart system    */
     _pg_defaultchart( &env, _PG_LINECHART, _PG_POINTANDLINE );

     strcpy( env.maintitle.title, "Median of Standard Squared Residuals" );
     _pg_chart( &env, (char _far *_far *)trim_lab, (float _far *)std_meds,
```

```
end_trim-start_trim+1);

    getch();
    _clearscreen( _GCLEARSCREEN );

    num_trim=end_trim-start_trim+1;
    for(index=0;index<num_idep;index++)
    {
       _pg_initchart();                    /* Initialize chart system    */
       _pg_defaultchart( &env, _PG_LINECHART, _PG_POINTANDLINE );

       sprintf(buffer,"Parameter %d",index+1-idep_start);
       strcpy( env.maintitle.title, buffer );

       _pg_chart( &env, (char _far * _far *)trim_lab, (float _far *)(g_beta+index*num_trim),
num_trim);
       getch();
    }
    _setvideomode( _DEFAULTMODE );

}
```

```
char *get_valid(FILE *input_fp)
/* This function reads records until it encounters an non blank record.
   This non-blank record is parsed so that the string to the right of
   the ":" is returned.

   fp_input pointer to the file that is to be read
   This function returns a pointer to a string with usable data. */

{


      do /* Until a valid record is reached */
      {
  fgets(ibuffer,256,input_fp);
      } while(strtok(ibuffer," \t\n")==NULL);
   (strtok(NULL,":"));
   return strtok(NULL,"\n");
}

char bitchk(char *trim_mask, int pos_chk)
```

```
{

   if (*(trim_mask+pos_chk)=='1') return '1';
   else return '0';
}
void setbit(char *trim_mask, int bit_num)
{
    *(trim_mask+bit_num)='1';
}
void invert(double *out_mat,double *in_mat,int mat_size)
{
/*  This fucntion inverts a matrix
    out_mat is a pointer to the output_matrix
    in_mat is a pointer to the input_matrix
    mat_size is the matrix size
*/
#define OUT_MAT(x,y) (*(out_mat+(x)*mat_size+(y)))
#define IN_MAT(x,y) (*(in_mat+(x)*mat_size+(y)))
#define WORK_MAT(x,y) (*(work_mat+(x)*mat_size+(y)))
   int row, col, index;
   double *work_mat, mult, div;
/* Initialize out_mat to the identity matrix
    and copy the input matrix so we do not corrupt it */
   work_mat=malloc(sizeof(double)*mat_size*mat_size);

   for(row=0;row<mat_size;row++)
   {
      for(col=0;col<mat_size;col++)
      {
  WORK_MAT(row,col)=IN_MAT(row,col);
  if (row==col) OUT_MAT(row,col)=1;
  else OUT_MAT(row,col)=0;
      }
   }
/* Invert the matrix */
   for (col=0;col<mat_size;col++)
   {
       div=WORK_MAT(col,col);
       for(index=0;index<mat_size;index++)
       {
   WORK_MAT(col,index)/=div;
   OUT_MAT(col,index)/=div;
       }
       for(row=0;row<mat_size;row++)
       {
  if(row==col) continue;
  else mult=WORK_MAT(row,col);
  for(index=0;index<mat_size;index++)
  {
      WORK_MAT(row,index)-=WORK_MAT(col,index)*mult;
      OUT_MAT(row,index)-=OUT_MAT(col,index)*mult;
```

```
   }
      }
   }
   free(work_mat);
}
void flip_bit(char *trim_mask, int pos_clear)
{
   if(*(trim_mask+pos_clear)=='1') *(trim_mask+pos_clear)='0';
   else *(trim_mask+pos_clear)='1';
}
int comp_sol(sol_T **arg1, sol_T **arg2)
{
/*  This function will be used by qsort to compare the solution pointed to
    by arg1 to by arg2.  We cannot simply subtract the sum of squares of the
    solution pointed to by arg2 from that of arg1.  When these values are
    both less than 1, the function will always return 0.  Thus we need all
    of the if statements */


    if (((*(arg1))->sum_sqrs-(*(arg2))->sum_sqrs)>0) return 1;
    if (((*(arg1))->sum_sqrs-(*(arg2))->sum_sqrs)==0) return 0;
    if (((*(arg1))->sum_sqrs-(*(arg2))->sum_sqrs)<0) return -1;




}
void get_std(sol_T *sol, int num_cases)
{
  /*  This rountine will assign values to the median sum of squares
      and the std_factor for the structure sol.
      The routine will also sort the residual array res_sort, by the value
      of the residual squared */
   int index;
   double *ss_sort, *s_vals, med_res;
   s_vals=malloc(sizeof(double)*num_cases);
/* This qsort uses c_r which compares the residuals.  The array is sorted by
   the value of the residuals and not the value of the residuals squared.
   The median residual is needed for the calcualtion of the standard factor */

   qsort(sol->res_sort,num_cases,sizeof(res_T),c_r);
   if(ceil(num_cases/2)==floor(num_cases/2))  /* There is an even number of
                          cases */

med_res=(double)((RES_SORT(sol,num_cases/2)->res_v+RES_SORT(sol,(num_cases/2)-1)->res_v)/2);
   else
   med_res=(double)((RES_SORT(sol,(int)(floor(num_cases/2)))->res_v));
/* This qsort uses c_rs which comapares the value of the residuals squared.
   The median value produced by this sort is the median of the squared
   residuals.  This sort is done after the sort of the unsquared residuals so
   that when it comes time to print the results, the residuals will be in the
```

```
   sorted by the square of their value */

   qsort(sol->res_sort,num_cases,sizeof(res_T),c_rs);
   if(ceil(num_cases/2)==floor(num_cases/2))  /* There is an even number of
                           cases */

sol->med=(double)(((SQ(RES_SORT(sol,num_cases/2)->res_v)+SQ(RES_SORT(sol,(num_cases/2)-1)->res_v)))/2);
   else
   sol->med=(double)SQ(((RES_SORT(sol,(int)(floor(num_cases/2)))->res_v)));


   for(index=0;index<num_cases;index++)
   {
      *(s_vals+index)=fabs(RES_SORT(sol,index)->res_v-med_res);
   }
   qsort(s_vals,num_cases,sizeof(double),comp_dbl);
   if(ceil(num_cases/2)==floor(num_cases/2))  /* There is an even number of
                           cases */
   sol->std_factor=(double)((*(s_vals+num_cases/2)+*(s_vals+(num_cases/2)-1))/2);
   else
   sol->std_factor=(double)(*(s_vals+num_cases/2));
   sol->std_factor=sol->std_factor/0.6475;
   free(s_vals);
}
int c_rs(res_T *arg1, res_T *arg2)
{
/*  This function will be used by qsort to compare residual structures.
    The structures are to be sorted by the values of the squared residuals.
    The structure contains the unsquared residuals.  Every time that this
    function does a compare, it must do it based on the square of the values
    in the res_T structure  */
    double tst_val;
    tst_val=(arg1->res_v*arg1->res_v)-(arg2->res_v*arg2->res_v);
    if(tst_val>0) return 1;
    if(tst_val==0) return 0;
    if(tst_val<0) return -1;
}

int comp_dbl(double *arg1, double *arg2)
{
/*  This function will compare two double values for a qsort function */
   double tst_val;
    tst_val=*arg1-*arg2;

    if(tst_val>0) return 1;
    if(tst_val==0) return 0;
    if(tst_val<0) return -1;
}


int c_r(res_T *arg1, res_T *arg2)
```

```
{
/*  This function will be used by qsort to compare residual structures.
    The structures are to be sorted by the values of the residuals.
    The structure contains the unsquared residuals.   */
    double tst_val;
    tst_val=(arg1->res_vi-(arg2->res_vi;
    if(tst_val>0) return 1;
    if(tst_val==0) return 0;
    if(tst_val<0) return -1;
}
```

APPENDIX C

LEAST TRIMMED SQUARES OUTPUT FILE

```
Number of cases trimmed: 1

Solution 1:
Percent of starts leading to solution: 100.000000
Average number of swaps to achieve solution: 1.000000
Sum of squares of the residuals for the non trimmed cases: 0.485091
Sum of a squares for all residuals (non-std, std): 0.684760, 8.152385
Median squared residual (non-std, std): 0.036786, 0.437952
Cases trimmed:
5

Beta vector:
5.079126
0.176773
0.763332


Case Numbers, Residuals, and Standard Residuals:
11,   -0.078630,   -0.271308
10,   0.097063,   0.334908
1,   0.112827,   0.389301
7,   -0.136145,   -0.469759
2,   0.148658,   0.512933
8,   0.191655,   0.661291
12,   -0.191938,   -0.662268
3,   0.202093,   0.697309
6,   -0.294164,   -1.014992
9,   0.301289,   1.039576
4,   -0.352707,   -1.216992
5,   -0.446843,   -1.541799


Number of cases trimmed: 2

Solution 1:
Percent of starts leading to solution: 100.000000
Average number of swaps to achieve solution: 1.700000
Sum of squares of the residuals for the non trimmed cases: 0.284664
Sum of a squares for all residuals (non-std, std): 0.928785, 10.971860
Median squared residual (non-std, std): 0.021929, 0.259045
Cases trimmed:
4  5

Beta vector:
5.443948
0.298907
0.539447
```

```
Case Numbers, Residuals, and Standard Residuals:
1,   0.031247,   0.107395
2,   0.047092,   0.161857
7,   -0.113400,   -0.389759
11,   -0.119689,   -0.411375
3,   0.126146,   0.433566
8,   0.132231,   0.454483
9,   0.162394,   0.558153
10,   0.248883,   0.855416
12,   -0.255311,   -0.877508
6,   -0.259593,   -0.892227
5,   -0.566751,   -1.947937
4,   -0.568256,   -1.953109


Number of cases trimmed: 3

Solution 1:
Percent of starts leading to solution: 80.000000
Average number of swaps to achieve solution: 2.625000
Sum of squares of the residuals for the non trimmed cases: 0.009873
Sum of a squares for all residuals (non-std, std): 2.897646, 1223.660786
Median squared residual (non-std, std): 0.000878, 0.370715
Cases trimmed:
4  5  10

Beta vector:
4.934469
0.954312
-0.090224


Case Numbers, Residuals, and Standard Residuals:
2,   0.002832,   0.058201
1,   0.010756,   0.221042
9,   -0.012429,   -0.255408
3,   0.012954,   0.266202
12,   0.017446,   0.358511
8,   0.019665,   0.404104
6,   -0.037000,   -0.760349
7,   0.053162,   1.092470
11,   -0.067386,   -1.384772
5,   -0.761474,   -15.648156
4,   -1.043505,   -21.443834
10,   1.104095,   22.688953


Number of cases trimmed: 4

Solution 1:
```

```
Percent of starts leading to solution: 70.000000
Average number of swaps to achieve solution: 2.714286
Sum of squares of the residuals for the non trimmed cases: 0.004709
Sum of a squares for all residuals (non-std, std): 2.876402, 1429.961707
Median squared residual (non-std, std): 0.000983, 0.488463
Cases trimmed:
4  5  10  11

Beta vector:
4.898808
0.950753
-0.080657


Case Numbers, Residuals, and Standard Residuals:
2,   -0.003440,   -0.076694
1,   0.003458,   0.077097
3,   0.004837,   0.107844
8,   0.010597,   0.236281
12,   0.010658,   0.237632
9,   -0.017242,   -0.384429
7,   0.040839,   0.910571
6,   -0.049707,   -1.108302
11,   -0.076635,   -1.708706
5,   -0.767489,   -17.112344
4,   -1.045494,   -23.310897
10,   1.087990,   24.258417


Number of cases trimmed: 5

Solution 1:
Percent of starts leading to solution: 80.000000
Average number of swaps to achieve solution: 3.125000
Sum of squares of the residuals for the non trimmed cases: 0.000000
Sum of a squares for all residuals (non-std, std): 2.590058, 6550110.497050
Median squared residual (non-std, std): 0.000000, 0.159661
Cases trimmed:
4  5  6  10  11

Beta vector:
4.740595
0.905501
0.000201


Case Numbers, Residuals, and Standard Residuals:
9,   0.000053,   0.084656
1,   0.000078,   0.124479
7,   0.000171,   0.272197
3,   -0.000179,   -0.285297
```

```
12,   -0.000184,   -0.292583
8,   -0.000219,   -0.348374
2,   0.000280,   0.444923
11,   -0.094499,   -150.278149
6,   -0.094730,   -150.645614
5,   -0.756782,   -1203.484772
10,   0.999513,   1589.490783
4,   -1.000204,   -1590.589748
```

APPENDIX D

EXACT LEAST MEDIAN SQUARES INPUT FILE

```
Number of Cases: 12
Number of Independant variables: 2
Use of intercept (y or n): y
Start q: 6
End q: 10
Display CRT graphics: y

INDEP      INDEP           DEP
VAR 1      VAR 2           VAR
==============================
#1
5.593      5.045           9.806
#2
5.697      5.191           9.900
#3
5.245      4.830           9.4911
#4
5.343      5.507           10.580
#5
5.380      5.100           10.370
#6
5.325      4.380           9.658
#7
5.252      4.393           9.497
#8
5.098      4.676           9.358
#9
5.615      5.313           9.826
#10
6.296      4.386           9.443
#11
5.461      4.792           9.781
#12
6.369      5.387           10.509
```

APPENDIX E

EXACT LEAST MEDIAN SQUARES SOURCE CODE

```
/* This program computes a lest median squares for a linear set of data */
#include <conio.h>
#include <pgchart.h>
#include <stdlib.h>

#include <math.h>
#include <malloc.h>
#include <stdio.h>
#include <graph.h>
#include <string.h>
#define INDEP(AR1,AR2) (*(indep+(AR1)*num_idep+(AR2)))
#define DEP(AR1) (*(dep+(AR1)))
#define XTX(AR1,AR2) (*(xtx+(AR1)*num_idep+(AR2)))
#define XTXI(AR1,AR2) (*(xtxi+(AR1)*num_idep+(AR2)))
#define H(AR1,AR2) (*(h+(AR1)*num_cases+(AR2)))
#define XTXIXT(AR1,AR2) (*(xtxixt+(AR1)*tc+(AR2)))
#define BETA(AR1) (*(beta+(AR1)))
#define X(AR1,AR2) (*(x+(AR1)*num_idep+(AR2)))
#define Y_PRIME(AR1) (*(y_prime+(AR1)))
#define E(AR1) (*(e+(AR1)))
#define Y(AR1) (*(y+(AR1)))
#define R_SQRED(AR1) (*(r_sqred+(AR1)))
#define THETA_C(AR1) (*(theta_c+(AR1)))
#define BEST_THETA(AR1,AR2) (*(best_theta+(AR1)+(AR2)*num_qs))
#define FTHETA(AR1,AR2) (*(ftheta+(AR1)+(AR2)*num_qs))
#define THETA_Q(AR1) (*(theta_q+(AR1)))
#define NUM_BIGGER(AR1) (*(num_bigger+(AR1)))
#define NEW_SOL(AR1) (*(new_sol+(AR1)))
#define SAVE_RS(AR1,AR2) (*(save_rs+(AR1)*num_cases+(AR2)))
#define Q_LAB(AR1) (*(q_lab+(AR1)))
#define Q_SOL(AR1) (*(q_sol+AR1))
#define RESID(AR1) (*(resid+(AR1)))
#define SQ(AR1) ((AR1)*(AR1))
#define SS(AR1) (*(ss+(AR1)))
#define STD_MEDS(AR1) (*(std_meds+(AR1)))
#define STD_SS(AR1) (*(std_ss+(AR1)))
#define MEDS(AR1) (*(meds+(AR1)))
#define RES_SORT(AR1,AR2) (((AR1)->res_sort+(AR2)))
#define RES_LAB(AR1) (*(res_lab+(AR1)))

   typedef struct res_struct
      {
         double res_v;
         int res_n;
      } res_T;

   typedef struct sol_struct
      {
         res_T  *res_sort;
         double std_factor;
```

```
        double med;
        double ss;
      } sol_T;
void delay(void);
char *get_valid(FILE *fp_input);
int comp_rid(double *arg1, double *arg2);
char bitchk(char *trim_mask, int pos_chk);
void setbit(char *trim_mask, int bit_num);
double get_sign(double arg);
void invert(double *out_mat,double *in_mat,int mat_size);
void flip_bit(char *trim_mask, int pos_clear);
int update(char *trim_mask,int *bbit,int num_cases,int pp1);

void get_std(sol_T *sol, int num_cases);
int c_rs(res_T *arg1, res_T *arg2);
int comp_dbl(double *arg1, double *arg2);
int c_r(res_T *arg1, res_T *arg2);

main(int num_args, char *args[2], char **menv)

/* This program accepts the input file name on
   the command line.  The file is assumed to have an extension of ".dat"
   The output from this program will be written to a file with a ".out"
   extension.  The output file will have the same name as the input file,
   and will be written in the same directory.  An example of what is expected
   on the command line is "B:\test" */

{
  sol_T **q_sol;

  char *file_name;        /* Input and Output filename */
  FILE *input_fp;         /* File pointer to input file */
  FILE *output_fp;        /* File pointer to output file */
  int  num_cases, num_idep, dep_start, num_track, num_start, cur_case, bbit;
  int  start, m_count, num_trim, exclude, row, col, offset, index, q;
  int  idep_start, idep_count, xclude, stop_look, *num_bigger;
  int  num_qs, cov_start, cov_end, tv;
  int  tc, bit_search, swaps, trim_search, swap_out, swap_in, num_sols;
  char int_flag, dif_flag, *new_sol, need_sort, do_g;
  char buffer[256], *tst_str;
  double *indep, *dep, *xtx, *xtxi, *xtxixt, *beta, *x, *y, *e, *y_prime;
  double r_sqr, r_abs, epsilon, *theta_c, *r_sqred, *theta_q;
  double *best_theta, r_squ, *save_rs, *resid, std_f;
  float  *ss, *meds, *std_ss, *std_meds;

  char *trim_mask, *swap_mask;
  char _far * _far *res_lab;
  char _far * _far *q_lab;
  chartenv env;
  float *ftheta;
  if(num_args!=2)  /* The user did not enter the correct number of arguments */
```

```
  {
     printf("This program expects to see an input file name on the command
line.\n");
     printf("The format for this input file is path\\filename without an
extension.\n");
     printf("An extension of .dat is assumed.\n");
     printf("The output file will be written to the same directory as the input
file.\n");
     printf("The output file will have an .out extension and will overwrite\n");
     printf("whatever was in filename.out file before the program was run.");
     exit(0);
  }
   file_name=malloc(sizeof(char)*80);
   file_name=strcpy(file_name,args[1]);
   file_name=strcat(file_name,".dat");

   input_fp=fopen(file_name,"rt");/* Open file for reading */

   if(input_fp==NULL) /* There was some problem opening the file */
   {
      printf("Problem opening %s for input",file_name);
      exit(0);
   }
/* Read header information from the file */
   num_cases=atoi(get_valid(input_fp));
   num_idep=atoi(get_valid(input_fp));
   int_flag=*(strtok(get_valid(input_fp)," \n"));
   cov_start=atoi(get_valid(input_fp));
   cov_end=atoi(get_valid(input_fp));
   do_g=*(strtok(get_valid(input_fp)," \n"));
   if(int_flag == 'y' || int_flag == 'Y')  /*  we are to compute an intercept */
      idep_start=1;
   else
      idep_start=0;
   num_idep+=idep_start;
   tc=num_idep+1;
   /*  n_over_2=ceil(num_cases/2);
       q=(int)(ceil((float)num_cases/2))+ceil((float)((float)(num_idep+1)/2));
   */
   num_qs=cov_end-cov_start+1;

   ss=malloc(sizeof(float)*num_qs);
   std_ss=malloc(sizeof(float)*num_qs);
   std_meds=malloc(sizeof(float)*num_qs);
   meds=malloc(sizeof(float)*num_qs);

   indep=malloc(sizeof(double)*num_cases*num_idep);
   dep=malloc(sizeof(double)*num_cases);
   trim_mask=malloc(sizeof(char)*num_cases+1);
   xtx=malloc(sizeof(double)*num_idep*num_idep);
   xtxi=malloc(sizeof(double)*num_idep*num_idep);
```

```
   xtxixt=malloc(sizeof(double)*num_idep*tc);
   x=malloc(sizeof(double)*tc*tc);
   beta=malloc(sizeof(double)*num_cases);
   y=malloc(sizeof(double)*tc);
   e=malloc(sizeof(double)*tc);
   theta_c=malloc(sizeof(double)*num_idep);
   r_sqred=malloc(sizeof(double)*num_cases);
   y_prime=malloc(sizeof(double)*tc);
   best_theta=malloc(sizeof(double)*num_idep*num_qs);
   num_bigger=malloc(sizeof(int)*num_qs);
   theta_q=malloc(sizeof(double)*num_qs);
   new_sol=malloc(sizeof(char)*num_qs);
   save_rs=malloc(sizeof(double)*num_qs*num_cases);
   ftheta=malloc(sizeof(float)*num_qs*num_idep);
   q_sol=malloc(sizeof(sol_T *)*num_qs);
   resid=malloc(sizeof(double)*num_cases);
   *(trim_mask+num_cases+1)='\000';
   for(index=0;index<num_qs;index++)
   {
      THETA_Q(index)=1000000;
      Q_SOL(index)=malloc(sizeof(sol_T));
      Q_SOL(index)->res_sort=malloc(sizeof(res_T)*num_cases);
   }
   for (cur_case=0;cur_case<num_cases;cur_case++)
   {
      do
      {
    if(feof(input_fp)) /* There was not enough data */
    {
       printf("Insufficient data: only %d cases found %d
expected\n",cur_case,num_cases);
       printf("Program terminating\n");
       exit(0);
    }
    fgets(buffer,256,input_fp);

    tst_str=strtok(buffer,"\t\n ");
    if(tst_str=='\0') tst_str=buffer;
      }while ((*(tst_str) < '0' || *(tst_str) > '9') && *(tst_str)!='-');

      if (idep_start==1) INDEP(cur_case,0)=1;
      for(idep_count=idep_start;idep_count<num_idep;idep_count++)
      {
    if (tst_str==NULL) /* There is insufficient data */
    {
       printf("data value missing in line %d\n",cur_case+1);
       printf("program terminating\n");
       exit(0);
    }
    INDEP(cur_case,idep_count)=atof(tst_str);
    tst_str=strtok(NULL," \t\n");
```

```
      }
      if (tst_str==NULL) /* There is insufficient data */
      {
     printf("data value missing in line %d\n",cur_case+1);
     printf("program terminating\n");
     exit(0);
       }

       DEP[cur_case]=atof(tst_str);
   }
   fclose(input_fp);

/* Initialize the bit mask */
   for(m_count=0;m_count<num_cases;m_count++)
   {
      if(m_count>=tc) /* We will not consider the case in the initial
             solution */
      *(trim_mask+m_count)='0';
      else
      *(trim_mask+m_count)='1';
   }
   bbit=tc-1;       /* Initailize the bottom bit to p+1 */

/* The do loop below will find OLS solutions and on all p+1 combinations
   of cases. */
   do
   {
/* Compute an OLS solution for the cases that were not trimmed */
/* Initalize matrixs */
      for(offset=0;offset<num_idep*num_idep;*(xtx+offset++)=0);

for(offset=0;offset<num_idep;*(y_prime+offset)=0,*(theta_c+offset)=0,*(beta+offset++)=0);
      for(offset=0;offset<tc;*(e+offset++)=0);
      for(offset=0;offset<num_cases;*(r_sqred+offset++)=0);
      bit_search=-1;
      for(offset=0;offset<tc;offset++)
      {
     for(bit_search++;bitchk(trim_mask,bit_search)=='0';bit_search++);
     Y[offset]=DEP[bit_search];

     for(index=0;index<num_idep;index++)
     {
         XTXIXT[index,offset]=0;
         X[offset,index]=INDEP[bit_search,index];
     }


      }
/* Compute X transpose X  */
      for (row=0;row<num_idep;row++)
      {
```

```
    for (col=0;col<num_idep;col++)
    {
       for(index=0;index<tc;index++)
       {
        XTX(row,col)+=X(index,row)*X(index,col);
        }
     }
       }
       invert(xtxi,xtx,num_idep);
/* Compute xtxixt */
      for(row=0;row<num_idep;row++)
      {
    for(col=0;col<tc;col++)
    {
       for(index=0;index<num_idep;index++)
       {
          XTXIXT(row,col)+=XTXI(row,index)*X(col,index);
       }
     }
       }
/* Compute beta vector */
      for(row=0;row<num_idep;row++)
      {
    for(index=0;index<tc;index++)
    {
       BETA(row)+=XTXIXT(row,index)*Y(index);
    }
       }
/* Compute the risiduals for the p+1 cases */
       r_squ=0;
       r_abs=0;
       for(row=0;row<tc;row++)
       {
      for(index=0;index<num_idep;index++)
      {
          E(row)+=X(row,index)*BETA(index);
      }
      E(row)=Y(row)-E(row);
      r_squ+=E(row)*E(row);
      r_abs+=fabs(E(row));
       }
      epsilon=r_squ/r_abs;
      for(index=0;index<tc;index++)
      {
     Y_PRIME(index)=Y(index)-epsilon*get_sign(E(index));
      }
/* Compute theta_c */
      for(row=0;row<num_idep;row++)
      {
    for(index=0;index<tc;index++)
    {
```

```
      THETA_C(row)+=XTXIXT(row,index)*Y_PRIME(index);
    }
     }
/* Compute the residuals */
     for(index=0;index<num_qs;NUM_BIGGER(index++)=0);
             /* num_bigger is used to count the number of residuals
             that are bigger than theta_q.  If we accumulate
             num_cases-q in this counter we know this is not a
             solution. num_cases-q has been computed above and
             the result is stored in stop_look */
     for(row=0;row<num_cases;row++)
     {
    for(index=0;index<num_idep;index++)
    {
       R_SQRED(row)+=INDEP(row,index)*THETA_C(index);
    }
    RESID(row)=DEP(row)-R_SQRED(row);
    R_SQRED(row)=RESID(row)*RESID(row);
    for(offset=0;offset<num_qs;offset++)
    {
       if(R_SQRED(row)>THETA_Q(offset)) NUM_BIGGER(offset)++;
    }
     }
     for(index=0;index<num_qs;NEW_SOL(index++)='0'); /*  NEW_SOL is initialized
                                to '0'.  If we find that
                                the solution under
                                consideration is  better
                                than the previous best
                                solution we will set the
                                NEW_SOL char to '1'. */
     need_sort='n'; /*  This value will be changed to 'y' in the loop below
             if a new solution is found. We will only need to sort the
             residuals if this value is  'y' at the end of the loop. */
     for(offset=0;offset<num_qs;offset++)
     {
    if(NUM_BIGGER(offset)<=num_cases-(cov_start+offset)) /* We have a new solution
*/
    {
       NEW_SOL(offset)='1';
       need_sort='y';
    }
     }
     if(need_sort=='y')
    {
       qsort(r_sqred,num_cases,sizeof(double),comp_rid);
       for(offset=0;offset<num_qs;offset++)
       {
          if(NEW_SOL(offset)=='1')
          {
          THETA_Q(offset)=R_SQRED(cov_start-1+offset); /* We need the -1
                                    because the
```

```
                                    R_SQRED is indexed
                                    from 0, while
                                    cov_start starts
                                    at 1 */
          for(index=0;index<(num_ideq);index++)
          {
             BEST_THETA(offset,index)=THETA_C(index);
          }
          Q_SOL(offset)->ss=0;
          for(index=0;index<num_cases;index++)
          {
             Q_SOL(offset)->ss+=R_SQRED(index);
             RES_SORT(Q_SOL(offset),index)->res_n=index+1;
             RES_SORT(Q_SOL(offset),index)->res_v=RESID(index);
             SAVE_RS(offset,index)=R_SQRED(index);
          }
          }
       }
    }
   }while(update(trim_mask,&bbit,num_cases,tc));

/* Now we print the results */
   file_name=strcpy(file_name,args[1]);
   file_name=strcat(file_name,".out");
   output_fp=fopen(file_name,"wt");/* Open file for writing */


   if(output_fp==NULL) /* There was some problem opening the file */
   {
      printf("Problem opening %s for output",file_name);
      exit(0);
   }

   for(index=0;index<num_qs;index++)
   {
      get_std(Q_SOL(index),num_cases);

      std_f=Q_SOL(index)->std_factor;
      SS(index)=(float)(Q_SOL(index)->ss);
      MEDS(index)=(float)(Q_SOL(index)->med);
      STD_SS(index)=(float)(Q_SOL(index)->ss/std_f/std_f);
      STD_MEDS(index)=(float)(Q_SOL(index)->med/std_f/std_f);

      fprintf(output_fp,"\n\nMedian Squares value for q=%d (non-std, std): %lf,
%lf\n",
             cov_start+index,
             THETA_Q(index),
             THETA_Q(index)/std_f/std_f);

      fprintf(output_fp,"SS (non-std, std): %lf, %lf",
             Q_SOL(index)->ss,
```

```
            Q_SOL[index]->ss/std_f/std_f);


      fprintf(output_fp,"\nTheta vector:\n");
      for(offset=0;offset<num_idep;offset++)
      {
    fprintf(output_fp,"%lf\n",BEST_THETA[index,offset]);
    FTHETA[index,offset]=(float)BEST_THETA[index,offset];

      }
      fprintf(output_fp,"\nCase number, Non-standard residual, Standard
Residual:\n");
      for(offset=0;offset<num_cases;offset++)
      {
    fprintf(output_fp,"%d,  %lf,  %lf \n",RES_SORT[Q_SOL[index],offset]->res_n,
                    RES_SORT[Q_SOL[index],offset]->res_v,
                    RES_SORT[Q_SOL[index],offset]->res_v/std_f);
      }


   }
   fclose(output_fp);
   if(do_g=='n' || do_g=='N') exit(0);
   res_lab=malloc(sizeof(char _far * far)*num_cases);
   q_lab=malloc(sizeof(char _far * far)*num_qs);
   for(index=cov_start;index<=cov_end;index++)
   {
      RES_LAB[index-cov_start]=_fmalloc(sprintf(buffer,"%d",index)+1);
      _fstrcpy(RES_LAB[index-cov_start],(char _far *)buffer);
   }

   for(index=cov_start;index<=cov_end;index++)
   {
      Q_LAB[index-cov_start]=_fmalloc(sprintf(buffer,"%d",index)+1);
      _fstrcpy(Q_LAB[index-cov_start],(char _far *)buffer);
   }
    if( ! setvideomode( _MAXRESMODE ) ) /* Find a valid graphics mode */
   exit( 1 );

    _clearscreen( _GCLEARSCREEN );
    _pg_initchart();                    /* Initialize chart system    */

    _pg_defaultchart( &env, _PG_LINECHART, _PG_POINTANDLINE );
    strcpy( env.maintitle.title, "SS For Non-Standard Residuals" );
    _pg_chart( &env, (char _far * far *)res_lab, (float _far *)ss, num_qs);

    getch();
    _clearscreen( _GCLEARSCREEN );
    _pg_initchart();                    /* Initialize chart system    */

    _pg_defaultchart( &env, _PG_LINECHART, _PG_POINTANDLINE );
```

```
    strcpy( env.maintitle.title, "Median of Non-Standard Squared Residuals" );
    _pg_chart( &env, (char _far * _far *)res_lab, (float _far *)meds, num_qs);

    getch();
    _clearscreen( _GCLEARSCREEN );
    _pg_initchart();                    /* Initialize chart system    */

    _pg_defaultchart( &env, _PG_LINECHART, _PG_POINTANDLINE );

    strcpy( env.maintitle.title, "SS For Standard Residuals" );
    _pg_chart( &env, (char _far * _far *)res_lab, (float _far *)std_ss, num_qs);

    getch();
    _clearscreen( _GCLEARSCREEN );
    _pg_initchart();                    /* Initialize chart system    */

    _pg_defaultchart( &env, _PG_LINECHART, _PG_POINTANDLINE );

    strcpy( env.maintitle.title, "Median of Standard Squared Residuals" );
    _pg_chart( &env, (char  far * far *)res_lab, (float _far *)std_meds, num_qs);

    getch();
    _clearscreen( _GCLEARSCREEN );
    for(index=0;index<num_idep;index++)
    {
       _pg_initchart();                 /* Initialize chart system    */

       _pg_defaultchart( &env, _PG_LINECHART, _PG_POINTANDLINE );

       sprintf(buffer,"Parameter %d",index+1-idep_start);
       strcpy( env.maintitle.title, buffer );

       _pg_chart( &env, (char _far * _far *)res_lab, (float _far
*)(ftheta+index*num_qs), num_qs);
       getch();
    }
    _setvideomode( _DEFAULTMODE );



   fcloseall();
}
```

```
char *get_valid(FILE *input_fp)
/* This function reads records until it encounters an non blank record.
   This non-blank record is parsed so that the string to the right of
   the ":" is returned.

   fp_input pointer to the file that is to be read
   This function returns a pointer to a string with usable data. */

{
   char buffer [256];      /* Buffer to store data read from file */


     do /* Until a valid record is reached */
     {
    fgets(buffer,256,input_fp);
     } while(strtok(buffer," \t\n")==NULL);
   (strtok(NULL,":"));
   return strtok(NULL,"\n");
}

char bitchk(char *trim_mask, int pos_chk)
{

   if (*(trim_mask+pos_chk)=='1') return '1';
   else return '0';
}
void setbit(char *trim_mask, int bit_num)
{
    *(trim_mask+bit_num)='1';
}
void invert(double *out_mat,double *in_mat,int mat_size)
{
/*  This fucntion inverts a matrix
    out_mat is a pointer to the output matrix
    in_mat is a pointer to the input matrix
    mat_size is the matrix size
*/
#define OUT_MAT(x,y) (*(out_mat+(x)*mat_size+(y)))
#define IN_MAT(x,y) (*(in_mat+(x)*mat_size+(y)))
#define WORK_MAT(x,y) (*(work_mat+(x)*mat_size+(y)))
   int row, col, index;
   double *work_mat, mult, div;
/* Initialize out_mat to the identity matrix
    and copy the input matrix so we do not corrupt it */
   work_mat=malloc(sizeof(double)*mat_size*mat_size);

   for(row=0;row<mat_size;row++)
   {
      for(col=0;col<mat_size;col++)
```

```
      {
    WORK_MAT(row,col)=IN_MAT(row,col);
    if (row==col) OUT_MAT(row,col)=1;
    else OUT_MAT(row,col)=0;
      }
   }
/* Invert the matrix */
   for (col=0;col<mat_size;col++)
   {
       div=WORK_MAT(col,col);
       for(index=0;index<mat_size;index++)
       {
     WORK_MAT(col,index)/=div;
     OUT_MAT(col,index)/=div;
       }
       for(row=0;row<mat_size;row++)
       {
    if(row==col) continue;
    else mult=WORK_MAT(row,col);
    for(index=0;index<mat_size;index++)
    {
        WORK_MAT(row,index)-=WORK_MAT(col,index)*mult;
        OUT_MAT(row,index)-=OUT_MAT(col,index)*mult;
     }
       }
   }
   free(work_mat);
}
void flip_bit(char *trim_mask, int pos_clear)
{
   if(*(trim_mask+pos_clear)=='1') *(trim_mask+pos_clear)='0';
   else *(trim_mask+pos_clear)='1';
}
int comp_rid(double *arg1, double *arg2)
{
/*  This function will be used by qsort to compare residuals
*/

    if(*arg1>*arg2) return 1;
    if(*arg1==*arg2) return 0;
    else return -1;

}
double get_sign(double arg)
{
/* This function returns a -1 if the argument is negative and a 1 other wise */
   if(arg<0) return -1;
   else return 1;
}

int update(char *trim_mask,int *nbit,int num_cases,int npi)
```

```
{
/*  This function will update the bit mask for the p+1 cases to be considered
    it is assumed that the first time that the routine is called the bit mask
    will have the first p+1 bits set and the rest with the value 0.  bbit is
    the bottom bit and represents the last bit in trim_mask that is set.
    This routine will update bbit */
    int bot_bits, index, num_set;
    printf("%s\n",trim_mask);

    if(*bbit<num_cases-1) /* merely move the bit down a notch */
    {
       flip_bit(trim_mask,(*bbit)++);
       flip_bit(trim_mask,*bbit);
       return 1;
    }
/* find out how many bits have accumulated at the bottom */
    for(bot_bits=0;bitchk(trim_mask,num_cases-bot_bits-1)=='1';bot_bits++)
    {
       flip_bit(trim_mask,num_cases-bot_bits-1);
    }

    if(bot_bits==pp1) /* We have been through all of the cases */
    return 0;
/* find the index of the bit to move down */
    for(index=num_cases-bot_bits-1;bitchk(trim_mask,index)=='0';index--);
    flip_bit(trim_mask,index);
/* set bits */
   for(num_set=0;num_set<bot_bits+1;num_set++)
   {
      flip_bit(trim_mask,index+num_set+1);
   }
   *bbit=index+num_set;
   return 1;
}




void get_std(sol_T *sol, int num_cases)
{
  /*  This rountine will assign values to the median sum of squares
      and the std_factor for the structure sol.
      The routine will also sort the residual array res_sort, by the value
      of the residual squared */
   int index;
   double *ss_sort, *s_vals, med_res;
   s_vals=malloc(sizeof(double)*num_cases);
/* This qsort uses c_r which compares the residuals.  The array is sorted by
   the value of the residuals and not the value of the residuals squared.
   The median residual is needed for the calcualtion of the standard factor */
```

```
   qsort(sol->res_sort,num_cases,sizeof(res_T),c_r);
   if(ceil(num_cases/2)==floor(num_cases/2))  /* There is an even number of
                              cases */

med_res=(double)(((RES_SORT(sol,num_cases/2)->res_v+RES_SORT(sol,(num_cases/2)-1)->res_v))/2);
   else
   med_res=(double)((RES_SORT(sol,(int)(floor(num_cases/2)))->res_v));
/* This qsort uses c_rs which comapares the value of the residuals squared.
   The median value produced by this sort is the median of the squared
   residuals.  This sort is done after the sort of the unsquared residuals so
   that when it comes time to print the results, the residuals will be in the
   sorted by the square of their value */

   qsort(sol->res_sort,num_cases,sizeof(res_T),c_rs);
   if(ceil(num_cases/2)==floor(num_cases/2))  /* There is an even number of
                              cases */

sol->med=(double)(((SQ(RES_SORT(sol,num_cases/2)->res_v)+SQ(RES_SORT(sol,(num_cases/2)-1)->res_v)))/2);
   else
   sol->med=(double)SQ(((RES_SORT(sol,(int)(floor(num_cases/2)))->res_v)));


   for(index=0;index<num_cases;index++)
   {
      *(s_vals+index)=fabs(RES_SORT(sol,index)->res_v-med_res);
   }
   qsort(s_vals,num_cases,sizeof(double),comp_dbl);
   if(ceil(num_cases/2)==floor(num_cases/2))  /* There is an even number of
                              cases */
   sol->std_factor=(double)((*(s_vals+num_cases/2)+*(s_vals+(num_cases/2)-1))/2);
   else
   sol->std_factor=(double)(*(s_vals+num_cases/2));
   sol->std_factor=sol->std_factor/0.6475;
   free (s_vals);
}
int c_rs(res_T *arg1, res_T *arg2)
{
/*  This function will be used by qsort to compare residual structures.
    The structures are to be sorted by the values of the squared residuals.
    The structure contains the unsquared residuals.  Every time that this
    function does a compare, it must do it based on the square of the values
    in the res_T structure  */
    double tst_val;
    tst_val=(arg1->res_v*arg1->res_v)-(arg2->res_v*arg2->res_v);
    if(tst_val>0) return 1;
    if(tst_val==0) return 0;
    if(tst_val<0) return -1;
}

int comp_dbl(double *arg1, double *arg2)
{
```

```
/*  This function will compare two double values for a qsort function */
   double tst_val;
    tst_val=*arg1-*arg2;

    if(tst_val>0) return 1;
    if(tst_val==0) return 0;
    if(tst_val<0) return -1;
}


int c_r(res_T *arg1, res_T *arg2)
{
/*  This function will be used by qsort to compare residual structures.
    The structures are to be sorted by the values of the residuals.
    The structure contains the unsquared residuals.   */
    double tst_val;
    tst_val=(arg1->res_v)-(arg2->res_v);
    if(tst_val>0) return 1;
    if(tst_val==0) return 0;
    if(tst_val<0) return -1;
}
```

APPENDIX F

EXACT LEAST MEDIAN SQUARES OUTPUT FILE

Median Squares value for q=6 (non-std, std): 0.000000, 0.005948
SS (non-std, std): 2.585829, 2713501.447393
Theta vector:
4.744950
0.904126
0.000832

Case number, Non-standard residual, Standard Residual:
3, -0.000010, -0.010190
9, -0.000038, -0.039374
8, -0.000075, -0.077126
2, -0.000075, -0.077126
7, -0.000075, -0.077126
1, 0.000075, 0.077126
12, 0.001189, 1.217874
11, 0.094630, 96.938411
6, 0.094934, 97.249725
5, 0.756608, 775.061775
10, -0.997977, -1022.317431
4, 0.999722, 1024.105311

Median Squares value for q=7 (non-std, std): 0.000000, 0.110378
SS (non-std, std): 2.590666, 5629769.843390
Theta vector:
4.741049
0.905572
0.000026

Case number, Non-standard residual, Standard Residual:
9, 0.000028, 0.041809
1, -0.000042, -0.062132
3, 0.000202, 0.298304
8, 0.000225, 0.332232
12, 0.000225, 0.332232
7, -0.000225, -0.332232
2, -0.000225, -0.332232
11, 0.094500, 139.306275
6, 0.094668, 139.554493
5, 0.756843, 1115.695037
10, -0.999642, -1473.615252
4, 1.000339, 1474.642427

Median Squares value for q=8 (non-std, std): 0.001420, 0.384547
SS (non-std, std): 3.417185, 925.539147
Theta vector:
5.175255

```
1.027102
-0.217224

Case number, Non-standard residual, Standard Residual:
2,  0.000956,  0.015740
1,  -0.017940,  -0.295243
3,  -0.022111,  -0.363899
6,  -0.035130,  -0.578157
9,  0.037680,  0.620118
11,  0.037680,  0.620118
8,  -0.037680,  -0.620118
12,  -0.037680,  -0.620118
7,  -0.118328,  -1.947381
5,  0.776780,  12.783840
4,  1.113193,  18.320347
10,  -1.246143,  -20.508362


Median Squares value for q=9 (non-std, std): 0.002247, 2111.360330
SS (non-std, std): 2.532352, 2379524.369400
Theta vector:
4.785891
0.906658
-0.000741

Case number, Non-standard residual, Standard Residual:
8,  -0.046567,  -45.139953
3,  -0.046632,  -45.202518
9,  -0.046837,  -45.401564
1,  -0.047089,  -45.646077
2,  -0.047273,  -45.824612
11,  0.047402,  45.949541
6,  0.047402,  45.949541
7,  -0.047402,  -45.949541
12,  -0.047402,  -45.949541
5,  0.710070,  688.310037
4,  0.953918,  924.685544
10,  -1.047958,  -1015.844093


Median Squares value for q=10 (non-std, std): 0.051829, 0.517364
SS (non-std, std): 0.785600, 7.842013
Theta vector:
5.537688
0.243836
0.592289

Case number, Non-standard residual, Standard Residual:
11,  0.073474,  0.232139
7,  0.076759,  0.242518
1,  -0.083561,  -0.264009
```

```
2,  -0.101394,  -0.320352
3,  -0.186264,  -0.588494
8,  -0.192308,  -0.607589
6,  0.227659,  0.719280
12,  0.227659,  0.719280
9,  -0.227659,  -0.719280
10,  -0.227659,  -0.719280
4,  0.477760,  1.509465
5,  0.499800,  1.579099
```

APPENDIX G

PRESENTATION CHARTS

INTRODUCTION

OBJECTIVES

DEFINITION OF TERMS

REVIEW OF REGRESSION ANALYSIS

ROBUST REGRESSION OVERVIEW

IMPLEMENTATION

CONCLUSION

OBJECTIVES

Choice of Project

- **Write software that has a useful application**
- **Dr. Triantis suggested Robust Regression**

Objectives of Project

- **Provide microcomputer implementation of Robust Regression techniques**
- **Tailor I/O process to user specifications**
- **Take advantage of the graphics capability of the modern microcomputer**

DEFINITION OF TERMS

Regression Analysis - A method of characterizing the relationship between a set of predictor variables and a parameter of interest.

Observation - A set of predictor variable values and a corresponding value for the parameter of interest (truth).

Breakdown Point - The maximum percentage of observations that can be aberrant without affecting the results of the Robust Regression model.

Coverage - A parameter associated with a Robust Regression algorithm; used to control the breakdown point.

DEFINITION OF TERMS

$\hat{Y}$ - Parameter value predicted by the regression model for a given set of predictor values.

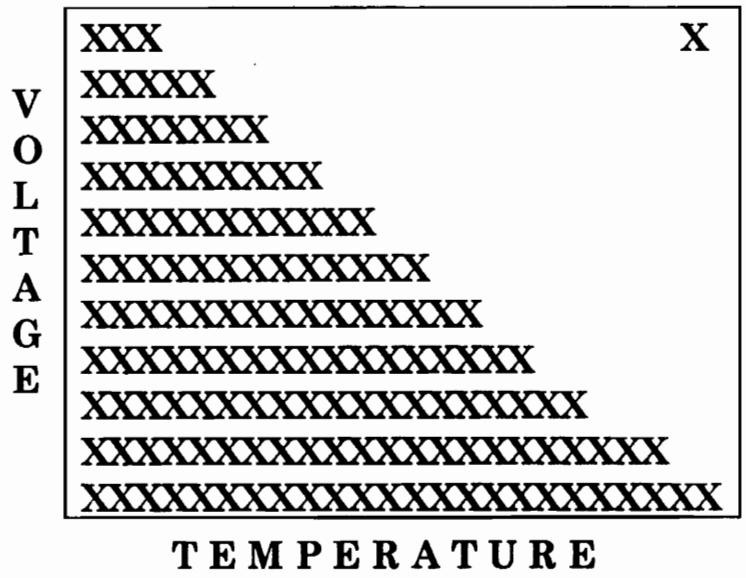
Residual - The difference between a known parameter associated with an observation and $\hat{Y}$ associated with the same observation.

Objective Function - Some function associated with the residuals of a regression algorithm which is either minimized or maximized.

Aberrant Data - An observation or set of observations which appears to be inconsistent with the remainder of the data.

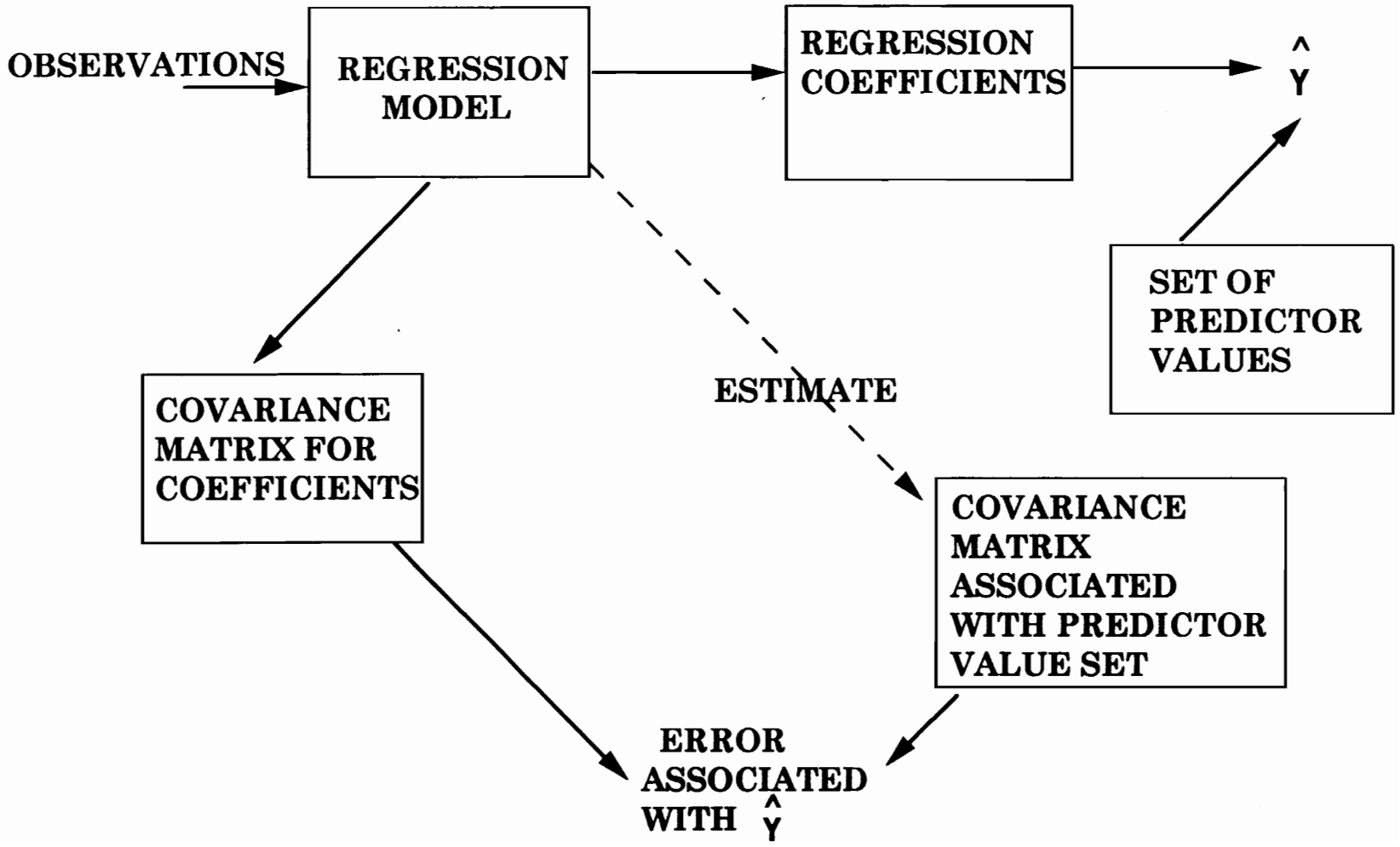
DEFINITION OF TERMS

Leverage Point - An observation with predictor variable values significantly different from the other observations.



Robust Regression Analysis - Regression analysis that is unaffected by some amount of aberrant data.

REVIEW OF REGRESSION ANALYSIS



ROBUST TECHNIQUES MAY NOT BE NECESSARY

IF:

1. **Control is well distributed (there are no leverage points).**

2. **The errors associated with $\hat{Y}$ are due to lack of precision in the measurements of the predictor variables (random errors) and not systematic errors.**

ROBUST REGRESSION OVERVIEW

	Least Trimmed Squares	Least Median Squares
Objective Function	Find the h observations which minimize the sum of the squared residuals when all other observations are ignored (h is a subset of all available observations).	Find the coefficients that minimize the qth largest residual squared.
Coverage	Determined by choice of h.	Determined by choice of q.
Computational Impact for increased coverage range	Computational requirement proportional to coverage range.	Little impact in increasing coverage range.
Maximum Breakdown Point	50%	50%

ROBUST REGRESSION OVERVIEW

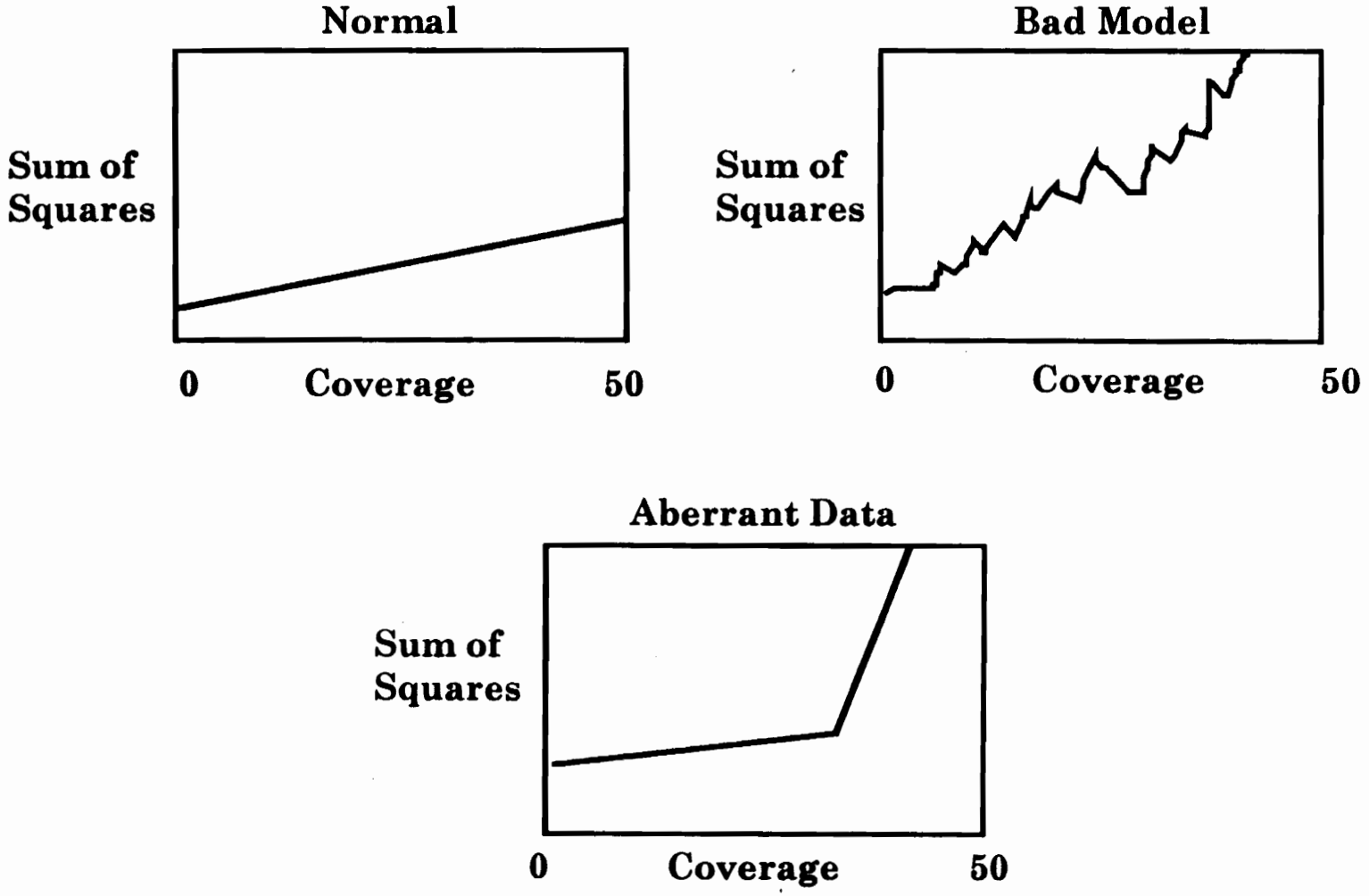
	Least Trimmed Squares	Least Median Squares
Computational Requirements (100 observations 20% breakdown point, 5 independent variables)	**Perform 70 80x80 matrix inversions.**	**Perform 5.36E20 6x6 matrix inversions.**
Probability of Finding Solution	**High**	**100%**

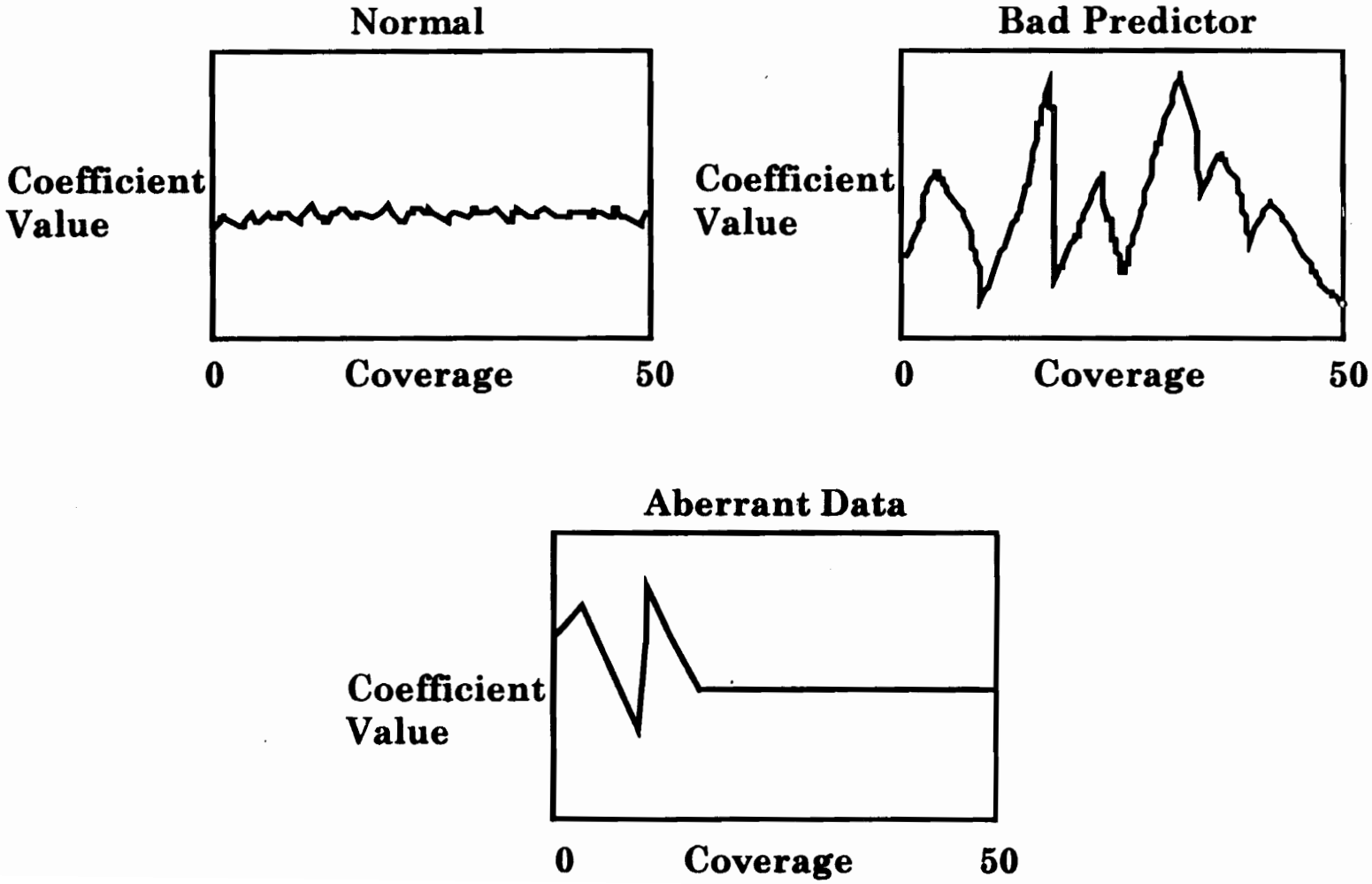
IMPLEMENTATION

INPUT

PROCESS

OUTPUTS


Normal
Sum of Squares
0
Coverage
50
Bad Model
Sum of Squares
0
Coverage
50
Aberrant Data
Sum of Squares
0
Coverage
50


Normal
Coefficient Value
0
Coverage
50
Bad Predictor
Coefficient Value
0
Coverage
50
Aberrant Data
Coefficient Value
0
Coverage
50

CONCLUSIONS

Robust Regression is not needed in all cases.

In most cases least trimmed squares will yield an acceptable solution with much less computational cost than the least median squares.

Improvements/Future Research:

- **Process**
  - **Approximate Least Median Squares**
  - **How many Starts are needed for a good LTS solution?**
  - **Is there a better way to choose the initial conditions for LTS?**
  - **Can more than one swap at a time help LTS?**

CONCLUSIONS
(continued)

Improvements/Future Research:

- **Implementation**
- **Guidelines for determining what data is aberrant and probability of making Type I error.**
- **Guidelines for interpreting coverage vs. coefficient graphs.**