Implementation and Application of a Direct Matrix Solution Multiple-Variable Linear Regression Model

Liu, P.; Quinton, J.

Publisher's version / Version de l'éditeur:
https://doi.org/10.4224/8895033
Technical Report, 2005

DOCUMENTATION PAGE

REPORT NUMBER	NRC REPORT NUMBER	DATE
TR-2005-19		December 2005
REPORT SECURITY CLASSIFICATION		**DISTRIBUTION**
Unclassified		Unlimited

TITLE

IMPLEMENTATION AND APPLICATION OF A DIRECT MATRIX SOLUTION MULTIPLE-VARIABLE LINEAR REGRESSION MODEL

AUTHOR(S)

Pengfei Liu and Justin Quinton

CORPORATE AUTHOR(S)/PERFORMING AGENCY(S)

Institute for Ocean Technology, National Research Council, St. John's, NL
Memorial University of Newfoundland, St. John's, NL

PUBLICATION

SPONSORING AGENCY(S)

Institute for Ocean Technology, National Research Council, St. John's, NL
Memorial University of Newfoundland, St. John's, NL

IOT PROJECT NUMBER	NRC FILE NUMBER
42_980_26	

KEY WORDS	PAGES	FIGS.	TABLES
Multiple-Variable Linear Regression, Data Interpolation and approximation, and experimental design	iii, 21, App. A-B	3	

SUMMARY

A multiple-variable linear regression model that was formulated by a direct establishment of a system of linear equations was presented. This direct method was then implemented using C programming language. The source code of the implementation was given for four variables. Code for more variables can be extended easily with the source code attached. This mathematical software, called Data Groups Summarizer and Interpolator (DGSI) was applied, as an example, for a set of marine nozzle propeller data including both interpolation to produce curves and creation of linear regression coefficients for interpolation. Interpolation by the DGSI showed an accuracy of less than one-tenth of a percent for a group of nozzle propellers. This method and source code given here are a general-purpose linear regression tool so they can be applied for similar cases in other engineering applications.

ADDRESS: National Research Council
Institute for Ocean Technology
Arctic Avenue, P. O. Box 12093
St. John's, NL A1B 3T5
Tel.: (709) 772-5185, Fax: (709) 772-2462

National Research Council Canada
Institute for Ocean Technology

Conseil national de recherches Canada
Institut des technologies océaniques

IMPLEMENTATION AND APPLICATION OF A DIRECT MATRIX SOLUTION MULTIPLE-VARIABLE LINEAR REGRESSION MODEL

TR-2005-19

Pengfei Liu and Justin Quinton

December 2005

TABLE OF CONTENTS

Implementation and Application of A Direct Matrix Solution Multiple-Variable Linear Regression Model

Pengfei Liu, National Research Council Canada, Institute for Ocean Technology
and Justin Quinton, Memorial University of Newfoundland

Abstract

A multiple-variable linear regression model that was formulated by a direct establishment of a system of linear equations was presented. This direct method was then implemented using C programming language. The source code of the implementation was given for four variables. Code for more variables can be extended easily with the source code attached. This mathematical software, called Data Groups Summarizer and Interpolator (DGSI) was applied, as an example, for a set of marine nozzle propeller data including both interpolation to produce curves and creation of linear regression coefficients for interpolation. Interpolation by the DGSI showed an accuracy of less than one-tenth of a percent for a group of nozzle propellers. This method and source code given here are a general-purpose linear regression tool so they can be applied for similar cases in other engineering applications.

Keywords: Multiple-Variable Linear Regression, Data Interpolation and approximation, and experimental design

1. INTRODUCTION

In engineering research and design work, linear regression models and computer software packages are often used to obtain a set of polynomial coefficients from a group of experimental or computational data. These coefficients are then used to interpolate the data to produce curves based on the given values of independent variables.

In marine propeller research and development work, model tests and numerical simulation are the two fundamental performance evaluation tools. Experimental tests to predict the hydrodynamic characteristics of a propeller have been used since about a century ago. To provide the design charts for naval architects to design a propeller for a particular ship, a typical marine propeller model series, for example, a podded propeller model series, are to be tested. Before a test program starts, a number of propeller models need to be made according to the range of the geometry parameters and working conditions of the propellers. A typical test program is to produce a set of curves for a series of propellers to represent their hydrodynamic performance. For the hydrodynamic performance of a marine propeller, there are mainly two curves, the thrust coefficient K_T (non-dimensional thrust) curve and the torque coefficient K_Q (non-dimensional required power) curve. Both curves are the function of an advance coefficient J (non-dimensional speed). Let's focus on the K_T curve. This curve expresses a relationship between the dependant variable K_T

Authors' address: Pengfei Liu, National Research Council Canada, Institute for Ocean Technology, Box 12093, 1 Kerwin Place, St. John's, NF Canada A1B 3T5, email: Pengfei Liu@nrc.ca; and Justin Quinton, Faculty of Engineering and Applied Science, Memorial University of Newfoundland, St. John's, NF Canada A1B 3X5, email: Justin.Quinton@nrc.ca

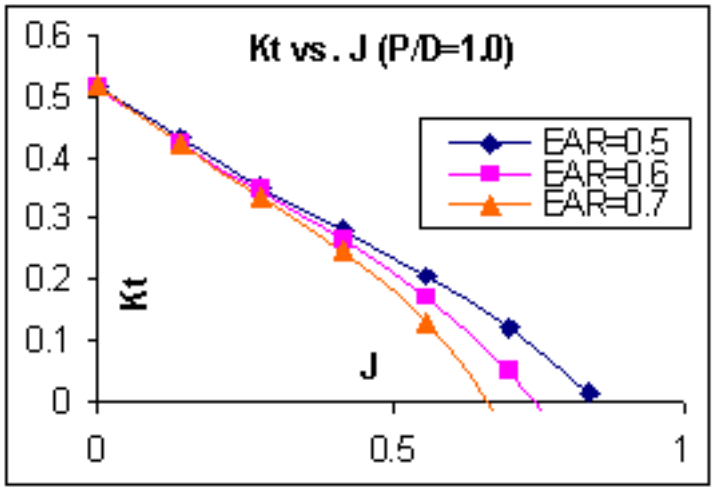

Fig. 1. Graph of K_T vs. J with $\frac{P}{D} = 1.0$ for three values of EAR.

and the independent variable J. Here we call J an primary independent variable. Figure 1 shows K_T versus J curves for the expanded area ratio $EAR = 0.5$, 0.6 and 0.7 for a marine propeller[Yossifov et al. 1989]. This is a set of two-independent-variable data group.

To cover a wide range of propellers that could be possibly employed for a typical marine vehicle category, the number of propeller models to be made needs to be determined. This number is normally determined by the variation of geometrical parameters, such as expanded area ratio EAR, number of blades Z, and nominal pitch ratio p/D. In this case, the number of total independent variables is four: J, EAR, Z and p/D, with J being the hydrodynamic motion parameter. If we chose four different expanded area ratios of EAR=0.5, 0.7, 0.9 and 1.1, four different blade numbers of Z=3, 4, 5, 6 and five pitch values of p/D=0.6, 0.8, 1.0, 1.2 and 1.4, then we need to build $4 \times 4 \times 5 = 80$ propeller models. Even if all the 80 models are tested, a linear regression model has to be used for a particular propeller, say, with an EAR=0.55, Z=5 and p/D=0.85, for which the K_T versus J curve is unknown. To reasonably reduce the model propeller manufacture cost, the variation intervals are made as large as possible. If each of the above variable's variation reduces one, the number of total required propeller models is $3 \times 3 \times 4 = 36$, a 55% saving on the model cost. This will save about \$220K if the cost of each propeller is at $5K$ (the lowest estimated price). Similarly, for a more expensive model in other application cases, the saving could be even bigger.

The key task of a multiple-variable linear regression model in this application is to provide a set of polynomial coefficients for which a K_T versus J curve can be found for a particular propeller that falls in to the series (for example, for a propeller with an $EAR = 0.65$, $Z = 6$, and $p/D = 1.15$). In the literature, in linear regression model for propeller applications, the number of total variables has not exceeded 4. This capability has not yet been found in commercial statistical software in the

current market either. The direct matrix formulation in the current work has also not seen to be published. For a larger set of data with more independent variables, a linear regression implementation that can handle more than four variables needs to be established.

Sufficient data points are required for the linear regression tool to generate polynomial coefficients. For the 36 model propeller case, if the advance coefficient J is taken at an interval of 0.05 within a range of 0-1.0, the number of total test runs is then $3 \times 3 \times 4 \times 20 = 720$. Therefore, 720 data points are to be used and inputted for the code to generate the polynomial coefficients.

Once the polynomial coefficients are obtained, the curve of K_T versus J for a particular propeller with an arbitrary value of EAR, p/D and Z within the range of the series definition can be determined by:

$$K_T = \sum_{i=0}^{i_m} \sum_{j=0}^{j_m} \sum_{k=0}^{k_m} \sum_{l=0}^{l_m} A_{ijkl} Z^i (EAR)^j (\frac{P}{D})^k J^l. \tag{1}$$

Due to memory shortage and computing power limits, the maximum values of i_m, j_m, k_m and l_m should be controlled as small as possible to save memory space of the generated matrix. However, too small a value of i_m, j_m, k_m and l_m will sacrifice the accuracy. Therefore, values of i_m, j_m, k_m and l_m need to be justified for both accuracy and computing efficiency. In the above example, if $i_m = j_m = k_m = l_m = 19$, the number of total coefficients to be found is $20 \times 20 \times 20 \times 20 \times = 160,000$. The required memory storage for the matrix with a double precision is then $160,000 \times 160,000 \times 8 = 190GB$. To solve this matrix requires large computing power. Because of this reason, all previous work used the value of power between 2-10 and gave a small number of the coefficients. With the rapid increase of processor clock speed, memory and high performance computing capability, it is possible to extend from 4 variables to 8 or more variables and a large number of generated coefficients should be read electronically without intensive labour.

2. ABOUT THE ALGORITHM

To use Equation 1 to interpolate the data given in Figure 2 to obtain a K_T versus J curve, we need a set of coefficients, that are represented by A_{ijkl}.

To find the set of polynomial coefficients A_{ijkl}, let

$$y = f(x_1, x_2, x_3, x_4) \tag{2}$$

be represented by a polynomial

$$y = \sum_{i=0}^{i_m} \sum_{j=0}^{j_m} \sum_{k=0}^{k_m} \sum_{l=0}^{l_m} A_{ijkl} x_1^i x_2^j x_3^k x_4^l. \tag{3}$$

For the values of exponents $i_m = 1$, $j_m = 1$, $k_m = 1$, and $l_m = 2$, Equation 3 become

$$y = A_{0000} x_1^0 x_2^0 x_3^0 x_4^0 + A_{0001} x_1^0 x_2^0 x_3^0 x_4^1 + \cdots + A_{1112} x_1^1 x_2^1 x_3^1 x_4^2. \tag{4}$$

Equation 4 may be written in a matrix form, which can be solved for the polynomial coefficients directly:

$$[X][A] = [Y], \tag{5}$$

where $[Y]$ is a vector with 24 elements, $[A]$ is a vector with 24 elements, and $[X]$ is a square matrix with a dimension of 24×24. $[Y]$ is filled with the values determined by the independent variables (for example, the values of Y at given x_1, x_2, x_3 and x_4), $[A]$ is unknown vector, and $[X]$ is filled with the elements calculated from the independent variables (for example x_1, x_2, x_3 and x_4) raised to their respective powers.

Expanding Equation 5, it gives

$$\begin{bmatrix} 1 & x^0_{1,1}x^0_{2,1}x^0_{3,1}x^1_{4,1} & \cdots & x^1_{1,1}x^1_{2,1}x^1_{3,1}x^2_{4,1} \\ 1 & x^0_{1,2}x^0_{2,2}x^0_{3,2}x^1_{4,2} & \cdots & x^1_{1,2}x^1_{2,2}x^1_{3,2}x^2_{4,2} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x^0_{1,23}x^0_{2,23}x^0_{3,23}x^1_{4,23} & \cdots & x^1_{1,23}x^1_{2,23}x^1_{3,23}x^2_{4,23} \\ 1 & x^0_{1,24}x^0_{2,24}x^0_{3,24}x^1_{4,24} & \cdots & x^1_{1,24}x^1_{2,24}x^1_{3,24}x^2_{4,24} \end{bmatrix} \begin{bmatrix} A_{0000} \\ A_{0001} \\ \vdots \\ A_{1111} \\ A_{1112} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{23} \\ y_{24} \end{bmatrix}. \tag{6}$$

The subscript number following the comma designates the point number. Solving Equation 4 yields the vector $[A]$, i.e. the polynomial coefficients A_{ijkl}.

3. IMPLEMENTATION

3.1 The Need

In marine propeller series development work, a base model propeller is usually designed first. Other models are made with variations in expanded area ratio EAR, pitch diameter ratio p/D, number of blades Z, etc. We call these geometry variables here the independent variables. In the model tests, we need to obtained the propulsion performance of a propeller model. There are mainly two physical quantities to represent propulsion performance, i.e., thrust coefficient K_T and torque coefficient K_Q, versus advance coefficient J. Therefore, K_T and K_Q are the dependant variables and J is the independent variable. For each model test, two categories of curves, i.e. a K_T and a K_Q curve, are produced. For each curve type, linear regression is used to find a set of polynomial coefficients. These coefficients are then used as input to generate new curves for a variation of one or all the independent variables. To automate the task, there are two things to be done, i.e., the coefficient generator (the group data summarizer) and the new curve generator (the interpolator). The integration of these two is named DGSI.

The summarizer takes experimental data appropriately ordered as input in an ASCII file of type *.xxv (where xx is the number of variables in the equation and v stands for variable), and generates coefficients for the data, written into an ASCII file of type *.OUT. The interpolator takes the coefficients and values for the independent variables as input in an ASCII file of type *.xxi, and generates values of the dependent variable, written into an ASCII file also of type *.OUT.

Together, the summarizer and interpolator form a powerful tool. Experimental data on a phenomena is collected and the summarizer is used to generate coefficients

for the experiment. The interpolator is then used to obtain values of the dependent variable at other points without performing other experiments or the need of new model propellers. The data is then organized into the desired format and displayed graphically using other programs such as Microsoft's Excel or Tecplot. DGSI does not perform extrapolation. Any values of the independent variables used outside of the boundary range which generated the coefficients will give results that should be considered unreliable. In this paper, examples are given for K_T versus J curve only. Code usage procedure for K_Q versus J curve is the same.

3.2 Input Files

The code requires an ASCII file as input (of appropriate type whether using the summarizer or interpolator) and gives an ASCII file as output. Input files have a specific format that must be observed for the program to behave as designed. Following are sample input files for the summarizer (Section 3.2.1) and the interpolator (Section 3.2.2). These samples have types *.04v and *.04i respectively, because each has four variables.

3.2.1 *Summarizer Input Files.* A sample summarizer input file is shown as an example as the following. In this example, very small values of the exponents are used to simply the length of explanation.

```
Number of variables
4
Number of polynomial terms for each variable
2  2  2  3
List variables and results, at least 2x2x2x3=24 lines for 4 variables x2, x3, x4, x1 and Kt
2 0.5 1 0 0.55
2 0.5 1 0.3 0.505
2 0.5 1 0.6 0.37
2 0.5 1.1 0 0.6
2 0.5 1.1 0.3 0.555
2 0.5 1.1 0.6 0.42
2 0.6 1 0 0.56
2 0.6 1 0.3 0.5141
2 0.6 1 0.6 0.3764
2 0.6 1.1 0 0.61
2 0.6 1.1 0.3 0.5641
2 0.6 1.1 0.6 0.4264
3 0.5 1 0 0.55
3 0.5 1 0.3 0.50275
3 0.5 1 0.6 0.361
3 0.5 1.1 0 0.6
3 0.5 1.1 0.3 0.55275
3 0.5 1.1 0.6 0.411
3 0.6 1 0 0.56
3 0.6 1 0.3 0.5114
3 0.6 1 0.6 0.3656
3 0.6 1.1 0 0.61
3 0.6 1.1 0.3 0.5614
3 0.6 1.1 0.6 0.4156
```

Certain rules are made for creating summarizer input files:

(1) Columns of numbers in the list of variable values must correspond to each number in the polynomial terms, with the exception of the last column, the resultant thrust coefficient, K_T values. The number of values of variable x_2 is 2 and they are 2 and 3; The number of values of variable x_3 is 2 and they are 0.5 and 0.6; The number of values of variable x_2 is 2 and they are 1.0 and 1.1;

and The number of values of variable x_1, the primary independent variable, is 3 and they are 0.0, 0.3 and 0.6.

(2) The last column of values are to be the resultant K_T values, the values of the dependant variable for the corresponding set of points, that come from a series tests or numerical predictions.

(3) Values are to be listed in ascending, repeating order in each column, and the right-most column, excluding the K_T column, must have the fastest changing index.

(4) Precise lettering and wording of the text in the file is not important, but some descriptive text needs to be included at places shown in the sample and be kept on a single line.

3.2.2 *Interpolator Input Files.* Sample interpolator input file is as follows:

```
Number of variables
4
Number of polynomial terms for each variable
2  2  2  3
List coefficient a[1-2][1-2][1-2][1-3]=>a[0][0][0][0], ..., a[1][1][1][2]
0
-0.000025
-0.44993
0.5
0.000022
-0.000064
0.1
0.000041
-0.000118
0
-0.000034
0.000106
0
0.000007
-0.000022
0
-0.000006
0.00002
0
-0.000012
-0.049963
0
0.00001
-0.000033
Number of sets of variables to inter-extrapolate
24
list of this 24 sets of variables
2 0.5 1 0
2 0.5 1 0.3
2 0.5 1 0.6
2 0.5 1.1 0
2 0.5 1.1 0.3
2 0.5 1.1 0.6
2 0.6 1 0
2 0.6 1 0.3
2 0.6 1 0.6
2 0.6 1.1 0
2 0.6 1.1 0.3
2 0.6 1.1 0.6
3 0.5 1 0
3 0.5 1 0.3
```

```
3 0.5 1 0.6
3 0.5 1.1 0
3 0.5 1.1 0.3
3 0.5 1.1 0.6
3 0.6 1 0
3 0.6 1 0.3
3 0.6 1 0.6
3 0.6 1.1 0
3 0.6 1.1 0.3
3 0.6 1.1 0.6
```

Some rules are made for the Interpolator input file:

(1) Coefficients are to be in the correct order and all zero values of the coefficient are be entered (no omission). Coefficient numbering must coincide with polynomial numbering.

Number of polynomial terms for each variable
2 2 2 3
List coefficient $a[1-2][1-2][1-2][1-3] \rightarrow a[0][0][0][0] \ldots a[1][1][1][2]$
value of $a[0][0][0][0]$
value of $a[0][0][0][1]$
$\vdots$
value of $a[1][1][1][2]$
...

(2) Number of coefficients must be equal to the product of the polynomial terms, i.e., Number of polynomial terms for each variable
2 2 2 3
...
Therefore, the number of total coefficients is $2 \times 2 \times 2 \times 3 = 24$, i.e., from $a[0][0][0][0]$ to $a[1][1][1][2]$, because the integer value of the exponent starts from 0.

(3) Columns of numbers in the list of variable values must correspond to each number in the polynomial terms, i.e.
Number of polynomial terms for each variable
2 2 2 3
...
List of a set of variables for $x_2 = 2$, $x3 = 0.5$, $x_4 = 1.0$ and x_1 =0.0, 0.3 and 0.6 is, for example, as follows:
2 0.5 1 0
2 0.5 1 0.3
2 0.5 1 0.6
...

Therefore, the variable corresponding to the polynomial term '3' must have three values. In this case, they are 0, 0.3 and 0.6.

Any deviation from these rules, either for the summarizer or the interpolator, may cause the output data to be incorrect or the entire output file to be nonsensical. In some cases of bad input file setup, errors may be generated upon running the program.

3.3 Obtaining Data from the Input Files

Once the files are prepared correctly and the program executed, the first thing the program must do is read the data from the files so it can be used. Two functions that are primarily used for accomplishing this are fgets, for getting the text labels, and fscanf, for getting the numerical data (both functions are from the standard C library).

Each item, whether a label or data, that is read from the input file is also written to the output file. Writing the read data to the output file makes it easier to detect errors in the input file; if the data is read incorrectly from the input file, it will be written incorrectly to the output file.

The polynomial terms are used to calculate how many sets of data points to read for the summarizer, or how many coefficients to read for the interpolator. It calculates how many of these sets (or coefficients) to read by simply taking the product of the polynomial terms, which will be called P. If the number of sets (or coefficients) does not equal P, then the program will not behave predictably and it will certainly not produce the correct output.

3.4 Program use of the Data

When the data is read, it must be stored where the program can access it quickly. The space required for storing the data is not known until the program reads the file so memory must be dynamically allocated for the data. Dynamic allocation of memory in the code is performed by the functions vector and matrix [Press et al. 1999]. Once the space is allocated, the values are simply stored in the vector or matrix as they are read. At the end of the program, the memory allocated is released by the program using the functions free_vector and free_matrix. Failing to free the memory may cause a 'memory leak' and the systems resources would quickly be consumed.

3.5 Building the Matrices to Solve

Gathering the initial data is the first step. Matrices must be built which can be solved to find the value of the coefficients. The $[Y]$ matrix from Equation 5 is created when the data is being read and is a column matrix with P rows. It is filled with the thrust coefficient, K_T values. The $[A]$ matrix (also a column matrix with P rows) is what we want to solve for, so the only matrix left to build is the $[X]$ matrix, which is a square matrix with dimensions $P \times P$.

Once all the data is read, the program begins a nested for-loop, with depth equal to the number of variables (four in our previous examples) plus one for the set of points. Each set of points is raised to its appropriate powers and placed in its corresponding column. When a row is complete, the program moves down to the next row and the next point. After all points have been processed, the matrix equation is ready for solving.

This is the main step in the program. After this step is complete, all that is

left to do is to input the matrix equation into an appropriate solver and write the output to a file. The source code for the four variable summarizer and interpolator functions is found in Appendix A.

3.6 Solving the System of Linear Equations

To solve matrix $[X]$, a matrix solver for dense, non-symmetric matrices that can accommodate large matrix size may be employed. The solution vector $[A]$, i.e., the coefficients, are listed in the output in an order of $Y[0] \rightarrow a[0][0][0][0], Y[1] \rightarrow a[0][0][0][1], \cdots, Y[P] \rightarrow a[i_m][j_m][k_m][l_m]$.

For cases when the matrix size exceeds 1000×1000, it is recommended to find a more suitable algorithm for computational efficiency. The Bi-Conjugate Gradient Stability (BICGSTAB) algorithm [Liu and Li 2002] is reliable and time efficient for large matrices; hence, ideal for this situation.

3.7 Output Files

The output file generated by the program has the same name as the input file, but is of type *.OUT. The beginning of the output file is essentially a copy of the input file (with the exception that the interpolator output file writes the input coefficients horizontally instead of vertically) with the output appended to the end of the file.

3.7.1 *Summarizer Output Files.* Summarizer output starts appending data by adding $x = \ldots$ followed by the corresponding value in the $[Y]$ matrix, and then the entire $[X]$ matrix. Summarizer output files become very large as the number of variables and value of the polynomial terms increase. This is understandable because the complexity of the governing equation is rapidly increasing. For a sample summarizer output file, see Appendix B.1.

3.7.2 *Interpolator Output Files.* Interpolator output starts appending data by writing the resultant output data. There is no preamble written into the file like in the summarizer. Data is written into the output file by each vector's number and its corresponding value followed by the resultant value. For a sample interpolator output file, see Appendix B.2.

The interpolator has much less work to do than the summarizer. Solving large matrices is processor intensive and takes a significant amount of time to finish. The interpolator does not have to solve any matrices. It only has to use the data it is given in an appropriate fashion. Knowing this, it is easy to see why interpolator output files are much smaller for equal complexity.

4. RESULTS AND DISCUSSIONS

To test the method and the implementation, a set of propeller propulsive performance data was used. Figure 2 shows this set of data with 4 variables. They are N, the nozzle type variable with $N = 1, 2, 3$ standing for nozzle type of No. 19A, No. 22 and No. 37; EAR, the expanded area ratio variable with EAR =0.5, 0.6 and 0.7; and nominal pitch p/D at 1.0, 1.1, 1.2 and 1.3, respectively[Yossifov et al. 1989]. As the K_T versus J curve for different pitch values are plotted on the same chart, the number of total charts is $3_{EAR} \times 3_{nozzle-type} = 9$.

The code produces accurate results with the range of the independent variables. To test this accuracy, the data from Figure 2 was entered into the summarizer for

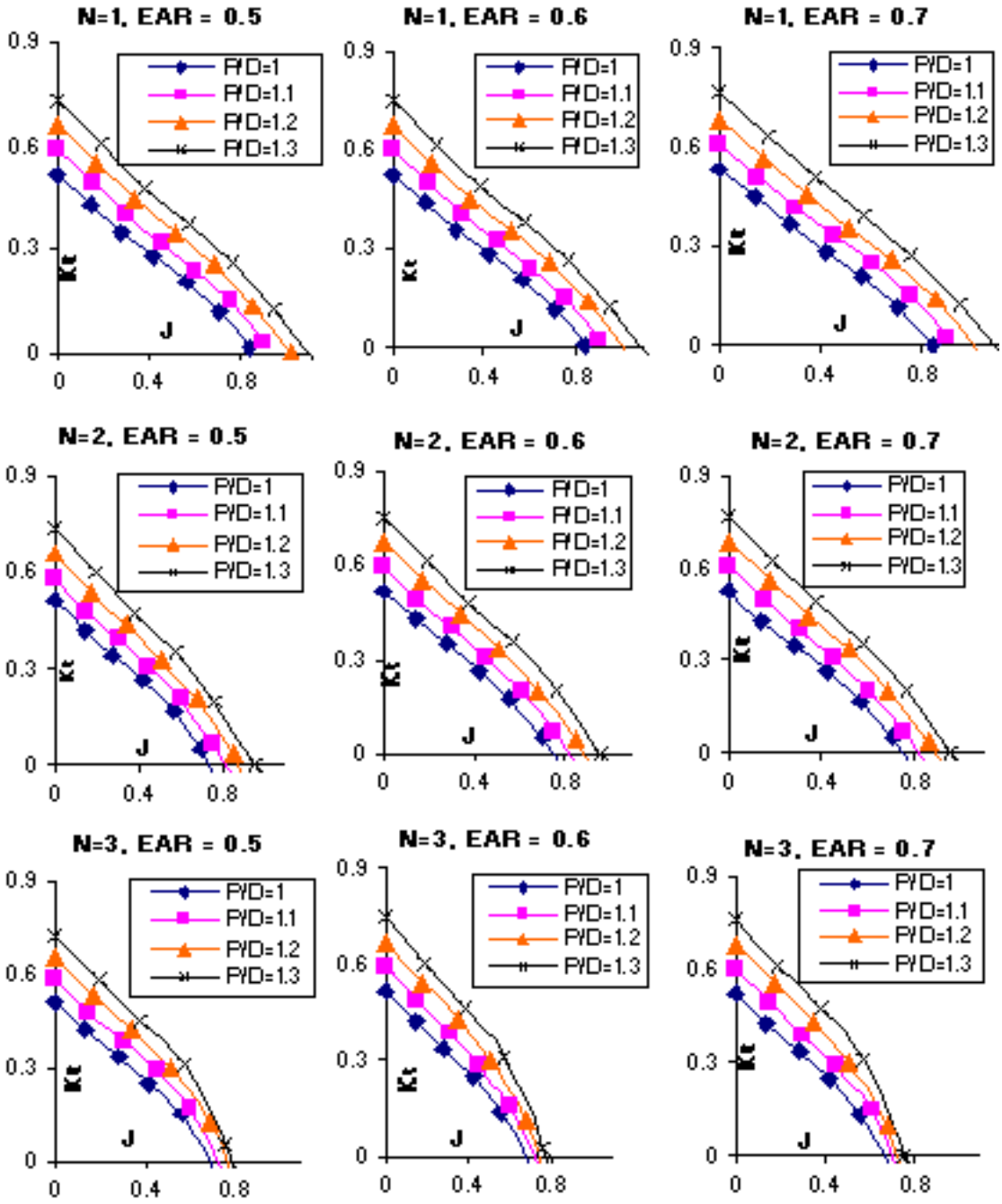

Fig. 2. Graph of K_T vs. J for various nozzles, and values of $\frac{P}{D}$ and EAR.

which trend coefficients were generated. Then, these new coefficients and the same points of the original graph were entered into the interpolator and the results were compared on a percent error, via:

$$\%Err = \left| \frac{K_{T1}(J) - K_{T2}(J)}{K_{T1}(J=0)} \right| \times 100\%, \tag{7}$$

where K_{T1} is the value of the thrust coefficient from the original graph and K_{T2} is the value of the thrust coefficient using the new coefficients. $K_{T1}(J=0)$ was always used as a reference (varying for different curves) for the percent error because it will never equal zero for any real K_T curve, thus avoiding the problem of division by zero.

Equation 7 was applied to every point to generate new curves and then these new curves were graphed (see Figure 3). These new curves were found to have percent deviations from the original curves of less than one-tenth of one percent (deviations of approximately the same magnitude were found for K_Q curves). Taking into consideration experimental error, this tiny deviation is insignificant and has no real impact upon the results or their usefulness in design.

The summarizer generates coefficients based on the trend in the set of input points and their corresponding values. These coefficients are specific to the number of terms used to generate them and the lower and upper boundaries of the independent variables used. Varying the number of terms or the boundaries of the independent variables creates the need to generate a new set of coefficients because the existing set will no longer be valid.

The interpolator uses the given coefficients to generate values for the supplied data points. These sets of points do not have to have particular order as each is treated independently using the trend. Interpolator output values have no correlation amongst themselves. Varying values of N (or Z), EAR, $\frac{P}{D}$, and J can be all mixed together and will not effect the outcome of each result. This is useful because it allows the user to set up the order of variables exactly as they will be used so reordering afterward is unnecessary.

5. CONCLUSION

A data linear regression and interpolation tool was created and a program, DGSI, was developed. This direct solution model to solve the system of linear equations directly was formulated and implemented in a C function. With a consideration of computing efficiency and propeller model cost, the method and implementation yielded a high degree of accuracy with a maximum error of 0.04%. The example showed that this tool is efficient and robust for the generation of polynomial coefficients and the interpolation. The source code in C and sample input and output ASCII files are also provided.

Nomenclature

K_T	→	total thrust coefficient of a ducted propeller
K_Q	→	torque coefficient

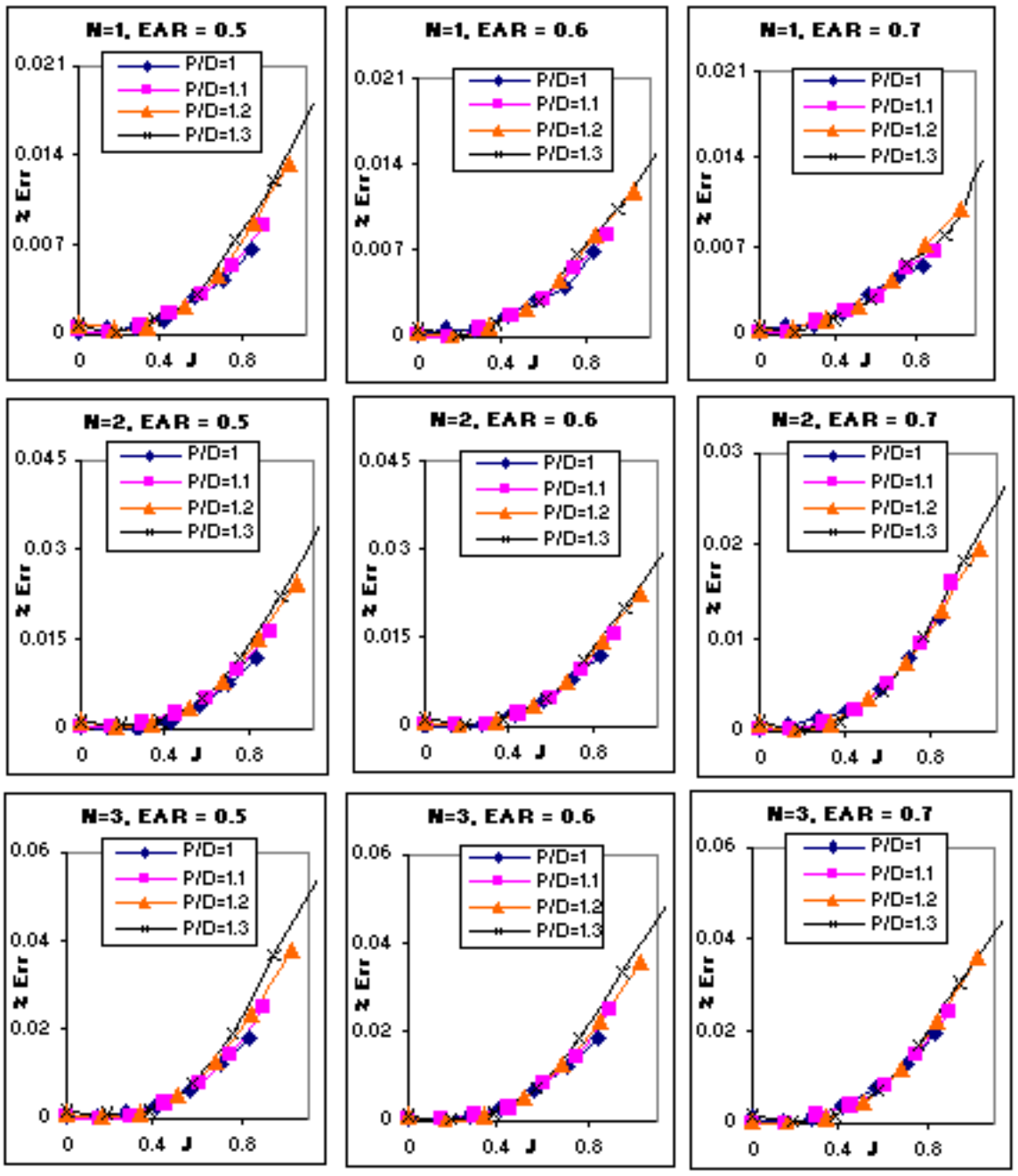

Fig. 3. Graphs of Percent Error in K_T vs. J

Z → number of blades

EAR → blade area ratio of a screw propeller
or expanded area ratio

$\frac{P}{D}$ → pitch ratio of a screw propeller

J → advance coefficient

N → Netherlands Ship Model Basin (NSMB) nozzle type and $\frac{L}{D}$:

$N = 1$ refers to NSMB 19A and $\frac{L}{D} = 0.5$
$N = 2$ refers to NSMB 22 and $\frac{L}{D} = 0.8$
$N = 3$ refers to NSMB 37 and $\frac{L}{D} = 0.5$

B_p → delivered horsepower coefficient

B_U → thrust horsepower coefficient

δ → speed coefficient

η_o → open water efficiency

ACKNOWLEDGMENTS

The authors thanks to National Research Council Canada for its support.

REFERENCES

Liu, P. and Li, K. 2002. Programming the Bi-CGSTAB Matrix Slover for HPC and Benchmarking IBM SP3 and Alpha ES40. *16th International Parallel & Distributed Processing Symposium, Ford Lauderdale, Frolida*, 269–317.

Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. 1999. *Numerical Recipes in C, The Art of Scientific Computing*, 2 ed. The Press Syndicate of the University of Cambridge, Cambridge, United Kingdom. 994 p.

Yossifov, K., Staneva, A., and Belchev, V. 1989. Equations for Hydrodynamic and Optimum Efficiency Characteristics of the Wageningen K_c Ducted Propeller Series. *Fourth International Symposium on Practical Design of Ships and Mobile Units*, 1–15.

APPENDIX A

DGSI Four Variable Function Source Code

APPENDIX

A. DGSI FOUR VARIABLE FUNCTION SOURCE CODE

A.1 Summarizer Function

```
void On4Variables()
{
char dummy[MAXSTR],dummys='\n';
int j1,j2,j3,j4,jT,k1,k2,k3,k4,kT;
int NVar,I1,I2,I3,I4,NVector;
float **ai,**x,*v1,*v2,*v3,*v4,term1,term2,term3,term4;

FILE* Input_file;
FILE* Output_file;

Input_file=fopen("V04_Sum_Real.04v","r");
Output_file=fopen("V04_Sum_Real.OUT","w");

if (!feof(Input_file))
{
//get setup info from input file
//and write to output file
fgets(dummy,MAXSTR,Input_file);
fprintf(Output_file,"%s",dummy);

fscanf(Input_file,"%d ",&NVar);
fprintf(Output_file,"%d\n",NVar);

fgets(dummy,MAXSTR,Input_file);
fprintf(Output_file,"%s",dummy);

fscanf(Input_file,"%d %d %d %d ",&I1,&I2,&I3,&I4);
fprintf(Output_file,"%d %d %d %d\n",I1,I2,I3,I4);

fgets(dummy,MAXSTR,Input_file);
fprintf(Output_file,"%s",dummy);


//dimension size of array to be created
NVector=I1*I2*I3*I4;
//ai is NVector x NVector matrix
ai=matrix(1,NVector,1,NVector);
//x is column matrix
x=matrix(1,NVector,1,1);
//vectors for each variables data
v1=vector(1,NVector);
v2=vector(1,NVector);
v3=vector(1,NVector);
v4=vector(1,NVector);

for (j1=1; j1<=I1; j1++)
for (j2=1; j2<=I2; j2++)
for (j3=1; j3<=I3; j3++)
for (j4=1; j4<=I4; j4++)
{
//index for vectors and matrix
jT=(j1-1)*I2*I3*I4 + (j2-1)*I3*I4 + (j3-1)*I4 + j4;

//gather data for vectors
fscanf(Input_file,"%f ",&v1[jT]);
fprintf(Output_file,"%f ",v1[jT]);

fscanf(Input_file,"%f ",&v2[jT]);
fprintf(Output_file,"%f ",v2[jT]);

fscanf(Input_file,"%f ",&v3[jT]);
fprintf(Output_file,"%f ",v3[jT]);

fscanf(Input_file,"%f ",&v4[jT]);
```

```
fprintf(Output_file,"%f ",v4[jT]);

fscanf(Input_file,"%f \n",&x[jT][1]);
fprintf(Output_file,"%f \n",x[jT][1]);
}
}

fprintf(Output_file,"\n");
fclose(Input_file);

for (j1=1; j1<=I1; j1++)
for (j2=1; j2<=I2; j2++)
for (j3=1; j3<=I3; j3++)
for (j4=1; j4<=I4; j4++)
{
//index for vectors and matrix
jT=(j1-1)*I2*I3*I4 + (j2-1)*I3*I4 + (j3-1)*I4 + j4;

//output column matrix
fprintf(Output_file,"x=%f \n",x[jT][1]);
}



//fill ai matrix
for (kT=1; kT<=NVector; kT++)
{
for (j1=1; j1<=I1; j1++)
for (j2=1; j2<=I2; j2++)
for(j3=1; j3<=I3; j3++)
for(j4=1; j4<=I4; j4++)
{
//index for vectors and matrix
jT=(j1-1)*I2*I3*I4 + (j2-1)*I3*I4 + (j3-1)*I4 + j4;

//get values for each term of product
if(j1==1)
term1=1.0;
else
term1=pow(v1[kT],float(j1-1));

if(j2==1)
term2=1.0;
else
term2=pow(v2[kT],float(j2-1));

if(j3 == 1)
term3 = 1.0;
else
term3 = pow(v3[kT],float(j3-1));

if(j4 == 1)
term4 = 1.0;
else
term4 = pow(v4[kT],float(j4-1));


ai[kT][jT]=term1*term2*term3*term4;
}
}

//write ai to output file
for (j1=1; j1<=NVector; j1++)
{
for (j2=1; j2<=NVector; j2++)
{
fprintf(Output_file,"%f ",ai[j1][j2]);
}
```

```
fprintf(Output_file,"\n");
}

//solve created matrix equation [ai][k]=[x]
//using gauss elimination
gaussj(ai,NVector,x,1);

fprintf(Output_file,"\n");
fprintf(Output_file,"Output ai[][] after gauss \n");

//output new ai
for (j1=1; j1<=NVector; j1++)
{
for (j2=1; j2<=NVector; j2++)
{
fprintf(Output_file,"%f ",ai[j1][j2]);
}

fprintf(Output_file,"\n");
}

fprintf(Output_file,"Solution for coefficients \n");

//output answer matrix
for (j1=1; j1<=I1; j1++)
for (j2=1; j2<=I2; j2++)
for(j3=1; j3<=I3; j3++)
for(j4=1; j4<=I4; j4++)
{
//index for vectors and matrix
jT=(j1-1)*I2*I3*I4 + (j2-1)*I3*I4 + (j3-1)*I4 + j4;

k1=j1-1;
k2=j2-1;
k3=j3-1;
k4=j4-1;

fprintf(Output_file,"k1=%d k2=%d k3=%d k4=%d a[%d][%d][%d][%d]=a[%d] %f \n",
k1,k2,k3,k4,k1,k2,k3,k4,jT,x[jT][1]);
}


//clean up:
//free allocated space and close open files
free_matrix(x,1,NVector,1,1);
free_matrix(ai,1,NVector,1,NVector);
free_vector(v1,1,NVector);
free_vector(v2,1,NVector);
free_vector(v3,1,NVector);
free_vector(v4,1,NVector);

fclose(Output_file);

}
```

A.2 Interpolator Function

```
void OnInter04()
{

char dummy[MAXSTR],dummys='\n';
int j1,j2,j3,j4,jT,k1;
int NVar,I1,I2,I3,I4,NVector,NInEx;
float *a,*v1,*v2,*v3,*v4,*result,sum,term1,term2,term3,term4;

FILE* In_file;
FILE* Out_file;

In_file=fopen("V04_Inter_Real.04i","r");
```

```
Out_file=fopen("V04_Inter_Real.OUT","w");

if (!feof(In_file))
{
//get setup info from input file
//and write to output file
fgets(dummy,MAXSTR,In_file);
fprintf(Out_file,"%s",dummy);

fscanf(In_file,"%d ",&NVar);
fprintf(Out_file,"%d\n",NVar);

fgets(dummy,MAXSTR,In_file);
fprintf(Out_file,"%s",dummy);

fscanf(In_file,"%d %d %d %d ",&I1,&I2,&I3,&I4);
fprintf(Out_file,"%d %d %d %d\n",I1,I2,I3,I4);

fgets(dummy,MAXSTR,In_file);
fprintf(Out_file,"%s\n",dummy);

//setup vector size
NVector=I1*I2*I3*I4;
a=vector01(1,NVector);

for (j1=1; j1<=I1; j1++)
for (j2=1; j2<=I2; j2++)
for (j3=1; j3<=I3; j3++)
for(j4=1; j4<=I4; j4++)
{
//index for vectors and matrix
jT=(j1-1)*I2*I3*I4 + (j2-1)*I3*I4 + (j3-1)*I4 + j4;

//fill vector and write
//values to output file
fscanf(In_file,"%f ",&a[jT]);
fprintf(Out_file,"%f ",a[jT]);
}

fprintf(Out_file,"\n\n");
fgets(dummy,MAXSTR,In_file);
fprintf(Out_file,"%s\n",dummy);

//get number of values to interpolate over
fscanf(In_file,"%d ",&NInEx);
fprintf(Out_file,"%d \n",NInEx);

v1=vector01(1,NInEx);
v2=vector01(1,NInEx);
v3=vector01(1,NInEx);
v4=vector01(1,NInEx);
result=vector01(1,NInEx);

fprintf(Out_file,"\n");
fgets(dummy,MAXSTR,In_file);
fprintf(Out_file,"%s\n",dummy);


//get values over which to interpolate
for(j1=1;j1<=NInEx;j1++)
{
fscanf(In_file,"%f %f %f %f \n",&v1[j1],&v2[j1],&v3[j1],&v4[j1]);
fprintf(Out_file,"%f %f %f %f \n",v1[j1],v2[j1],v3[j1],v4[j1]);
}

}

fprintf(Out_file,"\n");
```

```
fclose(In_file);

for (k1=1;k1<=NInEx;k1++)
{
sum=0.0;

for (j1=1; j1<=I1; j1++)
for (j2=1; j2<=I2; j2++)
for (j3=1; j3<=I3; j3++)
for(j4=1; j4<=I4; j4++)
{
//index for vectors and matrix
jT=(j1-1)*I2*I3*I4 + (j2-1)*I3*I4 + (j3-1)*I4 + j4;

//get value for each term
if(j1==1)
term1=1.0;
else
term1=pow(v1[k1],float(j1-1));

if(j2==1)
term2=1.0;
else
term2=pow(v2[k1],float(j2-1));

if(j3==1)
term3=1.0;
else
term3=pow(v3[k1],float(j3-1));

if(j4==1)
term4=1.0;
else
term4=pow(v4[k1],float(j4-1));

sum=a[jT]*term1*term2*term3*term4+sum;
}

//store sum
result[k1]=sum;
}

//output result at specified vector
for (j2=1;j2<=NInEx;j2++)
fprintf(Out_file,"v1=%f  v2=%f  v3=%f  v4=%f  result=%f \n",
v1[j2],v2[j2],v3[j2],v4[j2],result[j2]);

fprintf(Out_file,"\n");

//clean up:
//free allocated space and close open files
free_vector01(a,1,NInEx);
free_vector01(v1,1,NInEx);
free_vector01(v2,1,NInEx);
free_vector01(v3,1,NInEx);
free_vector01(v4,1,NInEx);
free_vector01(result,1,NInEx);

fclose(Out_file);
}
```

```
 0.270000 3.000000 0.900000 0.270000 1.500000 0.450000 0.135000 1.500000 0.450000 0.135000
1.000000 0.600000 0.360000 1.000000 0.600000 0.360000 0.500000 0.300000 0.180000 0.500000 0.300000 0.180000 3.000000 1.800000
 1.080000 3.000000 1.800000 1.080000 1.500000 0.900000 0.540000 1.500000 0.900000 0.540000
1.000000 0.000000 0.000000 1.100000 0.000000 0.000000 0.500000 0.000000 0.000000 0.550000 0.000000 0.000000 3.000000 0.000000
 0.000000 3.300000 0.000000 0.000000 1.500000 0.000000 0.000000 1.650000 0.000000 0.000000
1.000000 0.300000 0.090000 1.100000 0.330000 0.099000 0.500000 0.150000 0.045000 0.550000 0.165000 0.049500 3.000000 0.900000
 0.270000 3.300000 0.990000 0.297000 1.500000 0.450000 0.135000 1.650000 0.495000 0.148500
1.000000 0.600000 0.360000 1.100000 0.660000 0.396000 0.500000 0.300000 0.180000 0.550000 0.330000 0.198000 3.000000 1.800000
 1.080000 3.300000 1.980000 1.188000 1.500000 0.900000 0.540000 1.650000 0.990000 0.594000
1.000000 0.000000 0.000000 1.000000 0.000000 0.000000 0.600000 0.000000 0.000000 0.600000 0.000000 0.000000 3.000000 0.000000
 0.000000 3.000000 0.000000 0.000000 1.800000 0.000000 0.000000 1.800000 0.000000 0.000000
1.000000 0.300000 0.090000 1.000000 0.300000 0.090000 0.600000 0.180000 0.054000 0.600000 0.180000 0.054000 3.000000 0.900000
 0.270000 3.000000 0.900000 0.270000 1.800000 0.540000 0.162000 1.800000 0.540000 0.162000
1.000000 0.600000 0.360000 1.000000 0.600000 0.360000 0.600000 0.360000 0.216000 0.600000 0.360000 0.216000 3.000000 1.800000
 1.080000 3.000000 1.800000 1.080000 1.800000 1.080000 0.648000 1.800000 1.080000 0.648000
1.000000 0.000000 0.000000 1.100000 0.000000 0.000000 0.600000 0.000000 0.000000 0.660000 0.000000 0.000000 3.000000 0.000000
 0.000000 3.300000 0.000000 0.000000 1.800000 0.000000 0.000000 1.980000 0.000000 0.000000
1.000000 0.300000 0.090000 1.100000 0.330000 0.099000 0.600000 0.180000 0.054000 0.660000 0.198000 0.059400 3.000000 0.900000
 0.270000 3.300000 0.990000 0.297000 1.800000 0.540000 0.162000 1.980000 0.594000 0.178200
1.000000 0.600000 0.360000 1.100000 0.660000 0.396000 0.600000 0.360000 0.216000 0.660000 0.396000 0.237600 3.000000 1.800000
 1.080000 3.300000 1.980000 1.188000 1.800000 1.080000 0.648000 1.980000 1.188000 0.712800

Output ai[][] after gauss
197.995148 0.009359 -0.003111 -179.996429 -0.007230 0.002412 -164.996353 -0.007494 0.002415 149.997452 0.005636 -0.001792 -13
1.995163 -0.008365 0.002657 119.996262 0.006564 -0.002049 109.996078 0.006899 -0.002090 -99.997070 -0.005310 0.001567
-989.864319 1319.802490 -329.929138 899.879089 -1199.826050 299.937103 824.873901 -1099.813110 274.932404 -749.887512 999.834
839 -249.940018 659.890259 -879.843628 219.946976 -599.902832 799.862915 -199.953461 -549.896790 733.184143 -183.282379 499.9
08722 -666.536011 166.622040
1099.743896 -2199.644531 1099.873047 -999.768433 1999.682983 -999.886414 -916.427551 1832.995850 -916.544800 833.117126 -1666
.365112 833.224365 -733.130493 1466.392456 -733.241394 666.484436 -1333.090088 666.585205 610.919983 -1221.959961 611.021973
-555.384094 1110.878906 -555.476685
-179.995743 -0.008216 0.002604 179.996979 0.006181 -0.001934 149.996902 0.006463 -0.001948 -149.997940 -0.004687 0.001360 119
.995605 0.007534 -0.002292 -119.996666 -0.005797 0.001718 -99.996483 -0.006144 0.001760 99.997437 0.004611 -0.001260
899.875427 -1199.820190 299.935913 -899.889465 1199.842529 -299.943451 -749.884338 999.829712 -249.938736 749.897278 -999.850
464 249.945999 -599.898743 799.856995 -199.951889 599.910767 -799.875427 199.958084 499.904907 -666.530273 166.620392 -499.91
6321 666.548035 -166.626419
-999.762817 1999.674072 -999.884155 999.786072 -1999.710327 999.896729 833.111938 -1666.356934 833.221863 -833.133606 1666.39
0991 -833.234009 666.478088 -1333.081299 666.582703 -666.497803 1333.110718 -666.592651 -555.378052 1110.870117 -555.473999 5
55.396790 -1110.898682 555.483826
-329.998322 -0.005700 0.001301 300.000336 0.002303 -0.000159 329.999847 0.003525 -0.000535 -300.001556 -0.000545 -0.000469 21
9.997177 0.005635 -0.001123 -199.998901 -0.002799 0.000166 -219.998291 -0.003940 0.000509 199.999832 0.001424 0.000326
1649.779785 -2199.682861 549.885315 -1499.805420 1999.723877 -499.899475 -1649.796631 2199.701172 -549.891418 1499.820313 -19
99.739258 499.905060 -1099.823730 1466.419678 -366.583099 999.845398 -1333.119751 333.261230 1099.835571 -1466.432617 366.587
158 -999.856201 1333.131592 -333.264984
-1832.890137 3666.059082 -1833.114746 1666.267212 -3332.794678 1666.472412 1832.922485 -3666.094727 1833.125732 -1666.296753
3332.825684 -1666.482544 1221.874268 -2443.982666 1222.067017 -1110.799683 2221.815430 -1110.974976 -1221.897583 2444.007568
-1222.073975 1110.821289 -2221.838867 1110.981445
299.999054 0.004161 -0.000615 -300.001007 -0.000937 -0.000445 -300.000488 -0.002175 -0.000071 300.002106 -0.000658 0.001013 -
199.997726 -0.004525 0.000643 199.999374 0.001809 0.000269 199.998764 0.002954 -0.000081 -200.000259 -0.000543 -0.000711
-1499.800293 1999.715332 -499.897766 1499.824463 -1999.754028 499.911072 1499.815918 -1999.731934 499.903076 -1499.838257 199
9.767822 -499.915924 999.839355 -1333.110840 333.258759 -999.859924 1333.142578 -333.269592 -999.850464 1333.122803 -333.2623
60 999.870056 -1333.153442 333.272949
1666.259521 -3332.782227 1666.469360 -1666.300659 3332.846924 -1666.492188 -1666.289551 3332.814209 -1666.478882 1666.328003
-3332.874756 1666.500977 -1110.790161 2221.802490 -1110.971313 1110.824829 -2221.854492 1110.989258 1110.812134 -2221.825439
1110.977295 -1110.845093 2221.875977 -1110.994873
-65.998535 -0.002598 0.000772 59.999077 0.001740 -0.000491 54.998966 0.001962 -0.000537 -49.999416 -0.001214 0.000293 65.9983
22 0.002671 -0.000777 -59.998795 -0.001916 0.000527 -54.998653 -0.002147 0.000573 49.999077 0.001482 -0.000356
329.946716 -439.920990 109.970612 -299.952759 399.930542 -99.973877 -274.949921 366.591095 -91.638290 249.955521 -333.266693
83.308090 -329.954468 439.933960 -109.976814 299.959808 -399.942230 99.979622 274.956787 -366.603241 91.644226 -249.961914 33
3.277893 -83.313637
-366.568176 733.194946 -366.615784 333.244446 -666.543274 333.287689 305.462769 -610.978455 305.506042 -277.694031 555.437012
 -277.733307 366.584442 -733.221008 366.627960 -333.259674 666.567200 -333.299011 -305.477570 611.003052 -305.517700 277.7079
 77 -555.459961 277.744232
59.998707 0.002297 -0.000632 -59.999226 -0.001464 0.000368 -49.999107 -0.001700 0.000416 49.999546 0.000975 -0.000180 -59.998
463 -0.002430 0.000672 59.998920 0.001693 -0.000429 49.998779 0.001933 -0.000480 -49.999191 -0.001284 0.000271
-299.951263 399.928284 -99.973434 299.957031 -399.937408 99.976532 249.954239 -333.264740 83.307632 -249.959595 333.273285 -8
3.310623 299.958099 -399.939758 99.978958 -299.963257 399.947693 -99.981644 -249.960297 333.275513 -83.312950 249.965225 -333
.283173 83.315590
333.242340 -666.540161 333.286957 -333.251495 666.554443 -333.291931 -277.692078 555.434204 -277.732513 277.700684 -555.44763
2 277.737366 -333.256989 666.563599 -333.297974 333.265198 -666.575928 333.302185 277.705475 -555.456421 277.743164 -277.7133
18 555.468384 -277.747314
110.000015 0.000444 0.000248 -100.000862 0.000940 -0.000715 -110.000519 0.000256 -0.000481 100.001236 -0.001475 0.000889 -109
.999290 -0.001036 -0.000078 100.000031 -0.000159 0.000484 109.999680 0.000453 0.000287 -100.000343 0.000608 -0.000639
-549.913025 733.205261 -183.285019 499.923553 -666.555481 166.624252 549.918884 -733.211609 183.287125 -499.928650 666.560730
 -166.626114 549.926270 -733.227783 183.296097 -499.935608 666.575623 -166.634445 -549.930664 733.232849 -183.297684 499.9395
 45 -666.580139 166.635925
610.940308 -1221.985229 611.023010 -555.402039 1110.901245 -555.477112 -610.951599 1221.997192 -611.026428 555.412292 -1110.9
11743 555.480286 -610.969604 1222.032349 -611.045288 555.429382 -1110.944336 555.497803 610.978394 -1222.041748 611.047729 -5
55.437439 1110.953125 -555.500061
-100.000320 0.000132 -0.000492 100.001129 -0.001456 0.000933 100.000793 -0.000767 0.000703 -100.001480 0.001930 -0.001087 99.
999535 0.000611 0.000257 -100.000252 0.000541 -0.000641 -99.999901 -0.000073 -0.000447 100.000542 -0.000949 0.000774
499.921478 -666.552124 166.623596 -499.931427 666.568115 -166.629166 -499.926849 666.557861 -166.625381 499.936127 -666.57281
5 166.630768 -499.932861 666.571655 -166.633347 499.941772 -666.585510 166.638123 499.937012 -666.576294 166.634750 -499.9455
26 666.589661 -166.639420
-555.398987 1110.896729 -555.476074 555.415283 -1110.922241 555.485168 555.409485 -1110.907471 555.478943 -555.424866 1110.93
1641 -555.487854 555.425171 -1110.938721 555.496216 -555.439697 1110.960571 -555.503784 -555.433472 1110.947388 -555.498291 5
55.447388 -1110.968750 555.505798
Solution for coefficients
k1=0 k2=0 k3=0 k4=0 a[0][0][0][0]=a[1] 0.000000
k1=0 k2=0 k3=0 k4=1 a[0][0][0][1]=a[2] -0.000025
k1=0 k2=0 k3=0 k4=2 a[0][0][0][2]=a[3] -0.449930
k1=0 k2=0 k3=1 k4=0 a[0][0][1][0]=a[4] 0.500000
```

APPENDIX B

Output Files

B. OUTPUT FILES

The following samples have been wrapped to fit the page and in the process some numbers and words may have been spilt, having beginnings on one line and endings on another. The ASCII output files are not word wrapped.

B.1 Sample Summarizer Output File

```
Number of variables
4
Number of polynomial terms for each variable
2 2 2 3
List variables and results, at least 2x2x2x3=24 lines for 4 variables Z, EAR, P/D and J vs. Kt
2.000000 0.500000 1.000000 0.000000 0.550000
2.000000 0.500000 1.000000 0.300000 0.505000
2.000000 0.500000 1.000000 0.600000 0.370000
2.000000 0.500000 1.100000 0.000000 0.600000
2.000000 0.500000 1.100000 0.300000 0.555000
2.000000 0.500000 1.100000 0.600000 0.420000
2.000000 0.600000 1.000000 0.000000 0.560000
2.000000 0.600000 1.000000 0.300000 0.514100
2.000000 0.600000 1.000000 0.600000 0.376400
2.000000 0.600000 1.100000 0.000000 0.610000
2.000000 0.600000 1.100000 0.300000 0.564100
2.000000 0.600000 1.100000 0.600000 0.426400
3.000000 0.500000 1.000000 0.000000 0.550000
3.000000 0.500000 1.000000 0.300000 0.502750
3.000000 0.500000 1.000000 0.600000 0.361000
3.000000 0.500000 1.100000 0.000000 0.600000
3.000000 0.500000 1.100000 0.300000 0.552750
3.000000 0.500000 1.100000 0.600000 0.411000
3.000000 0.600000 1.000000 0.000000 0.560000
3.000000 0.600000 1.000000 0.300000 0.511400
3.000000 0.600000 1.000000 0.600000 0.365600
3.000000 0.600000 1.100000 0.000000 0.610000
3.000000 0.600000 1.100000 0.300000 0.561400
3.000000 0.600000 1.100000 0.600000 0.415600

x=0.550000
x=0.505000
x=0.370000
x=0.600000
x=0.555000
x=0.420000
x=0.560000
x=0.514100
x=0.376400
x=0.610000
x=0.564100
x=0.426400
x=0.550000
x=0.502750
x=0.361000
x=0.600000
x=0.552750
x=0.411000
x=0.560000
x=0.511400
x=0.365600
x=0.610000
x=0.561400
x=0.415600
1.000000 0.000000 0.000000 1.000000 0.000000 0.000000 0.500000 0.000000 0.000000 0.500000 0.000000 0.000000 2.000000 0.000000
 0.000000 2.000000 0.000000 0.000000 1.000000 0.000000 0.000000 1.000000 0.000000 0.000000
1.000000 0.300000 0.090000 1.000000 0.300000 0.090000 0.500000 0.150000 0.045000 0.500000 0.150000 0.045000 2.000000 0.600000
 0.180000 2.000000 0.600000 0.180000 1.000000 0.300000 0.090000 1.000000 0.300000 0.090000
1.000000 0.600000 0.360000 1.000000 0.600000 0.360000 0.500000 0.300000 0.180000 0.500000 0.300000 0.180000 2.000000 1.200000
 0.720000 2.000000 1.200000 0.720000 1.000000 0.600000 0.360000 1.000000 0.600000 0.360000
1.000000 0.000000 0.000000 1.100000 0.000000 0.000000 0.500000 0.000000 0.000000 0.550000 0.000000 0.000000 2.000000 0.000000
 0.000000 2.200000 0.000000 0.000000 1.000000 0.000000 0.000000 1.100000 0.000000 0.000000
1.000000 0.300000 0.090000 1.100000 0.330000 0.099000 0.500000 0.150000 0.045000 0.550000 0.165000 0.049500 2.000000 0.600000
 0.180000 2.200000 0.660000 0.198000 1.000000 0.300000 0.090000 1.100000 0.330000 0.099000
1.000000 0.600000 0.360000 1.100000 0.660000 0.396000 0.500000 0.300000 0.180000 0.550000 0.330000 0.198000 2.000000 1.200000
 0.720000 2.200000 1.320000 0.792000 1.000000 0.600000 0.360000 1.100000 0.660000 0.396000
1.000000 0.000000 0.000000 1.000000 0.000000 0.000000 0.600000 0.000000 0.000000 0.600000 0.000000 0.000000 2.000000 0.000000
 0.000000 2.000000 0.000000 0.000000 1.200000 0.000000 0.000000 1.200000 0.000000 0.000000
1.000000 0.300000 0.090000 1.000000 0.300000 0.090000 0.600000 0.180000 0.054000 0.600000 0.180000 0.054000 2.000000 0.600000
 0.180000 2.000000 0.600000 0.180000 1.200000 0.360000 0.108000 1.200000 0.360000 0.108000
1.000000 0.600000 0.360000 1.000000 0.600000 0.360000 0.600000 0.360000 0.216000 0.600000 0.360000 0.216000 2.000000 1.200000
 0.720000 2.000000 1.200000 0.720000 1.200000 0.720000 0.432000 1.200000 0.720000 0.432000
1.000000 0.000000 0.000000 1.100000 0.000000 0.000000 0.600000 0.000000 0.000000 0.660000 0.000000 0.000000 2.000000 0.000000
 0.000000 2.200000 0.000000 0.000000 1.200000 0.000000 0.000000 1.320000 0.000000 0.000000
1.000000 0.300000 0.090000 1.100000 0.330000 0.099000 0.600000 0.180000 0.054000 0.660000 0.198000 0.059400 2.000000 0.600000
 0.180000 2.200000 0.660000 0.198000 1.200000 0.360000 0.108000 1.320000 0.396000 0.118800
1.000000 0.600000 0.360000 1.100000 0.660000 0.396000 0.600000 0.360000 0.216000 0.660000 0.396000 0.237600 2.000000 1.200000
 0.720000 2.200000 1.320000 0.792000 1.200000 0.720000 0.432000 1.320000 0.792000 0.475200
1.000000 0.000000 0.000000 1.000000 0.000000 0.000000 0.500000 0.000000 0.000000 0.500000 0.000000 0.000000 3.000000 0.000000
 0.000000 3.000000 0.000000 0.000000 1.500000 0.000000 0.000000 1.500000 0.000000 0.000000
1.000000 0.300000 0.090000 1.000000 0.300000 0.090000 0.500000 0.150000 0.045000 0.500000 0.150000 0.045000 3.000000 0.900000
```

```
k1=0 k2=0 k3=1 k4=1 a[0][0][1][1]=a[5] 0.000022
k1=0 k2=0 k3=1 k4=2 a[0][0][1][2]=a[6] -0.000064
k1=0 k2=1 k3=0 k4=0 a[0][1][0][0]=a[7] 0.100000
k1=0 k2=1 k3=0 k4=1 a[0][1][0][1]=a[8] 0.000041
k1=0 k2=1 k3=0 k4=2 a[0][1][0][2]=a[9] -0.000118
k1=0 k2=1 k3=1 k4=0 a[0][1][1][0]=a[10] 0.000000
k1=0 k2=1 k3=1 k4=1 a[0][1][1][1]=a[11] -0.000034
k1=0 k2=1 k3=1 k4=2 a[0][1][1][2]=a[12] 0.000106
k1=1 k2=0 k3=0 k4=0 a[1][0][0][0]=a[13] 0.000000
k1=1 k2=0 k3=0 k4=1 a[1][0][0][1]=a[14] 0.000007
k1=1 k2=0 k3=0 k4=2 a[1][0][0][2]=a[15] -0.000022
k1=1 k2=0 k3=1 k4=0 a[1][0][1][0]=a[16] -0.000000
k1=1 k2=0 k3=1 k4=1 a[1][0][1][1]=a[17] -0.000006
k1=1 k2=0 k3=1 k4=2 a[1][0][1][2]=a[18] 0.000020
k1=1 k2=1 k3=0 k4=0 a[1][1][0][0]=a[19] -0.000000
k1=1 k2=1 k3=0 k4=1 a[1][1][0][1]=a[20] -0.000012
k1=1 k2=1 k3=0 k4=2 a[1][1][0][2]=a[21] -0.049963
k1=1 k2=1 k3=1 k4=0 a[1][1][1][0]=a[22] 0.000000
k1=1 k2=1 k3=1 k4=1 a[1][1][1][1]=a[23] 0.000010
k1=1 k2=1 k3=1 k4=2 a[1][1][1][2]=a[24] -0.000033
```

B.2 Sample Interpolator Output File

```
Number of variables
4
Number of polynomial terms for each variable
2 2 2 3
List coefficient a[1-2][1-2][1-2][1-3]=>a[0][0][0][0], ..., a[1][1][1][2]

0.000000 -0.000025 -0.449930 0.500000 0.000022 -0.000064 0.100000 0.000041 -0.000118 0.000000 -0.000034 0.000106 0.000000
0.000007 -0.000022 0.000000 -0.000006 0.000020 0.000000 -0.000012 -0.049963 0.000000 0.000010 -0.000033

Number of sets of variables to inter-extrapolate

24

List of this 24 sets of variables

2.000000 0.500000 1.000000 0.000000
2.000000 0.500000 1.000000 0.300000
2.000000 0.500000 1.000000 0.600000
2.000000 0.500000 1.100000 0.000000
2.000000 0.500000 1.100000 0.300000
2.000000 0.500000 1.100000 0.600000
2.000000 0.600000 1.000000 0.000000
2.000000 0.600000 1.000000 0.300000
2.000000 0.600000 1.000000 0.600000
2.000000 0.600000 1.100000 0.000000
2.000000 0.600000 1.100000 0.300000
2.000000 0.600000 1.100000 0.600000
3.000000 0.500000 1.000000 0.000000
3.000000 0.500000 1.000000 0.300000
3.000000 0.500000 1.000000 0.600000
3.000000 0.500000 1.100000 0.000000
3.000000 0.500000 1.100000 0.300000
3.000000 0.500000 1.100000 0.600000
3.000000 0.600000 1.000000 0.000000
3.000000 0.600000 1.000000 0.300000
3.000000 0.600000 1.000000 0.600000
3.000000 0.600000 1.100000 0.000000
3.000000 0.600000 1.100000 0.300000
3.000000 0.600000 1.100000 0.600000

v1=2.000000  v2=0.500000  v3=1.000000  v4=0.000000  result=0.550000
v1=2.000000  v2=0.500000  v3=1.000000  v4=0.300000  result=0.505000
v1=2.000000  v2=0.500000  v3=1.000000  v4=0.600000  result=0.370000
v1=2.000000  v2=0.500000  v3=1.100000  v4=0.000000  result=0.600000
v1=2.000000  v2=0.500000  v3=1.100000  v4=0.300000  result=0.555000
v1=2.000000  v2=0.500000  v3=1.100000  v4=0.600000  result=0.420000
v1=2.000000  v2=0.600000  v3=1.000000  v4=0.000000  result=0.560000
v1=2.000000  v2=0.600000  v3=1.000000  v4=0.300000  result=0.514100
v1=2.000000  v2=0.600000  v3=1.000000  v4=0.600000  result=0.376400
v1=2.000000  v2=0.600000  v3=1.100000  v4=0.000000  result=0.610000
v1=2.000000  v2=0.600000  v3=1.100000  v4=0.300000  result=0.564100
v1=2.000000  v2=0.600000  v3=1.100000  v4=0.600000  result=0.426400
v1=3.000000  v2=0.500000  v3=1.000000  v4=0.000000  result=0.550000
v1=3.000000  v2=0.500000  v3=1.000000  v4=0.300000  result=0.502750
v1=3.000000  v2=0.500000  v3=1.000000  v4=0.600000  result=0.361000
v1=3.000000  v2=0.500000  v3=1.100000  v4=0.000000  result=0.600000
v1=3.000000  v2=0.500000  v3=1.100000  v4=0.300000  result=0.552750
v1=3.000000  v2=0.500000  v3=1.100000  v4=0.600000  result=0.411000
v1=3.000000  v2=0.600000  v3=1.000000  v4=0.000000  result=0.560000
v1=3.000000  v2=0.600000  v3=1.000000  v4=0.300000  result=0.511400
v1=3.000000  v2=0.600000  v3=1.000000  v4=0.600000  result=0.365600
v1=3.000000  v2=0.600000  v3=1.100000  v4=0.000000  result=0.610000
v1=3.000000  v2=0.600000  v3=1.100000  v4=0.300000  result=0.561400
v1=3.000000  v2=0.600000  v3=1.100000  v4=0.600000  result=0.415600
```