The following is the chapter in my book that deals with nonlinear regression software:

Chapter 6 SOFTWARE

6.1 Introduction

One of the earliest applications of digital computers was least squares analysis of experimental data. The Manhattan Project during World War II included a large emphasis on experiments to determine basic properties such as half lives of radioactive isotopes, radiation shielding parameters, biological effects of radiation and many other properties of vital interest. The fundamentals of nonlinear least squares analysis was known then and was summarized in a book by W. E. Deming in 1943 [DE43]. An unclassified Los Alamos publication in 1960 by R. Moore and R. Zeigler described the software used at Los Alamos for solving nonlinear least squares problems [MO60]. Besides describing their general purpose software, they discussed some of the problems encountered in converging to a solution for some mathematical models.

Most readers of this book are either users of available NLR (nonlinear regression) software or are interested in evaluating and/or obtaining NLR software. Some readers, however, will be interested in writing their own software to solve a specific problem. Chapter 2 includes sufficient details to allow a user to rapidly get a system up and running. For all readers it should be useful to survey features that one would expect to see in a general purpose NLR program. It should be emphasized that there is a difference between a general purpose NLR program and a program written to quickly solve a specific problem. Indeed, using a language like MATLAB, some of my students in a graduate course that I have been teaching for a number of years (Design and Analysis of Experiments) have produced NLR code to solve specific homework problems.

Statistical software is available through the internet from a massive variety of sources. A Google search for "statistical software" turned up 9.5 million hits! Some of the software is free and other software programs are available for a price that can vary over a wide range. Some of the software includes nonlinear regression applications. Refining the search by adding "nonlinear regression" turned up over 600,000 hits. Many of these hits describe nonlinear regression modules that are part of larger statistical packages. Further refining the search to S-plus, the number of hits was over 26,000. Nonlinear regression software in S-plus is described by Venables and Ripley

[VE02]. Huet et. al. describe a program called NLS2 that runs under the R statistical software environment as well as S-plus [HU03]. Refining the search to SPSS, the number of hits was over 30,000. The SPSS Advanced Statistics Manual includes details for nonlinear regression analyses within SPSS [ZE98]. Refining the search to SAS, the number of hits was about 51,000. The NLIN procedure in the SAS system is a general purpose nonlinear regression program and is described in a paper by Oliver Schabenberger [SC98]. Refining the search to MATLAB, over 41,000 hits were noted. MATLAB *m* files for performing nonlinear regression analyses are included in [CO99]. The MATLAB Statistical Toolbox includes a function called *nlinfit* for performing nonlinear regression [www.mathworks.com/products/statistics].

In Section 6.2 features that are common to general purpose NLR programs are described and features that are desirable but not available in all the programs are also described. In Section 6.3 the NIST Statistical Reference Datasets are discussed. These well-known datasets are used to evaluate NLR programs and search algorithms. In Section 6.4 the subject of convergence is discussed. For most users, performance of NLR programs is primarily based upon a single issue: does the program achieve convergence for his or her problems of interest? In Section 6.5 a problem associated with linear regression is discussed. Multi-dimensional modeling is discussed in Section 6.6 and software performance is discussed in Section 6.7.

6.2 General Purpose Nonlinear Regression Programs

There are a number of general purpose nonlinear regression programs that can easily be obtained and allow the user to run most problems that he or she might encounter. Some of the programs are offered as freeware and some of the programs must be purchased. Some programs are offered on a free trial basis but must then be purchased if the user is satisfied and wishes to use the program after termination of the trial period. This section includes a survey of features that one encounters while reviewing nonlinear regression software. The purpose of this chapter is to provide the reader with the necessary background required to make a reasoned choice when deciding upon which program to use for his or her specific applications.

When one works within the framework of a general purpose statistical software environment (e.g., SAS, SPSS, S-plus, MATLAB Statistical Toolbox), a reasonable choice for nonlinear regression is a program that is compatible with the environment. Data created by one module of the system can then be used directly by the nonlinear regression module. Alternatively one can use a general purpose nonlinear regression program that runs independently (i.e., not within a specific statistical software environment). One problem with this alternative is data compatibility but this need not be a major obstacle. Most statistical software environments are Excel compatible, so if the nonlinear regression program is also Excel compatible, then data can be easily moved from the statistical software environment through Excel to the nonlinear regression program. In addition, ASCII text files can be used by almost all general purpose programs and statistical environments.

To qualify as a general purpose nonlinear regression program I feel that as a minimum, the following features should be included:

1) Mathematical models should be entered as input parameters.
2) The program should accept nonlinear models with respect to the unknown parameters (and not just nonlinear with respect to the independent variables).

3) There should be no need for the user to supply derivatives (neither analytical nor numerical) of the mathematical model.
4) The program should be able to accommodate mathematical models that are multi-dimensional in both the dependent and independent variables.
5) The user should be able to weight the data according to any weighting scheme of his or her choosing.
6) The program should include a sophisticated convergence algorithm. The National Institute of Standards nonlinear regression datasets (described in Section 6.3) provide a rich variety of problems that can be used to test the quality of a program's ability to achieve convergence.

In addition, there are a number of desirable features that one would like to see in a general nonlinear regression program:

1) Allow the user to name the dependent and independent variables and the unknown parameters.
2) Allow the user to define symbolic constants.
3) Allow specification of Bayesian estimators for the unknown parameters.
4) Include a simulation feature useful for designing experiments. (See Chapter 5 for a discussion and examples related to this feature.)
5) Allow input of Excel text files.
6) Include a feature to generate an interpolation table that lists values of the dependent variable and their standard deviations for a specified set of the independent variable or variables.
7) Allow program usage from within a general purpose statistical or programming environment.
8) Include a feature for generation of graphical output.

Treating mathematical models as input parameters is probably the most important feature of a general purpose NLR program. If the user is forced to program a function for every problem encountered, then the NLR program is not really "general purpose". If the user is working in an interactive mode and notes that a particular function does not yield results of sufficient accuracy, he or she should be able to enter a new function without having to exit the NLR program to reprogram the function.

The need for symbolic constants is a feature that can be most useful for problems in which convergence is difficult. This subject is discussed in Section 6.4.

There are several debatable features that are really a matter of user preference. Should the program use a parameter file for specifying the parameters of a particular analysis or should the program work through a GUI interface? Today, most computer programs (not just nonlinear regression programs) are interactive and allow the user to specify what he or she wishes to do thru a menu driven series of questions. For nonlinear regression, the number of parameters can be considerable so if the program is accessed through a GUI interface, there should be some method for shortcutting the process when the change from a previous analysis is minor. This particular problem is avoided if parameter files are used. All one has to do is edit the parameter file and make changes or perhaps copy the file under a new name and then change the new file.

The need for graphic output is a very reasonable user requirement, but should it be an integral part of an NLR general purpose program? As long as one can easily port the data to another program that supports graphics, then this should be a reasonable compromise. For example, if the

NLR program outputs text data, this output can then be inputted to a program like Excel to obtain the necessary graphical output.

6.3 The NIST Statistical Reference Datasets

The U.S. National Institute of Standards and Technology (NIST) initiated a project to develop a standard group of statistical reference datasets (StRD's). In their words the object of the project was "to improve the accuracy of statistical software by providing reference datasets with certified computational results that enable the objective evaluation of statistical software." One of the specific areas covered was datasets for nonlinear regression. The NIST StRD project home page can be accessed at:

http://www.itl.nist.gov/div898/strd/index.html

To examine the datasets, go into *Dataset Archives* and then *Nonlinear Regression*. A summary of the NIST nonlinear regression datasets is included in Table 6.3.1:

Name	***Difficulty***	***Parms***	***Num pts***	***Function***
Misrala	Lower	2	14	b1*(1-exp[-b2*x])
Chwirut1	Lower	3	214	exp[-b1*x]/(b2+b3*x)
Chwirut2	Lower	3	54	exp(-b1*x)/(b2+b3*x)
Lanczos3	Lower	6	24	b1*exp(-b2*x) + b3*exp(-b4*x) + b5*exp(-b6*x)
Gauss1	Lower	8	250	b1*exp(-b2*x) + b3*exp(-(x-b4)**2 / b5**2) + b6*exp(-(x-b7)**2 / b8**2)
Gauss2	Lower	8	250	Same as Gauss1
DanWood	Lower	2	6	b1*x**b2
Misralb	Lower	2	14	b1 * (1-(1+b2*x/2)**(-2))
Kirby2	Average	5	151	(b1 + b2*x + b3*x**2) / (1 + b4*x + b5*x**2)
Hahn1	Average	7	236	(b1+b2*x+b3*x**2+b4*x**3) / (1+b5*x+b6*x**2+b7*x**3)
Nelson	Average	3	128	b1 - b2*x1 * exp[-b3*x2]
MGH17	Average	5	33	b1 + b2*exp[-x*b4] + b3*exp[-x*b5]
Lanczos1	Average	6	24	b1*exp(-b2*x) + b3*exp(-b4*x) + b5*exp(-b6*x)
Lanczos2	Average	6	24	Same as Lanczos1
Gauss3	Average	8	250	Same as Gauss1
Misralc	Average	2	14	b1 * (1-(1+2*b2*x)**(-.5))
Misrald	Average	2	14	b1*b2*x*((1+b2*x)**(-1))
Roszman1	Average	4	25	b1 - b2*x - arctan[b3/(x-b4)]/pi

Name	***Difficulty***	***Parms***	***Num pts***	***Function***
ENSO	Average	9	168	b1 + b2*cos(2*pi*x/12) + b3*sin(2*pi*x/12) + b5*cos(2*pi*x/b4) + b6*sin(2*pi*x/b4) + b8*cos(2*pi*x/b7) + b9*sin(2*pi*x/b7)
MGH09	Higher	4	11	b1*(x**2+x*b2) / (x**2+x*b3+b4)
MGH10	Higher	3	16	b1 * exp[b2/(x+b3)]
Thurber	Higher	7	37	(b1 + b2*x + b3*x**2 + b4*x**3) / (1 + b5*x + b6*x**2 + b7*x**3)
BoxBOD	Higher	2	6	b1*(1-exp[-b2*x])
Ratkwosky3	Higher	3	9	b1 / (1+exp[b2-b3*x])
Ratkowsky4	Higher	4	15	b1 / ((1+exp[b2-b3*x])**(1/b4))
Eckerle4	Higher	3	35	(b1/b2) * exp[-0.5*((x-b3)/b2)**2]
Bennett5	Higher	3	154	b1 * (b2+x)**(-1/b3)

Table 6.3.1 Datasets from the NIST Nonlinear Regression Library

There are 27 different data sets included in the library including the actual data files (in text format) and results from least squares analyses of the data. The datasets are classified by Level of Difficulty (lower, average and higher), number of parameters (varying from 2 to 9), and number of data points (varying from 6 to 250). The datasets including the mathematical models are listed in Table 6.3.1. Each data set includes two different starting points: one near the solution and one further away from the solution. Also included are the least squares values and their estimated standard deviations. Results also include the sum of the squares of the residuals (i.e., S) and the Residual Standard deviation (i.e., $sqrt(S / (n-p))$). The datasets include a number of challenging problems that test the ability of a program to converge to a solution. However, the choice of datasets is limited to mathematical models that include a single independent variable x. Another limitation is that only unit weighting is used for the all problems. Details for one of the datasets (BoxBOD) are shown in Figure 6.3.1.

The BoxBOD problem has only two unknown parameters (i.e., b1 and b2) and only six data points and yet it is listed as *higher* in level of difficulty because of the difficulty of converging to a solution from the Start 1 initial values.

One of the most well-known general purpose nonlinear regression programs is **NLREG** (www.nlreg.com). They describe their results using the NIST datasets as follows: "The NIST reference dataset suite contains 27 datasets for validating nonlinear least squares regression analysis software. **NLREG** has been used to analyze all of these datasets with the following results: **NLREG** was able to successfully solve 24 of the datasets, producing results that agree with the validated results within 5 or 6 significant digits. Three of the datasets (Gauss1, Gauss2 and Gauss3) did not converge, and NLREG stopped with the message: *Singular convergence. Mutually dependent parameters*? The primary suggested starting values were used for all datasets except for MGH17, Lanczos2 and BoxBOD which did not converge with the primary suggested starting values but did converge with the secondary suggested starting values."

The differences between the three Gauss datasets are in the data. A plot of the Gauss1 data is shown in Figure 6.3.2. All three models include two Gaussian peaks with an exponentially decaying background. The peaks in the Gauss3 dataset are much closer together than the peaks in the other two datasets and that is why its level of difficulty is considered higher. However, NIST lists all three datasets as either lower or average level of difficulty. I ran Gauss1, Gauss2 and Gauss3 using **REGRESS** and had no problem converging from the primary suggested starting values. My guess is that somehow an error was introduced in the **NLREG** tests for these three datasets because it isn't logical that **NLREG** would fail for these and yet pass all the tests for the higher level of difficulty.

Another available NLR general purpose program is **LabFit** which can be located at http://www.angelfire.com/rnb/labfit/index.htm. Results for all the NIST datasets are included on the website. In their words they "achieved convergence for all the primary starting values for all the datasets and the results are statistically identical to the certified values".

Results for the NLR program **Stata** 8.1 can be seen at http://www.stata.com/support/cert/nist/. They achieved convergence for all of the datasets but for one dataset of average difficulty (MGH17) and 4 of higher difficulty (MGH09, MGH10, Eckerle4 and Ratkowsky4) they only achieved convergence from the nearer starting points. **Stata** is a "complete, integrated statistical package that provides everything needed for data analysis, data management, and graphics". The NLR module is only one feature of the **Stata** package.

```
NIST/ITL StRD
Dataset Name:  BoxBOD            (BoxBOD.dat)
Description:   These data are described in detail in Box, Hunter and
Hunter (1978).  The response variable is biochemical oxygen demand
(BOD) in mg/l, and the predictor variable is incubation time in days.

Reference: Box,G.P., W.G.Hunter, and J.S.Hunter(1978).
           Statistics for Experimenters.
           New York, NY: Wiley, pp. 483-487.

Data:      1 Response  (y = biochemical oxygen demand)
           1 Predictor (x = incubation time)
           6 Observations
           Higher Level of Difficulty
           Observed Data

Model:     Exponential Class
           2 Parameters (b1 and b2)

           y = b1*(1-exp[-b2*x])  +  e
Start 1 Start 2    Parameter     Standard Deviation
b1=1      100   2.1380940889E+02  1.2354515176E+01
b2=1     0.75   5.4723748542E-01  1.0455993237E-01

Residual Sum of Squares:       1.1680088766E+03
Residual Standard Deviation:   1.7088072423E+01
Degrees of Freedom:            4
Number of Observations:        6

Data:   y            x
      109            1
      149            2
      149            3
      191            5
      213            7
      224           10
```

Figure 6.3.1 Data and Results for NIST Dataset BoxBOD

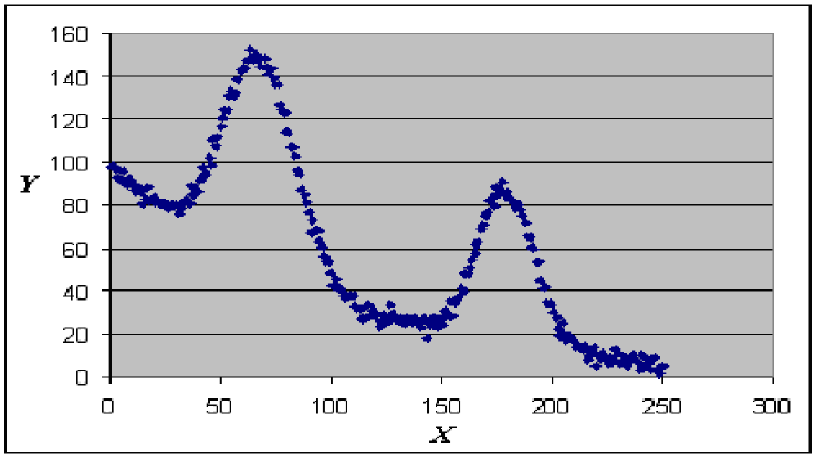

Figure 6.3.2 Gauss1 data from NIST Nonlinear Regression Library

Results for the program **Datafix** (a product of Oakdale Engineering) are available at http://www.curvefitting.com/datasets.htm. They achieved convergence for all datasets "without using analytical derivatives" but do not specify if this was from the primary or secondary starting points.

An Excel based NLR program is included as part of the XLSTAT package. Details can be obtained at http://www.xlstat.com/indexus.html. This package runs within Excel and they include the Ratkowsky4 example in their demonstration. Their solution requires programming of the derivatives of the modeling function and therefore cannot be considered as a general purpose NLR program. However, they have programmed complete solutions including derivatives for a limited number of functions.

An NLR program is included in the TSP econometrics package. The results for the NIST nonlinear reference datasets can be seen on the TSP International website:
http://www.tspintl.com/products/tsp/benchmarks/nlstab.txt
They achieved convergence on all the datasets except Lanczos1. No mention is made regarding the starting points for the various tests.

The **LIMDEP** program (a product of Econometric Software) is another general purpose statistical econometric package. Details regarding the **LIMDEP** program can be obtained at:
http://www.limdep.com/programfeatures_accuracy.shtml
The **LIMDEP** NLR module was tested using the NIST datasets as well as other benchmark datasets described by McCullough [MC99]. In their own words: "**LIMDEP** was able to solve nearly all the benchmark problems using only the program default settings, and all of the rest with only minor additional effort." This statement makes a lot of sense. Most general purpose NLR programs have default settings but for difficult problems, some minor adjustments in the parameters can lead to convergence. This subject is considered in Section 6.4.

6.4 Nonlinear Regression Convergence Problems

In Section 6.3 the NIST dataset library of NLR problems is discussed. The library has been used extensively to test NLR computer programs. The library has also been used to test convergence algorithms. The choice of an algorithm is a fundamental issue when developing NLR software and there are a number of options open to the software developer. It should be emphasized that there is no single algorithm that is best for all problems. What one hopes to achieve is an algorithm that performs well for most problems. In addition, for problems that are difficult to converge, a good NLR program should offer the user features that can help achieve convergence. In this section some of the features that enhance convergence are discussed using examples from the NIST library.

There are two basic classes of search algorithms that can be used for NLR problems:

1) Algorithms based upon usage of function derivatives to compute a vector of changes in the manner described in Section 2.4.
2) Stochastic algorithms that intelligently search thru a defined unknown parameter space.

The straight forward Gauss-Newton (GN) algorithm (Equations 2.4.16 and 2.4.17) is the starting point for most algorithms of the first type. This simple algorithm leads to convergence for many NLR problems but is not sufficient for more difficult problems like some of those encountered in the NIST datasets. To improve the probability of achieving convergence, Equation 2.4.16 can be replaced by:

$$\boldsymbol{a_k = a0_k + caf * A_k} \qquad \boldsymbol{k = 1 \text{ to } p} \qquad (6.4.1)$$

where ***caf*** is called the convergence acceleration factor. As a default ***caf*** is one, but for difficult problems, using a value of ***caf*** < 1 can sometimes lead to convergence. A more sophisticated approach is to calculate the value of ***S*** computed using the new values of $\boldsymbol{a_k}$ and compare this value with the value of ***S*** computed using the old values. As long as the value of ***S*** decreases, continue along this line (i.e., increase ***caf***). However, if the reverse is true (i.e., $\boldsymbol{S_{new}} > \boldsymbol{S_{old}}$) the value of ***caf*** is decreased (even to a negative number). It should be emphasized that ***caf*** is an input parameter and all changes of ***caf*** should be done algorithmically within the program for a given iteration. For the next iteration the value of ***caf*** is restarted at the input value. Sometimes it turns out that along the direction suggested by the ***A*** vector, ***S*** rises in both directions (i.e., ***caf*** > 0 and ***caf*** < 0). When this happens the algorithm can be modified to alter the direction. The Marquardt algorithm (sometimes called the Levenberg-Marquardt algorithm) is very popular and is used to modify the basic Gauss Newton algorithm [LE44, MA63, GA92]. Equations 2.4.16 or 6.4.1 are still used but the ***A*** vector is computed using a modified procedure. Instead of computing the ***A*** vector using Equation 2.4.9, the following equation is used:

$$\boldsymbol{A = (C + \lambda D)^{-1} V} \qquad (6.4.2)$$

The matrix ***D*** is just the diagonal of the ***C*** matrix (with all off diagonal terms set to zero) and $\boldsymbol{\lambda}$ is a scalar parameter. By trying several different values of $\boldsymbol{\lambda}$ a new direction can often be found which leads to a better reduction of ***S*** then achieved using Equation 2.4.16 or 6.4.1.

Tvrdik and Krivy survey several standard algorithms using the higher difficulty problems from the NIST datasets [TV04]. This paper can also be accessed online at http://albert.osu.cz/tvrdik/down/files/comp04.pdf. The algorithms used are those included in several NLR standard packages: NCSS 2001 which uses a Levenberg-Marquardt (LM) algorithm S-PLUS 4.5 which uses a GN algorithm, SPSS 10.0 which uses a modified LM algorithm and SYSTAT 8.0 which includes both a modified GN algorithm and an algorithm based upon the simplex method. Their results are shown in Table 6.4.1.

	NCSS		SYST GN		SYST Sim		S-Plus		SPSS	
Start:	1	2	1	2	1	2	1	2	1	2
Bennett5	2	1	OK	OK	F	F	OK	OK	OK	OK
BoxBOD	F	F	OK	OK	F	F	OK	OK	F	OK
Eckerle4	F	3	OK	OK	F	F	F	OK	OK	OK
MGH09	F	F	F	OK	OK	OK	F	2	OK	OK
MGH10	F	F	OK	OK	F	OK	F	OK	F	F
Ratkowsky3	OK	OK	OK	OK	F	F	F	OK	OK	OK
Ratkowsky4	F	3	OK	OK	F	F	F	OK	OK	OK
Thurber	F	F	OK	OK	OK	OK	F	F	F	F

Table 6.4.1 Comparison of algorithms for NIST datasets.

For each dataset, the programs were started from the far (1) and near (2) points as listed in the NIST reference datasets. An entry of F means that the program failed to converge and OK means that it did converge and ***S*** was accurate to at least 4 significant digits. A numerical entry means that it converged to 1, 2 or 3 significant digits. Clearly the SYSTAT program using the modified GN algorithm outperformed the other program but this does not mean that a GN algorithm is necessarily best. It does, however, prove that by cleverly modifying the basic algorithm one can achieve better results.

One of the easiest features that can be employed in an NLR program is to limit the search for some or all of the unknown parameters. For example, consider the BoxBOD dataset from the NIST library. Details are shown in Figure 6.3.1. Results obtained using the REGRESS program with only the default parameters are shown in Figure 6.4.1. An examination of the results shows that the value of **B2** becomes a huge negative number. Looking at the data in Figure 6.3.1 and the function used to specify **Y** we see that **Y** increases with **X** so **B2** must be a positive number. Setting a value of **B2MIN** = 0.001 and rerunning the program, the results in Figure 6.4.2 are obtained after 80 iterations. The ability to specify minimum and maximum values for the unknown parameters is an essential feature in a general purpose NLR program.

```
PARAMETERS USED IN REGRESS ANALYSIS: Thu Dec 02, 2004
   INPUT PARMS FILE: boxbod.par
   INPUT DATA  FILE: boxbod.par
   REGRESS  VERSION: 4.10, Nov 15, 2004
      STARTREC - First record used           :      1
      N - Number of recs used to build model  :      6
      NO_DATA - Code for dependent variable    -999.0
      NCOL - Number of data columns           :      2
      NY   - Number of dependent variables    :      1
      YCOL1 - Column for dep var Y               :  1
      SYTYPE1 - Sigma type for Y              :      1
         TYPE 1:  SIGMA Y = 1
      M - Number of independent variables     :      1
      Column for X1                           :      2
      SXTYPE1 - Sigma type for X1             :      0
         TYPE 0:  SIGMA X1 = 0
    Analysis for Set 1
      Function Y:  B1*(1-EXP[-B2*X])
      EPS - Convergence criterion           : 0.00100
      CAF - Convergence acceleration factor :   1.000
```

```
ITERATION          B1          B2  S/(N.D.F.)
        0     1.00000     1.00000    46595.60
        1    89.08912   114.70610    12878.94
        2   185.20000     <-10^49      >10^0
Singular matrix condition
```

Figure 6.4.1 Results for BoxBOD using Default Settings

There are some problems in which the values of the unknown parameters vary slowly but convergence is very difficult to achieve. For such problems setting upper and lower bounds on the parameters accomplishes nothing. The Bennett5 problem from the NIST datasets is an example of such a problem. Using the far starting points for the 3 unknowns, REGRESS required over 536,000 iterations to achieve convergence! Using the near starting points the results were not much better: over 390,000 iterations were required. REGRESS uses a modified GN algorithm but if the progress for an iteration is not sufficient it then uses an LM algorithm. A better approach for problems of this type is to use a stochastic algorithm. Stochastic algorithms avoid the need for function derivatives. A search space is defined by setting minimum and maximum values for all the unknown parameters. A random number generator is used to set a starting point within the space and then a heuristic is used to find the next point. In the same paper as mentioned above [TV04], Tvrdik and Krivy describe 5 different stochastic algorithms and then compare them using the same datasets as listed in Table 6.4.1. Their results show large performance differences from problem to problem and algorithm to algorithm. Four of the five managed to achieve solutions for the Bennett5 problem.

Another option for problems that are difficult to converge is to use symbolic constants. For example, the parameter file for the REGRESS runs for the Bennett5 problem included the following function specification:

```
unknown b1, b2, b3;
y ='b1 * (b2+x)^(-1/b3)'
```

Knowing the solution in advance, and noticing that the values of the unknowns were progressing in the correct direction, I just let REGRESS run until convergence was achieved. However, if the amount of data had been much greater than the 154 data records associated with this dataset, the time required to reach convergence would have been very large indeed. An alternative to this approach is to use symbolic constants. For example, one could hold **b1** constant and do a two parameter fit using the following function specification:

```
constant b1;
unknown b2, b3;
y ='b1 * (b2+x)^(-1/b3)'
```

Once least square values of **b2** and **b3** are located for the inputted value of **b1** the value can be changed and a new combination can be located. Comparing the values of ***S*** obtained for the different values of **b1**, one can home in on a region likely to contain the best value of **b1**. Once this region has been identified, one could then return to the original function specification to make the final 3 parameter search. The number of iterations using this procedure is much less than starting the process searching for all 3 parameters but requires a lot of user intervention and judgment.

For very difficult problems a combination approach is sometimes used. The process is started by doing a very course grid search through the entire space just computing ***S*** at all points in the grid.

The best region to start the search is around the point for which ***S*** is a minimum. All the unknowns are then bounded within this region and a detailed search is then initiated. If convergence is still a problem, then the use of symbolic constants and/or a stochastic algorithm can be used to further reduce the size of the search space.

```
PARAMETERS USED IN REGRESS ANALYSIS: Thu Dec 02, 2004
     ITERATION          B1          B2  S/(N.D.F.)
            0     1.00000     1.00000    46595.60
            1    89.08912   114.70610    12878.94
            2   185.20000     0.00100    46567.68
            3     9985.49     0.05420  7946907.22
            4    -1977.29     0.07206   917128.67
            5  -907.83514     0.00172    51485.53
            6     7854.00     0.00100    28275.45
            7    15098.02     0.00193     6215.57
            8    14635.85     0.00203     6194.60
            - - - - - - - - - - - - - - - - - - - - -
            - - - - - - - - - - - - - - - - - - - - -
           79   213.87781     0.54643   292.00568
           80   213.82425     0.54706   292.00238
POINT          X1           Y        SIGY       YCALC
    1     1.00000   109.00000     1.00000    90.10764
    2     2.00000   149.00000     1.00000   142.24097
    3     3.00000   149.00000     1.00000   172.40360
    4     5.00000   191.00000     1.00000   199.95134
    5     7.00000   213.00000     1.00000   209.17267
    6    10.00000   224.00000     1.00000   212.91397

PARAM INIT_VALUE  MINIMUM  MAXIMUM   VALUE     SIGMA
   B1    1.00000 Not Spec Not Spec 213.81258  12.35691
   B2    1.00000  0.00100 Not Spec   0.54720   0.10452
Variance Reduction:          88.05
S/(N - P)         :      292.00223
RMS (Y - Ycalc)   :       13.95235
```

Figure 6.4.2 Results for BoxBOD using B2MIN = 0.001

6.5 Linear Regression: a Lurking Pitfall

A general purpose NLR (nonlinear regression) program can easily handle linear regression problems. Software developed for nonlinear problems can be used with no change to solve linear problems. However, there is a hidden danger in using linear models that often plagues new users of curve-fitting software. When data is available and there is no physically meaningful mathematical model to explain the variation of a dependent variable ***y*** as a function of ***x***, the most tempting approach to the problem is to use a simple polynomial:

$$\boldsymbol{y} = \boldsymbol{a}_1 + \boldsymbol{a}_2 \boldsymbol{x} + \boldsymbol{a}_3 \boldsymbol{x}^2 + \ldots + \boldsymbol{a}_p \boldsymbol{x}^{\boldsymbol{p}-1} \qquad (6.5.1)$$

If one is only looking for an adequate function to predict ***y*** for any value of ***x*** then why not just start with a straight line (i.e., ***p*** = 2) and increase ***p*** until the average root-mean-square (RMS) error is acceptable? This approach, although theoretically very appealing, can lead to very difficult numerical problems that arise due to the fact that computers work to a finite number of significant digits of accuracy.

To explain the problem, consider data in which the values of $\boldsymbol{x}$ are equally spaced from 0 to 1 and unit weighting is used. The derivative of Equation 6.5.1 with respect to $\boldsymbol{a_k}$ is simply $\boldsymbol{x^{k-1}}$ so from Equations 2.4.14 and 2.4.15 the terms of the $\boldsymbol{C}$ matrix and the $\boldsymbol{V}$ vector are:

$$C_{jk} = \sum_{i=1}^{i=n} \frac{\partial f}{\partial a_j} \frac{\partial f}{\partial a_k} = \sum_{i=1}^{i=n} x^{j-1} x^{k-1} = \sum_{i=1}^{i=n} x^{j+k-2} \qquad (6.5.2)$$

$$V_k = \sum_{i=1}^{i=n} Y_i \frac{\partial f}{\partial a_k} = \sum_{i=1}^{i=n} Y_i x^{k-1} \qquad (6.5.3)$$

Once we have computed the terms of the $\boldsymbol{C}$ matrix and the $\boldsymbol{V}$ vector we use Equation 2.4.9 to solve for the vector $\boldsymbol{A}$:

$$A = C^{-1} V \qquad (2.4.9)$$

This vector includes all $\boldsymbol{p}$ values of the $\boldsymbol{a_k}$'s. We can estimate the value of $\boldsymbol{C_{jk}}$ by using the following approximation:

$$C_{jk} = \sum_{i=1}^{i=n} x^{j+k-2} = n\left(x^{j+k-2}\right)_{avg} \approx n\int_0^1 x^{j+k-2} dx = \frac{n}{j+k-1} \qquad (6.5.4)$$

For example, for $\boldsymbol{p}$=4 the $\boldsymbol{C}$ matrix is approximately:

$$C = n\begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{bmatrix} \qquad (6.5.5)$$

For those readers familiar with linear algebra, they will recognize this matrix as the well known Hilbert matrix and it has the following property:

$$cond(C) \approx e^{3.5p} = 10^{3.5p/ln(10)} \equiv 10^{1.5p} \qquad (6.5.6)$$

In other words, as the number of unknowns (i.e., $\boldsymbol{p}$) increases, the condition of the matrix (the ratio of the largest to smallest eigenvalues of the matrix) increases exponentially. Using **cond(*C*)** we can estimate the errors in the $\boldsymbol{a_k}$'s due to errors from the terms of the $\boldsymbol{V}$ vector:

$$\frac{\|\delta A\|}{\|A\|} \le cond(C) \frac{\|\delta V\|}{\|V\|} \qquad (6.5.7)$$

This equation means that the fractional errors in the terms of the $\boldsymbol{A}$ vector are no more than $\boldsymbol{cond(C)}$ times the fractional errors in the $\boldsymbol{V}$ vector. For example, let us assume that the values of $\boldsymbol{Y}$ are accurate to 5 decimal digits so that the fractional errors in the terms of the $\boldsymbol{V}$ vector are of the order of 10^{-5}. If $\boldsymbol{cond(C)}$ is about 100, then the fractional errors in the terms of the $\boldsymbol{A}$ vector are at worst of the order of 10^{-3}. This loss of accuracy comes about due to the process of inverting the $\boldsymbol{C}$ matrix. In other words, if $\boldsymbol{cond(C)}$ is about 100 we can expect a loss of about 2 digits of accuracy in solving Equation 2.4.6 (i.e., $\boldsymbol{CA} = \boldsymbol{V}$). A set of linear equations like Equation 2.4.6 is said to be "ill-conditioned" when the value of the condition become a large number.

Examining Equations 6.5.6 and 6.5.7, the pitfall in using Equation 6.5.1 for curve fitting can be seen. As $\boldsymbol{p}$ increases, $\boldsymbol{C}$ becomes increasingly ill-conditioned. The $\log_{10}$ of $\boldsymbol{cond(C)}$ is the maximum number of decimal digits that might be lost in solving Equation 2.4.6. So if $\boldsymbol{p}$ = 5, 6 or 7 then the condition is $10^{7.5}$, 10^{9} or $10^{10.5}$ and the number of digits of accuracy that might be lost are 7.5, 9 or 10.5! We see that even though Equation 6.5.1 is a very tempting solution for obtaining a simple equation relating $\boldsymbol{y}$ to $\boldsymbol{x}$, it is increasingly numerically problematical as $\boldsymbol{p}$ increases.

The NIST datasets include linear as well as nonlinear problems. The most difficult problem is the 'Filippelli problem'. This dataset has 82 point and the proposed model is Equation 6.5.1 with $\boldsymbol{p}$ =11. The LIMDEP website includes their solution to this problem and they describe the problem as follows:

> "LIMDEP's linear regression computations are extremely accurate. The 'Filippelli problem' in the NIST benchmark problems is the most difficult of the set. Most programs are not able to do the computation at all. The assessment of another widely used package was as follows: Filippelli test: XXXXX found the variables so collinear that it dropped two of them -- that is, it set two coefficients and standard errors to zero. The resulting estimates still fit the data well. Most other statistical software packages have done the same thing and most authors have interpreted this result as acceptable for this test. We don't find this acceptable. First, the problem is solvable. See LIMDEP's solution below using only the program defaults - just the basic regression instruction. Second, LIMDEP would not, on its own, drop variables from a regression and leave behind some arbitrarily chosen set that provides a 'good fit.' If the regression can't be computed within the (very high) tolerance of the program, we just tell you so. For this problem, LIMDEP does issue a warning, however. What you do next is up to you, not the program."

It should be emphasized that the Filippelli problem is a problem that was proposed to test software and not a real problem in which Mr. Filippelli was actually trying to get usable numbers. If one proceeds using Equation 6.5.1 directly, consider the loss of accuracy using a 10th order polynomial (i.e., $\boldsymbol{p}$ = 11) to fit the data. The number of digits of accuracy lost is at a maximum 16.5! Even if the values of $\boldsymbol{Y}$ are true values with no uncertainty, just inputting them into double precision numbers in the computer limits their accuracy to about 15 digits. So a loss of 16.5 digits makes the results completely meaningless. The $\boldsymbol{C}$ matrix is so ill-conditioned that it is no wonder that most packages fail when trying to solve the Filippelli problem. I tried running this problem using REGRESS and could not progress beyond $\boldsymbol{p}$ = 9.

So how did LIMDEP succeed while others have failed? I don't know the algorithm used by LIMDEP to solve problems based upon Equation 6.5.1, but if I was interested in creating software to solve such problems I would use orthogonal polynomials [RA78, WO71]. The idea originally proposed by G. Forsythe [FO57] is to replace Equation 6.5.1 with the following:

$$y = \sum_{k=0}^{k=p} a_k u_k(x) \qquad (6.5.8)$$

The $u_k(x)$ terms are a set of p polynomials all orthogonal to one another. Orthogonality for a particular set of data and a particular weighting scheme implies the following:

$$\sum_{i=1}^{i=n} w_i u_j(x_i) u_k(x_i) = 0 \qquad \text{for } j \neq k. \qquad (6.5.9)$$

Equation 2.4.5 is applicable to all linear models and is therefore applicable to Equation 6.5.8. Substituting u for g in Equation 2.4.5 we get p+1 equations of the following form (where the index k is from 0 to p):

$$a_0 \sum w_i u_0 u_k + a_1 \sum w_i u_1 u_k + \ldots + a_p \sum w_i u_p u_k = \sum w_i Y_i u_k \qquad (6.5.10)$$

Applying Equation 6.5.9 to 6.5.10 we end up with p+1 equations for a_k that can be solved directly:

$$a_k \sum w_i u_k u_k = \sum w_i Y_i u_k \quad k = 0 \text{ to } p \qquad (6.5.11)$$

$$a_k = \frac{\sum w_i Y_i u_k}{\sum w_i u_k u_k} \quad k = 0 \text{ to } p \qquad (6.5.12)$$

If a set of polynomials can be constructed with this property (i.e., Equation 6.5.9), then we can compute the terms of the A vector without inverting the C matrix. Or looking at it another way, the diagonal terms of the C^{-1} matrix are the inverses of the diagonal terms of the C matrix and all the off-diagonal terms are zero. Forsythe suggests the following scheme for computing polynomials satisfying Equation 6.5.9:

$$u_0(x) = 1 \qquad (6.5.13a)$$

$$u_1(x) = (x - \alpha_1)u_0(x) \qquad (6.5.13b)$$

$$u_2(x) = (x - \alpha_2)u_1(x) - \beta_1 u_0(x) \qquad (6.5.13c)$$

$$\cdot$$

$$u_p(x) = (x - \alpha_p)u_{p-1}(x) - \beta_{p-1}u_{p-2}(x) \qquad (6.5.13d)$$

The α's and β's are computed as follows:

$$\alpha_k = \frac{\sum_{i=0}^{n} x_i w_i (u_{k-1}(x_i))^2}{\sum_{i=0}^{n} w_i (u_{k-1}(x_i))^2} \qquad (6.5.14)$$

$$\beta_k = \frac{\sum_{i=0}^{n} w_i (u_k(x_i))^2}{\sum_{i=0}^{n} w_i (u_{k-1}(x_i))^2} \qquad (6.5.15)$$

The order of the computations is to first compute α_1 and the values of u_1, then β_1, α_2 and the values of u_2, then β_2, etc. until all the u's are known. Using Equation 6.5.12 the a_k's can be computed and thus all terms required by Equation 6.5.8 are known. As an example consider the data in Table 6.5.1.

Point	Y	x
1	7.05	0
2	16.94	1
3	31.16	2
4	48.88	3
5	71.31	4
6	96.81	5
7	127.21	6

Table 6.5.1 Data for Orthogonal Polynomial Example

Assuming unit weighting (i.e., $w_i = 1$), since $u_0 = 1$, from Equation 6.5.14 we compute α_1 as follows:

$$\alpha_1 = \frac{\sum_{i=0}^{n} x_i (u_0)^2}{\sum_{i=0}^{n} (u_0)^2} = \frac{21}{7} = 3$$

and therefore from Equation 6.5.13b $u_1 = x - 3$. We next compute β_1 and α_2 using Equations 6.5.15 and 6.5.14:

$$\beta_1 = \frac{\sum_{i=0}^{n}(u_1)^2}{\sum_{i=0}^{n}(u_0)^2} = \frac{9+4+1+0+1+4+9}{7} = \frac{28}{7} = 4$$

$$\alpha_2 = \frac{\sum_{i=0}^{n} x_i (u_1)^2}{\sum_{i=0}^{n}(u_1)^2} = \frac{0+4+2+0+4++20+54}{9+4+1+0+1+4+9} = \frac{84}{28} = 3$$

and therefore from Equation 6.5.13c $\boldsymbol{u_2}$ =($\boldsymbol{x}$-3)($\boldsymbol{x}$ – 3) – 4 = $\boldsymbol{x^2}$ - 6$\boldsymbol{x}$ + 5. In a similar manner we can compute $\boldsymbol{\beta_2}$ =3 and $\boldsymbol{\alpha_3}$ = 3 and thus $\boldsymbol{u_3}$ = ($\boldsymbol{x}$-3)$\boldsymbol{u_2}$ – 3($\boldsymbol{x}$-3) = $\boldsymbol{x^3}$-9$\boldsymbol{x^2}$+20$\boldsymbol{x}$-6. To use the $\boldsymbol{u_k}$'s to fit the data we next must compute $\boldsymbol{a_0}$, $\boldsymbol{a_1}$, $\boldsymbol{a_2}$ and $\boldsymbol{a_3}$ using Equation 6.5.12. The details of the calculation are included in Table 6.5.2.

The results in Table 6.5.2 include four different fits to the data:

$$y = a_0u_0 = 57.05$$
$$y = a_0u_0 + a_1u_1 = 57.05 + 20.01(x-3)$$
$$y = a_0u_0 + a_1u_1 + a_2u_2$$
$$= 57.05 + 20.01(x-3) + 2.004(x^2 - 6x + 5)$$
$$y = a_0u_0 + a_1u_1 + a_2u_2 + a_3u_3$$
$$= 57.05 + 20.01u_1 + 2.004u_2 + 0.00389u_3$$

The terms ***S* / (*n*-*p*-1)** are the sums of the squares of the residuals divided by the number of degrees of freedom. Using the goodness-of-fit criterion explained in Section 3.3 we note that the parabolic equation yields the best results because ***S* /(*n*-*p*-1)** is minimized for ***p*** =2 (i.e., 3 terms). We can convert this equation to the simple form of Equation 6.5.1:

$$y = (57.05 - 3*20.01 + 5*2.004) + (20.01 - 6*2.004)x + 2.004x^2$$
$$y = 7.04 + 7.986x + 2.004x^2$$

i	Y_i	x_i	u_0	u_1	u_2	u_3
1	7.05	0	1	-3	5	-6
2	16.94	1	1	-2	0	6
3	31.16	2	1	-1	-3	6
4	48.88	3	1	0	-4	0
5	71.31	4	1	1	-3	-6
6	96.81	5	1	2	0	-6
7	127.21	6	1	3	5	6
		$\sum Y_i u_k$	**393.36**	**560.37**	**168.37**	**0.84**
		$\sum u_k^2$	**7**	**28**	**84**	**216**
		a_k	**57.05**	**20.01**	**2.004**	**0.00389**
		S	**11552.5**	**337.7**	**0.198**	**0.194**
		$S/(n\text{-}p\text{-}1)$	**1925.4**	**67.54**	**0.049**	**0.065**

Table 6.5.2 Fitting Data using Orthogonal Polynomials

Regardless of the value of p the resulting equation derived using orthogonal polynomials can be converted to the simple very appealing polynomial form (i.e., Equation 6.5.1). For difficult linear problems such as the Filippelli problem this technique avoids the numerical pitfalls arising from the direct use of 6.5.1.

6.6 Multi-Dimensional Models

An important feature of general purpose NLR (nonlinear regression) programs is the ability to handle multi-dimensional problems. Throughout the book the discussion has primarily been about the relationship between a dependent scalar variable y and an independent scalar variable x. However, there are many problems throughout many fields of science and engineering where either x or y or both are vector variables. To test NLR programs it is useful to have a few examples of problems of these types. Unfortunately the nonlinear regression NIST datasets are limited to problems in which x and y are both scalars.

The theory and use of the **GraphPad Prism** program is included in a book written by H. Motulsky and A. Christopoulos [MO03]. The book can be downloaded from the GraphPad Software website (www.graphpad.com) and includes a very nice example of a problem in which the dependent variable y is a vector. Although **GraphPad Prism** is a general purpose NLR program, the book emphasizes analysis of biological and pharmaceutical experiments. Using GraphPad terminology, global models are models in which y is a vector and some of the unknowns are shared between the separate models for the components of y. A GraphPad example relevant to the pharmaceutical industry is the use of global models to analyze the dose-response curves of two groups (a treated group and a control group). The purpose of the experiment is to measure what they call **ec50** (the dose concentration that gives a response half-way between the minimum and maximum responses. For this experiment the x variable is the *log* of the dose, the first component of the y vector is the response of the control group and the second component is the response of the treated group. The problem is well documented in their book and data is included so that the problem can be used as a test dataset for any NLR program.

The experiment was analyzed using **REGRESS** and the results are very close to the results obtained with **Graphpad Prism**. The equations were specified as follows:

```
dependent ycont, ytreat;
independent x;
unknown  bottom, top, hillslope, logec50c,
         logec50t;
ycont  = 'bottom+(top-bottom)/
         (1+10^((logec50c-x)*hillslope))'
ytreat = 'bottom+(top-bottom)/
         (1+10^((logec50t-x)*hillslope))'
```

The two components of the ***y*** vector are **ycont** and **ytreat**. The unknown parameters shared by both equations are `bottom`, `top` and `hillslope`. The two remaining unknowns are the logs of **ec50** for the control and treatment groups (i.e., `logec50c` and `logec50t)`. The data is included in Table 6.6.1. The results are seen in Figure 6.6.1. REGRESS required 9 iterations to converge to the solution. The alternative to the global approach for this problem is to treat each curve separately. The reason for treating this problem using a global model is explained in the Graphpad document: the resulting accuracies for the values of `ec50` are reduced considerably using global modeling. The number of degrees of freedom for this problem (i.e., ***n-p***) is 10 – 5 = 5.

Point	x (log dose)	Ycont	Ytreat
1	-7.0	165	124
2	-6.0	284	87
3	-5.0	442	195
4	-4.0	530	288
5	-3.0	573	536

Table 6.6.1 Data for dose-response curve analysis from Graphpad.

```
REC Y-INDEX     X        YCONT      SIGYCONT  CALC_VALUE
  1      1   -7.00000  165.000      1.00000   152.28039
  2      1   -6.00000  284.000      1.00000   271.95980
  3      1   -5.00000  442.000      1.00000   455.54116
  4      1   -4.00000  530.000      1.00000   549.35957
  5      1   -3.00000  573.000      1.00000   573.06096

REC Y-INDEX     X        YTREAT    SIGYTREAT  CALC_VALUE
  1      2   -7.00000  124.000      1.00000   112.35928
  2      2   -6.00000   87.000      1.00000   123.13971
  3      2   -5.00000  195.000      1.00000   172.89774
  4      2   -4.00000  288.000      1.00000   321.78672
  5      2   -3.00000  536.000      1.00000   491.61468

PARAMETER INIT_VALUE  MINIMUM  MAXIMUM   VALUE    SIGMA
   BOTTOM    0.00000 Not Spec Not Spec  109.781  27.807
      TOP    1000.00 Not Spec Not Spec  578.939  34.182
HILLSLOPE    1.00000 Not Spec Not Spec  0.72458  0.1845
 LOGEC50C   -7.00000 Not Spec Not Spec -5.61755  0.1963
 LOGEC50T   -2.00000 Not Spec Not Spec -3.88429  0.1909

 Variance Reduction:          97.67  (Average)
    VR:          YCONT        99.26
    VR:         YTREAT        96.08
 S/(N - P)          :       1181.32
 RMS (Y - Ycalc)    :      24.30351  (all data)
      RMS(Y1-Ycalc):       13.15230
      RMS(Y2-Ycalc):       31.75435
```

Figure 6.6.1 Results from REGRESS analysis of data in Table 6.6.1.

A second example in which ***y*** is a vector is included in Section 6.8. A problem that demonstrates modeling with two independent variables was included in my first book [WO67]. This problem was related to a measurement of parameters related to the neutronics of heavy water nuclear reactors. The model was based upon the following equation:

$$y = \frac{(1+a_1 x_1)(1+a_2 x_1)}{(1+a_1 x_2)(1+a_2 x_2)} \qquad (6.6.1)$$

The unknowns a_1 and a_2 must be positive but there is no guarantee that the method of least squares will satisfy this requirement. However, we can force positive values by simply using b^2 in place of a. The modified equation is thus:

$$y = \frac{(1+b_1^2 x_1)(1+b_2^2 x_1)}{(1+b_1^2 x_2)(1+b_2^2 x_2)} \qquad (6.6.2)$$

The two unknowns are now b_1 and b_2 and regardless of the resulting signs of b_1 and b_2, the squared values are always positive. It should be noted that there are four possible solutions: both b_1 and b_2 can be positive or negative. Depending upon the initial guesses for b_1 and b_2, if convergence is achieved, the solution will be close to one of the four possibilities. The data for this problem is included in Table 6.6.2 and the results of the REGRESS analysis are seen in Figure 6.6.2. Note that for this problem since the σ's vary from point to point Equation 2.3.7

must be used to properly weight the data. The initial guesses were b_1 = 1 and b_2 = 10 and convergence was achieved with 3 iterations.

```
PARAM INIT_VALUE MINIMUM  MAXIMUM    VALUE      SIGMA
   B1  1.00000  Not Spec Not Spec  1.61876     0.22320
   B2 10.00000  Not Spec Not Spec  5.29172     0.34342

   Variance Reduction:          99.32
   S/(N - P)         :        6.98221
   RMS (Y - Ycalc)   :        0.01946
   RMS ((Y-Ycalc)/Sy):        2.62056
```

Figure 6.6.2 Results from REGRESS analysis of data in Table 6.6.2.

Point	Y	σ_y	x_1	σ_{x1} / x_1	x_2	σ_{x2} / x_2
1	0.7500	0.01000	0.0137	0.0056	0.0258	0.0057
2	0.5667	0.00833	0.0137	0.0056	0.0459	0.0065
3	0.4000	0.00620	0.0137	0.0056	0.0741	0.0070
4	0.8750	0.01243	0.0240	0.0086	0.0320	0.0068
5	0.7000	0.01022	0.0240	0.0086	0.0453	0.0057
6	0.5750	0.00863	0.0240	0.0086	0.0640	0.0054
7	0.3800	0.00586	0.0240	0.0086	0.0880	0.0055
8	0.5750	0.00863	0.0260	0.0093	0.0666	0.0122
9	0.2967	0.00777	0.0260	0.0093	0.1343	0.0134
10	0.1550	0.00290	0.0260	0.0093	0.2291	0.0140
11	0.0900	0.00189	0.0260	0.0093	0.3509	0.0143

Table 6.6.2 Modeling Data for Analysis of Equation 6.6.2.

Note that the value of b_1 is measured to 100 * 0.223 / 1.619 = 13.8% accuracy and b_2 is measured to 6.5% accuracy, but what we are really interested in are the values of a_1 and a_2 and their associated σ's. In general if we have v as a function of u we can relate σ_v to σ_u as follows:

$$\sigma_v^2 = \left(\frac{\partial f}{\partial u}\sigma_u\right)^2 \quad \text{where} \quad v = f(u) \qquad (6.6.3)$$

For $v = u^2$ from Equation 6.6.3 we get:

$$\sigma_v^2 = (2u\sigma_u)^2 \quad \text{where} \quad v = u^2 \qquad (6.6.4)$$

Dividing the equation by $v^2 = u^4$ we end up with the following simple relationship:

$$\left(\frac{\sigma_v}{v}\right)^2 = \left(\frac{2\sigma_u}{u}\right)^2 \quad \text{where} \quad v = u^2 \qquad (6.6.5)$$

In other words the relative uncertainty in v is twice as large as that for u. Using Equation 6.6.5 we see that the relative uncertainties of the a 's are twice those of the b 's. Thus for the problem in Figure 6.6.2, $a_1 = 1.619^2 = 2.621$ and σ_{a1} = 2.621*2*0.138 = 0.723. Similarly, a_2 = 27.99 and σ_{a2} = 3.64. It is interesting to note that REGRESS can solve this problem directly for the a 's by replacing Equation 6.6.1 by the following alternative:

$$y = \frac{(1+abs(a_1)x_1)(1+abs(a_2)x_1)}{(1+abs(a_1)x_2)(1+abs(a_2)x_2)} \qquad (6.6.6)$$

The ***abs*** (absolute) operator is a valid REGRESS operator that can be used in any function specification.

6.7 Software Performance

There are many ways to measure the performance of NLR (nonlinear regression) programs but for most problems the only relevant measure is the ability to converge to a solution for difficult problems. The NIST datasets are very useful for testing the ability of NLR programs to converge and this subject was considered in Sections 6.3 and 6.4. However, there are some problems where software performance metrics other than convergence are important. In particular, problems in which the amount of data is large, the time required to converge to a solution may become important. Another area where time is important is for calculations embedded within real time systems (e.g., anti-missile missile systems). When decisions must be made within a fraction of a second, if an NLR calculation is part of the decision making process, it is important to make the calculation as fast as possible. For real time applications general purpose NLR software would never be used. The calculation would be programmed to optimize speed for the particular system and mathematical model under consideration.

Since time is dependent upon hardware, one would prefer measures that are hardware independent. In this section some useful measures of performance (other than the ability to converge) are discussed. The total time that a program requires to achieve convergence for a particular program and a particular computer is approximately the following:

Converge_Time = Num_Iterations * Avg_Time_per_Iter (6.7.1)

The number of iterations required to achieve convergence is of course problem dependent but it can be used as a measure of performance when used for comparisons with common data sets such as the NIST datasets. The average time per iteration is of course computer dependent, but the effect of the computer is only a multiplicative speed factor:

Avg_Time_per_Iter = Speed_Factor * Avg_Calcs_per_Iter (6.7.2)

For traditional algorithms such as Gauss-Newton (GN) or Levenberg-Marquardt (LM) or some sort of combination, the average number of calculations per iteration can be broken down into 2 major terms:

Avg_Calcs_per_Iteration = Avg_CA_Calcs + Avg_S_Calcs (6.7.3)

The first term is a measure of the effort to compute the ***C*** matrix and then the ***A*** vector times the average number of times this operation is performed per iteration. The second term is a measure of the effort to compute the weighted sum-of-squares ***S*** times the average number of times this operation is performed per iteration. Both terms are proportional to ***n***, the number of data points. The first term also has a component that is proportional to $\boldsymbol{p}^3$ (the complexity of solving ***p*** simultaneous equations).

These equations are meaningless for people evaluating existing software as the actual numbers for a given problem are usually unavailable to the normal user. However, for those interested in developing software for performing NLR analyses for problems with important speed requirements, these equations give some indication where one should concentrate the effort at achieving speed.

For stochastic algorithms, these equations are not applicable. The concept of iterations is not really relevant. The entire calculation becomes essentially a series of calculations of $\boldsymbol{S}$. Whether or not this results in a faster overall computation is not obvious and clearly the speed of such algorithms is problem dependent.

6.8 The REGRESS Program

Throughout the book results for a number of examples have been obtained using the REGRESS program. The reason why I have chosen REGRESS is quite simple: I wrote it. The program can be downloaded from: www.technion.ac.il/wolberg. The history of the development of this program goes back to my early career when I was in charge of designing a sub-critical heavy water nuclear reactor facility. One of the experiments that we planned to run on the facility involved a nonlinear regression based upon Equation 6.6.2. In the 1960's commercial software was rare so we had no choice other than writing our own programs. It became quite apparent that I could generalize the software to do functions other than Equation 6.6.2. All that had to be done was to supply a function to compute $\boldsymbol{f}(\boldsymbol{x})$ and another function to compute the required derivatives. We would then link these functions to the software and could thus reuse the basic program with any desired function. At the time we called the program ANALYZER.

In the early 1970's I discovered a language called FORMAC that could be used for symbolic manipulation of equations. FORMAC was compatible with FORTRAN and I used FORTRAN and FORMAC to write a program similar to ANALYZER and I called the new program REGRESS. The REGRESS program accepted equations as input quantities. Using FORMAC, the program automatically generated equations for the derivatives and created FORTRAN subroutines that could then be used to perform the nonlinear regression (NLR). All these steps, including compilation and link-editing of the subroutines, were performed automatically without any user intervention. The REGRESS program became a commercial product on the NCSS time-sharing network and I had the opportunity to work with a number of NCSS clients and learned about many different applications of NLR.

In the mid 1970's I realized that with languages that support recursive program, I could avoid the need to externally compile subroutines. Recursion is the ability to call a subroutine from within itself. Using recursion, it became a doable task to write a routine to symbolically differentiate functions. Using PL/1 I rewrote REGRESS and added many new features that I realized were desirable from conversations with a number of users of REGRESS. I've returned to the REGRESS program on many occasions since the original version. In the 1980's I started teaching a graduate course called Design and Analysis of Experiments and I supplied REGRESS to the students. Many of the students were doing experimental work as part of their graduate research and the feedback from their experiences with REGRESS stimulated a number of interesting developments. In the early 1990's I rewrote REGRESS in the C language. Through the many version changes REGRESS has evolved over the years and is still evolving.

The REGRESS program lacks some features that are included in other general NLR programs. Some students who have recently used REGRESS have suggested that the program should have a GUI (Graphic User Interface) front end. Such a GUI would give REGRESS the look and feel of a modern program. Personally I have my doubts that this will make the program appreciably more user-friendly and have so far resisted creating such an interface. A more serious problem with REGRESS was the need to create data files in a format that the program could understand. Many users of the program gather data that ends up in an Excel Spread Sheet. The problem for such users was how to get the data into REGRESS. It turned out that the solution was quite simple: Excel allows users to create text files. A feature was added to accept Excel text files. Another important issue was the creation of graphic output. One of the features of REGRESS is that the entire interactive session is saved as a text file. The current method for obtaining graphics output is to extract the output data from the text file and then input it into a program such as Excel that supports graphics. Since this turns out to be a relatively painless process, the need for REGRESS to generate graphic output is not a pressing issue.

The REGRESS program includes some features that are generally not included in other NLR programs. The most important feature in REGRESS that distinguishes it from other general purpose NLR programs is the Prediction Analysis (experimental design) feature described in Chapter 5. Another important feature that I have not seen in other general purpose NLR programs is the ***int*** operator. This is an operator that allows the user to model initial value nonlinear integral equations. For example consider the following set of two equations:

$$\begin{aligned} y_1 &= a_1\int_0^x y_2 dx + a_2 \\ y_2 &= a_3\int_0^x y_1 dx + a_4 \end{aligned} \qquad (6.8.1)$$

These highly nonlinear and recursive equations can be modeled in REGRESS as follows:

_y_1 = '_a_1 * int(_y_2, 0, x) + _a_2'
_y_2 = '_a_3 * int(_y_1, 0, x) + _a_4'

This model is recursive in the sense that ***y*1** is a function of ***y*2** and ***y*2** is a function of ***y*1.** Not all general purpose NLR programs support recursive models. The user supplies values of ***x***, ***y*1** and ***y*2** for ***n*** data points and the program computes the least squares values of the a_k 's.

Another desirable REGRESS feature is a simple method for testing the resulting model on data that was not used to obtain the model. In REGRESS the user invokes this feature by specifying a parameter called NEVL (number of evaluation points). Figure 6.8.1 includes some of the REGRESS output for a problem based upon Equation 6.8.1 in which the number of data records for modeling was 8 and for evaluation was 7. Each data record included values of ***x***, ***y*1** and ***y*2** (i.e., a total of 16 modeling and 14 evaluation values of ***y*).** The program required 15 iterations to converge.

```
     Function Y1:   A1 * INT(Y2,0,X) + A2
     Function Y2:   A3 * INT(Y1,0,X) + A4

K    A0(K)    AMIN(K)   AMAX(K)      A(K)   SIGA(K)
1  0.50000  Not Spec  Not Spec   1.00493   0.00409
2  1.00000  Not Spec  Not Spec   2.00614   0.00459
3  0.00000  Not Spec  Not Spec  -0.24902   0.00079
4 -1.00000  Not Spec  Not Spec  -3.99645   0.00663

Evaluation of Model for Set 1:
  Number of points in evaluation data set:     14
  Variance Reduction (Average)              100.00
         VR:           Y1                   100.00
         VR:           Y2                   100.00
  RMS (Y - Ycalc)     (all data)           0.01619
         RMS (Y-Yc) - Y1                   0.02237
         RMS (Y-Yc)/Sy) - Y1               0.00755
         RMS (Y-Yc) - Y2                   0.00488
         RMS (Y-Yc)/Sy) - Y2               0.00220
  Fraction Y_eval positive               :  0.214
  Fraction Y_calc positive               :  0.214
     Fraction Same Sign                     :  1.000

  Data Set  Variable    Min     Max  Average Std_dev
  Modeling       X1  0.0100  6.2832   1.6970  2.3504
  Modeling       Y1 -7.9282  2.0000  -1.2393  3.7499
  Modeling       Y2 -4.1189  4.0000  -2.2600  3.1043

  Evaluate       X1  0.1500  5.2360   1.6035  1.8876
  Evaluate       Y1 -8.0000  1.3900  -2.1940  3.4524
  Evaluate       Y2 -4.1169  2.9641  -2.6260  2.7180
```
Figure 6.8.1 Recursion, the *int* operator & evaluation points