COMPUTER METHODS IN APPROXIMATION

Jørgen Kjær

Haldor Topsøe, Vedbæk, Denmark

1970

ISBN 87 500 1088 3
Production Akademisk Forlag
Printing Zano Print

PREFACE

The present book is a continuation of the just published book on: Computer Methods in Linear and Quadratic Models, which covered the topics: Solution of linear and non-linear equations, generation of linear and quadratic models, and optimization. We now consider the two related fields: Linear regression analysis and approximation of functions by means of orthogonal polynomials.

An elementary introduction is given to the methods involved in the standard linear regression analysis. Various special features have been investigated in collaboration with Regnecentralen, A/S, Copenhagen. The problems considered were: Preliminary centering of the observations around the first observation with a later adjustment to centering around the mean values, and the question of solving the normal equations directly instead of using matrix inversion.

The use of orthogonal polynomials for approximation of functions at the Haldor Topsøe computer installation was started on the computer DASK, where we had access to a program of this type for functions of a single variable, developed in machine language by Mr. P. Mondrup, Regnecentralen. From 1961 a similar program was made by us for the DASK computer and later rewritten for the GIER computer.

In 1961 Mr. J. Adriansen of Haldor Topsøe made a method description of the use of orthogonal polynomials in approximation of functions of two variables. ALGOL programs using this method were made for DASK and GIER. In 1963 I extended the method to functions of N variables where N is input to the program at run time. These programs were made for GIER.

On the basis of the ALGOL procedures described in this book, similar FORTRAN subroutines have been written by Mrs. Lissen Knudsen (regression analysis) and Mr. G. Johansen (polynomial approximation). These programs are now in use in the GIPS system at the Haldor Topsøe IBM system 360/44 computer installation.

Vedbæk, October 1970

Jørgen Kjær

CONTENTS

1. INTRODUCTION

The book is divided into two parts: Part 1 on linear regression analysis and Part 2 on orthogonal polynomial approximation.

Linear regression analysis is the standard designation of methods for approximation of one or more functions of one or more variables by means of linear functions and using more observations than strictly necessary to give an exact fit in the known points. This is in contrast to the model generation methods, see Kjær (1970), in which the minimum number of observations is used.

The basic principle is first illustrated in Chapter 2 which describes the regression analysis of a function of a single variable. In this case the calculation is very simple and can be performed in one scanning of the observations.

Chapter 3 describes the regression analysis of functions of several variables. The essential part is the establishment of the normal equations and their solution. The standard procedure for this purpose, FIT7, is discussed.

Linear regression analysis can be extended to the non-linear case, if higher powers of the original variables are included as pseudo variables. Examples are given of this method in Chapter 4.

Various special features in the regression analysis have been investigated in collaboration with Regnecentralen, A/S, Copenhagen. Some of the results obtained are discussed in Chapter 5. The two points considered are: Preliminary centering of the observations around the first observation with a later adjustment to centering around the mean values, and the question of solving the normal equations directly instead of using matrix inversion. The investigation shows that centering around the first observation can have a favorable - although small - numerical effect. Matrix inversion can give very poor results and is therefore not recommended.

The centering around the first observation can be used immediately, if it is required that the linear approximation passes exactly through the first observation.

Certain scaling problems encountered when the ALGOL procedures are translated into FORTRAN are breifly explained.

The special methods involving step-wise regression analysis are not discussed here.

The use of orthogonal polynomials for approximation of functions of one or more variables has certain advantages compared to standard regression analysis. For functions of a single variable the orthogonal polynomials give a faster and more accurate calculation. This is also the case for more variables, but we then get the drawback that the function values must be available in a regular grid. This makes the method less attractive for general purpose approximation. Normal regression analysis is completely satisfactory when the number of coefficients is not too high.

The basic properties of the orthogonal polynomials and their generation are described in Chapter 6. The practical application of the polynomials to functions of one, two, and several variables is described in Chapters 7 to 10. Chapter 11 describes various procedures for evaluation of the polynomial values.

Chapter 12 explains how to handle the case when the approximation polynomial must satisfy certain special conditions, such as passing exactly through one or more points or having specified values of the derivative at certain points.

PART 1. LINEAR REGRESSION ANALYSIS

2. LINEAR FUNCTIONS OF A SINGLE VARIABLE

The problem at hand is illustrated by Table 1 below and Figure 1. We have a function:

$$y = f(x)$$

given as a table of corresponding values of x and y for a certain number, OBS, of observations:

x	y
300	2413
400	3323
500	4365
600	5549
700	6871
800	8321
900	9887
1000	11560
1100	13320
1200	15170
1300	17100
1400	19090
1500	21130

Table 1. $y = f(x)$.

In this table the independent variable, x, is the temperature in degrees Kelvin and y is the enthalpy of methane in kcal/kgmole. The 13 observations are shown on Figure 1 (page 13). We shall now calculate a straight line which according to some criterion gives the best possible approximation to the given observations. A line of this type is shown in Figure 1.

It is an essential assumption in this approximation method that the observations are available in a set of discrete points only. We know nothing about the function values between the observation points. Each observation may or may not be erratic, i.e. there may be a measurement error or a rounding error.

2.1. Method of Least Squares. In the method of least squares we require that the sum of the squares of the approximation errors at the 13 points must attain a minimum. The approximation error (or deviation) at a point is the difference between the y-value on the straight line and the given y-value at that point. If we divide the sum of the squares of the errors by the number of points (minus 1) and take the square root we get the mean error for all the points. The method of least squares requires minimum of the mean error.

There is another possible criterion: Minimum of the maximum error at any point. This is of special interest if it is required to reproduce an analytical function such as exp(x) or sin(x) within a given range. The exact function values are then available at any point, and it is a reasonable requirement that the maximum approximation error is a minimum or less than a specified tolerance at all points. When the function is only available at discrete points we cannot use the criterion of minimum of the maximum error, and this method is not considered further here.

2.2. Normal Equations. The analytical expression for the straight line we wish to use as the approximation may be written as:

(2.1) $Y = A + B \times X$

We assume that we have OBS observations and that the X-Y-table is given as the two arrays:

```
        array X, Y[1:OBS];
```

The approximation error in point no. I can be written as:

(2.2) $A + B \times X[I] - Y[I]$

and the square of the error:

(2.3) $(A + B \times X[I] - Y[I]) \uparrow 2$

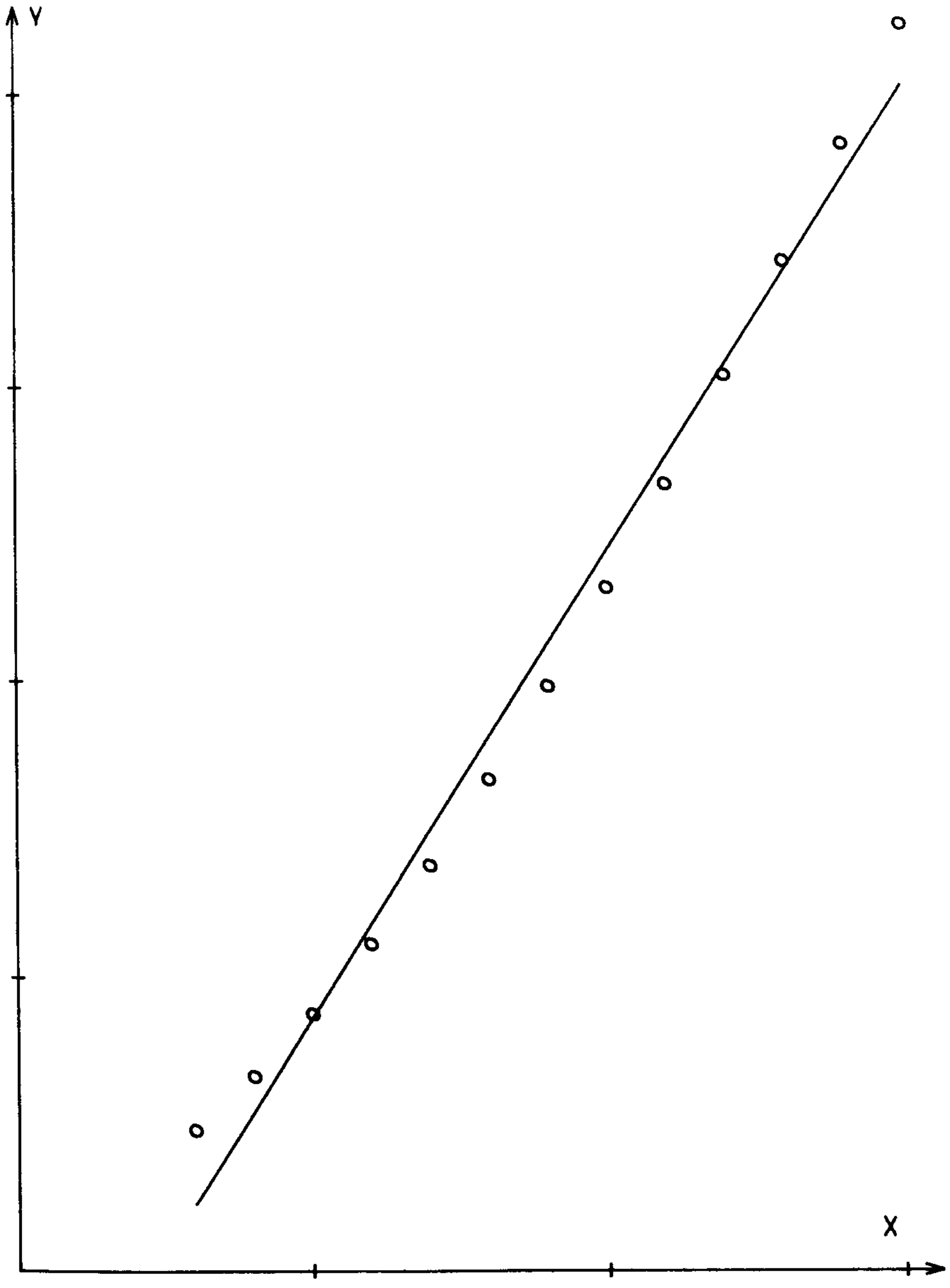

Figure 1

Linear Approximation to 13 Observations

The sum of the squares of the errors (or deviations) becomes:

$$\begin{aligned}
(2.4)\qquad \mathrm{SSD} := {} & (A + B\times X[1] - Y[1])\uparrow 2 \\
& + (A + B\times X[2] - Y[2])\uparrow 2 \\
& + \cdots\cdots \\
& \quad \cdots\cdots \\
& + (A + B\times X[\mathrm{OBS}] - Y[\mathrm{OBS}])\uparrow 2
\end{aligned}$$

As the arrays X and Y are given, SSD is a function of the two variables, A and B, only. The condition that SSD is a minimum means that the two partial derivatives of SSD with respect to A and B must be zero. Calculation of the derivatives gives:

$$\begin{aligned}
(2.5)\qquad \mathrm{dSSD/dA} = {} & 2\times(A + B\times X[1] - Y[1])\times 1 \\
& + 2\times(A + B\times X[2] - Y[2])\times 1 \\
& + \cdots\cdots \\
& \quad \cdots\cdots \\
& + 2\times(A + B\times X[\mathrm{OBS}] - Y[\mathrm{OBS}])\times 1
\end{aligned}$$

$$\begin{aligned}
(2.6)\qquad \mathrm{dSSD/dB} = {} & 2\times(A + B\times X[1] - Y[1])\times X[1] \\
& + 2\times(A + B\times X[2] - Y[2])\times X[2] \\
& + \cdots\cdots \\
& \quad \cdots\cdots \\
& + 2\times(A + B\times X[\mathrm{OBS}] - Y[\mathrm{OBS}])\times X[\mathrm{OBS}]
\end{aligned}$$

We now put the two derivatives equal to zero, divide by two, and perform the summations. The sums are written as:

$$\begin{aligned}
(2.7)\qquad & \mathrm{SX} := X[1] + X[2] + \cdots\cdots + X[\mathrm{OBS}]; \\
(2.8)\qquad & \mathrm{SX2} := X[1]\uparrow 2 + X[2]\uparrow 2 + \cdots\cdots + X[\mathrm{OBS}]\uparrow 2; \\
(2.9)\qquad & \mathrm{SY} := Y[1] + Y[2] + \cdots\cdots + Y[\mathrm{OBS}]; \\
(2.10)\qquad & \mathrm{SXY} := X[1]\times Y[1] + X[2]\times Y[2] + \cdots\cdots + X[\mathrm{OBS}]\times Y[\mathrm{OBS}];
\end{aligned}$$

The two zero conditions then become:

$$\begin{aligned}
(2.11)\qquad & A\times\mathrm{OBS} + B\times\mathrm{SX} - \mathrm{SY} = 0 \\
(2.12)\qquad & A\times\mathrm{SX} + B\times\mathrm{SX2} - \mathrm{SXY} = 0
\end{aligned}$$

From these we find A and B to:

```
(2.13)    A := (SX2×SY - SX×SXY)/DEN;
(2.14)    B := (OBS×SXY - SX×SY)/DEN;
```

in which the denominator, DEN, is:

```
(2.15)    DEN := OBS×SX2 - (SX)↑2;
```

We can also calculate the value of SSD:

```
(2.16)    SSD :=
   (A↑2+(B×X[1])↑2+Y[1]↑2+2×A×B×X[1]-2×A×Y[1]-2×B×X[1]×Y[1])
 + (A↑2+(B×X[2])↑2+Y[2]↑2+2×A×B×X[2]-2×A×Y[2]-2×B×X[2]×Y[2])
 + etc.
```

The summation gives:

```
(2.17)    SSD := OBS×A↑2+B↑2×SX2+SY2+2×A×B×SX-2×A×SY-2×B×SX×SY;
```

Here, SY2 is the sum of the Y-squares:

```
(2.18)    SY2 := Y[1]↑2 + Y[2]↑2 + ..... + Y[OBS]↑2;
```

Insertion of A and B from equations (2.13) and (2.14) into equation (2.17) gives:

```
(2.19)    SSD := SY2+(2×SX×SY×SXY-OBS×SXY↑2-SX2×SY↑2)/DEN;
```

The mean error is then found as:

```
(2.20)    mean error := sqrt(SSD/(OBS-1));
```

We must divide by OBS-1 and not by OBS, because there is one relation between X and Y.

2.3. The Procedure FIT1. This procedure performs the calculations described above using the same formulas. The declaration is:

```
procedure FIT1(OBS, mean error, A, B, X, Y);
value OBS;
integer OBS;
real mean error, A, B;
array X, Y;
begin
   integer i;
   real xi, yi, SX, SX2, SY, SXY, SY2, DEN;
   SX := SX2 := SY := SXY := SY2 := 0;
   for i := 1 step 1 until OBS do
   begin
      xi := X[i];
      yi := Y[i];
      SX := SX + xi;
      SX2 := SX2 + xi↑2;
      SY := SY + yi;
      SXY := SXY + xi×yi;
      SY2 := SY2 + yi↑2;
   end for i;
   DEN := OBS×SX2 - SX↑2;
   A := (SX2×SY-SX×SXY)/DEN;
   B := (OBS×SXY-SX×SY)/DEN;
   mean error:=sqrt((SY2+(2×SX×SY×SXY-OBS×SXY↑2-SX2×SY↑2)/DEN)/(OBS-1));
end FIT1;
```

The formal parameters are those explained above. The essential part of the procedure is a for-statement for calculation of the five sums. At the end the values of A, B, and the mean error are found.

An example of the use of FIT1 is given below in the program d-353. The program reads the table of X and Y shown on page 11 and finds the linear approximation:

```
Y := -3573.1155 + 15.773462×X;
```

This line is shown on Figure 1, page 13. The program is:

Program d-353. Test of FIT1.

```
begin
   integer i;
   real A, B, mean error, mean2, ycal, error;
   array X, Y[1: 13];
   copy FIT1<
   select(17);
   writetext(‹<
Read input to d-353:›);
   lyn;
   select(8);
   for i := 1 step 1 until 13 do
   begin
      X[i] := read real;
      Y[i] := read real;
   end for i;
   FIT1(13, mean error, A, B, X, Y);
   writetext(‹<
Output d-353
     X  YOBS  YCAL  ERROR
›);
   mean2 := 0;
   for i := 1 step 1 until 13 do
   begin
      ycal := A + B×X[i];
      error := ycal - Y[i];
      writecr;
      write(‹-ddddd›, X[i], Y[i], ycal, error);
      mean2 := mean2 + error↑2;
   end for i;
   writecr;
   writetext(‹<    A             B            mean error    mean2›);
   writecr;
   write(‹-dddd.dddd00›, A, B, mean error, sqrt(mean2/12));
   writecr;
end;
```

Output from the program is:

```
Output d-353
    X  YOBS  YCAL  ERROR


  300  2413  1159 -1254
  400  3323  2736  -587
  500  4365  4314   -51
  600  5549  5891   342
  700  6871  7468   597
  800  8321  9046   725
  900  9887 10623   736
 1000 11560 12200   640
 1100 13320 13778   458
 1200 15170 15355   185
 1300 17100 16932  -168
 1400 19090 18510  -580
 1500 21130 20087 -1043
   A              B          mean error    mean2
-3573.1155    15.773462   681.07978    681.07734
```

The program calculates and prints two mean errors. The first is the value calculated by FIT1 and the second is calculated directly by summation of the error squares. The small difference between the two values is due to rounding errors.

3. LINEAR FUNCTIONS OF SEVERAL VARIABLES

A linear function of several - say, 3 - variables may be written like this:

(3.1) $y := y0 + c1\times x1 + c2\times x2 + c3\times x3;$

The problem is now to calculate the coefficients c1, c2, and c3 as well as the constant term, y0, from a set of observations. Let us first consider an example with two variables with the material from the following table:

y	x1	x2
43.92	410	280
38.18	440	300
33.20	440	240
30.57	460	260
39.68	420	260
38.95	430	280

Here, y is the ammonia equilibrium mole per cent as a function of the temperature, x1 deg.C, and the pressure, x2 atm.abs. The six values have been selected at random from the table on page 16 in Kjær (1963).

The linear expression we must find can be written as:

(3.2) $y := y0 + c1\times x1 + c2\times x2;$

but it appears more practical from a numerical point of view to measure the variables relative to their mean values. We can then write:

(3.3) $y := ymean + c1\times(x1-xmean1) + c2\times(x2-xmean2);$

The following nomenclature is used for the observations and the calculation results:

<u>integer</u> VAR: Number of independent variables.

<u>integer</u> OBS: Number of observations.

<u>integer</u> FUNC: Number of dependent functions. In the example above we have: VAR = 2, OBS = 6, and FUNC = 1. The case of FUNC > 1 is of interest if more than one dependent function has been measured in each observation for the same sets of the independent variables.

The observation data can be presented either as two arrays:

```
yobs[1:OBS,1:FUNC]
xobs[1:OBS,1:VAR]
```

or, as a single array:

```
yxdata[1:OBS,1:FUNC+VAR]
```

Here, the first FUNC columns contain the y-values and the last VAR columns the x-values. This representation is more practical for large amounts of observations stored on a backing store.

The calculation results must be obtained as the arrays:

xmean[1:VAR]: The mean values of the independent variables.

coef[1:VAR,1:FUNC]: These are the calculated linear regression coefficients represented as FUNC columns, one for each of the dependent functions, and each column containing the VAR coefficients. In the example above we have coef[1:2,1:1] with:

```
c1 := coef[1,1];
c2 := coef[1,2];
```

It is also possible to calculate how accurate these coefficients are. This can be expressed as the array:

coeferror[1:VAR,1:FUNC]. Here, coeferror[i,j] is the standard deviation on the coefficient coef[i,j].

The constant term, ymean, in the linear expression is expanded into an array:

ymean[1:FUNC]: These are the constant terms for the FUNC functions.

3.1. Normal Equations. In the following we first consider the case of FUNC = 1 and shall write down the VAR normal equations which define that the sum of the squares of the deviations between the observed values of y and those calculated from the linear expressions is a minimum with respect to variation of the coefficients. The calculation method is taken from Fisher (1948), pag. 156.

The square of the deviation at observation point no. i can be written as:

```
(3.4)     SD := (ymean +
                 coef[1]×(xobs[i,1] - xmean[1])
               + coef[2]×(xobs[i,2] - xmean[2])
               + .....
                 .....
               + coef[VAR]×(xobs[i,VAR] - xmean[VAR]) - yobs[i])↑2
```

As we are assuming FUNC = 1, we have omitted the subscript 1:FUNC from ymean, coef, and yobs.

Summation of SD over all observations gives SSD which is too long to be written down here. The normal equation no. j expresses:

```
(3.5)     dSSD/dcoef[j] = 0
```

If we perform the squaring of the terms in equation (3.4) and insert equation (3.5), the normal equation can be established after rearrangements which will not be reproduced here. For the case of VAR = 3 and FUNC = 1 the three linear normal equations become:

```
(3.6)  coef[1]×SXX[1,1]+coef[2]×SXX[1,2]+coef[3]×SXX[1,3] = SXX[1,4]
(3.7)  coef[1]×SXX[2,1]+coef[2]×SXX[2,2]+coef[3]×SXX[2,3] = SXX[2,4]
(3.8)  coef[1]×SXX[3,1]+coef[2]×SXX[3,2]+coef[3]×SXX[3,3] = SXX[3,4]
```

The system of VAR normal equations is defined by the coefficient matrix:

```
SXX[1:VAR,1:VAR+1]
```

The element, SXX[j,k], of the square part of the matrix $(1\leq k\leq VAR)$ is found by summation of:

(3.9) (xobs[i,j]-xmean[j])×(xobs[i,k]-xmean[k])

for i = 1 to i = OBS. The right-hand side of the equation system, which is written as the column SXX[1:VAR,VAR+1] is found in a similar way by summation of the y-values. The element SXX[j,VAR+1] is found by summation of:

(3.10) (xobs[i,j]-xmean[j])×(yobs[i]-ymean)

from i = 1 to i = OBS.

When the normal equations are solved, e.g. after the LEQ1 method, we obtain the coefficients as a result of the solution. For FUNC > 1, the number of right-hand sides is increased from 1 to FUNC and the matrix SXX takes the form: SXX[1:VAR,1:VAR+FUNC]. The number of columns in the solution is then also increased from 1 to FUNC.

The calculation of the coefficient errors, coeferror[1:VAR,1:FUNC], requires that we calculate the inverse of the square matrix SXX[1:VAR, 1:VAR]. If we denote this inverse by INV[1:VAR,1:VAR], the formula is:

(3.11) coeferror[j,k] := meanerror[k]×sqrt(INV[j,j]);

The error in coefficient no. j for function no. k is found by multiplication of the mean error for function no. k by the square root of the diagonal term no. j in the inverse matrix. This is further explained in the reference cited above. The mean error is found by calculation of the linear expression for y in the given points: ycalc[1:OBS,1:FUNC] and performing the summation of:

(3.12) meanerror[k] := meanerror[k]+(ycalc[i,k]-yobs[i,k])↑2;

for i = 1 to i = OBS. We then take the square root after division by the number of degrees of freedom: OBS-VAR-1.

In order to find the coefficients themselves it is not necessary to perform the matrix inversion. We could then change the method so that we always calculate the inverse without consideration of the right-hand sides. Multiplication of the original right-hand sides: SXX[1:VAR, VAR+1:VAR+FUNC] by the inverse matrix would then yield the coefficients. However, as shown in Chapter 5 the matrix inversion gives poorer accuracy in determination of the coefficients than does the direct solution of the linear equations according to the LEQ1 method. We have, therefore, adapted the method of simultaneous direct solution and matrix inversion. The SXX-matrix which for VAR = 3 and FUNC = 1 looks like this:

```
SXX  SXX  SXX  SXX
SXX  SXX  SXX  SXX
SXX  SXX  SXX  SXX
```

is extended to include the unit matrix:

```
SXX  SXX  SXX  1    0    0    SXX
SXX  SXX  SXX  0    1    0    SXX
SXX  SXX  SXX  0    0    1    SXX
```

We then perform the direct solution of linear equations on this augmented matrix with VAR+FUNC right-hand sides. The unit matrix is then replaced by the inverse matrix and the coefficients appear as the last FUNC columns:

```
---  ---  ---  INV  INV  INV  coef
---  ---  ---  INV  INV  INV  coef
---  ---  ---  INV  INV  INV  coef
```

This means that the SXX matrix used in the method must have the size SXX[1:VAR,1:2×VAR+FUNC]. Otherwise, the calculations are the same as explained above.

The procedure LEQ1 and its use for solution of linear equation and matrix inversion is described by Kjær (1970).

3.2. The Procedure FIT7.

This procedure uses the calculation strategy described above and has the declaration:

```
integer procedure FIT7(VAR, OBS, FUNC, yxdata, xmean, coef,
coeferror, ymean, meanerror, ycalc, eps);
value VAR, OBS, FUNC, eps;
integer VAR, OBS, FUNC;
real eps;
array yxdata, xmean, coef, coeferror, ymean, meanerror, ycalc;
begin
   integer i, j, k;
   real xj;
   array SXX[1:VAR, 1:(2×VAR+FUNC)];
   for i := 1 step 1 until FUNC do
   ymean[i] := meanerror[i] := 0;
   for j := 1 step 1 until VAR do
   begin
      xmean[j] := 0;
      for k := 2×VAR+FUNC step -1 until 1 do SXX[j,k] := 0;
      SXX[j,j+VAR] := 1;
   end for j;
   for i := 1 step 1 until OBS do
   begin
      for k := 1 step 1 until FUNC do
      ymean[k] := ymean[k] + yxdata[i,k]/OBS;
      for j := 1 step 1 until VAR do
      xmean[j] := xmean[j] + yxdata[i,FUNC+j]/OBS;
   end for i;
   for i := 1 step 1 until OBS do
   for j := 1 step 1 until VAR do
   begin
      xj := yxdata[i, FUNC+j] - xmean[j];
      for k := 1 step 1 until FUNC do
      SXX[j, 2×VAR+k] := SXX[j, 2×VAR+k]
      + xj×(yxdata[i,k] - ymean[k]);
      for k := j step 1 until VAR do
      SXX[k,j] := SXX[j,k] := SXX[j,k]
      + xj×(yxdata[i,FUNC+k] - xmean[k]);
   end for j and i;
   FIT7 := i := LEQ1(VAR, VAR + FUNC, SXX, eps);
```

```
  if i = 0 then
  begin
     for j := 1 step 1 until VAR do
     for i := 1 step 1 until FUNC do
     coef[j,i] := SXX[j, 2×VAR+i];
     for i := 1 step 1 until OBS do
     begin
        for k := 1 step 1 until FUNC do
        ycalc[i,k] := ymean[k];
        for j := 1 step 1 until VAR do
        begin
           xj := yxdata[i,FUNC+j] - xmean[j];
           for k := 1 step 1 until FUNC do
           ycalc[i,k] := ycalc[i,k] + xj×coef[j,k];
        end for j;
        for k := 1 step 1 until FUNC do
        meanerror[k] := meanerror[k] + (ycalc[i,k] - yxdata[i,k])↑2;
     end for i;
     for k := 1 step 1 until FUNC do
     meanerror[k] := sqrt(meanerror[k]/(OBS-VAR-1));
     for j := 1 step 1 until VAR do
     for k := 1 step 1 until FUNC do
     coeferror[j,k] := meanerror[k]×sqrt(abs(SXX[j,VAR+j]));
  end if i = 0;
end FIT7;
```

All the formal parameters in FIT7 have been explained above, except the real eps which is the minimum pivot element accepted by LEQ1. If a pivot element becomes smaller than eps, FIT7 is set to 1 and the calculation interrupted. Otherwise, FIT7 is set to 0.

Further details of the work of the procedure are:

All elements in the arrays ymean, meanerror[1:FUNC] and xmean[1:VAR] are set to zero. The elements in the array SXX[1:VAR,1:2×VAR+FUNC] are also set to zero except the diagonal:

$$\begin{matrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{matrix}$$

The observations are then scanned for calculation of the mean values ymean and xmean. The SXX matrix is then generated in a second scanning of the observations. For each observation we have a main for-statement counting j from 1 to VAR. For each value of j the right-hand sides are calculated and another for-statement in k updates the elements of the main matrix SXX[1:VAR,1:VAR]. As this part of the matrix is symmetric, it is only necessary to calculate the upper triangle directly.

A call is then made of LEQ1 with VAR equations and VAR+FUNC right-hand sides. If there is no pivot trouble, the calculation is finished with transfer of the coefficients from the last SXX-columns to the coef-columns. The observation material is then scanned again for calculation of ycalc[1:OBS,1:FUNC], meanerror[1:FUNC], and coeferror[1:VAR,1:FUNC] as explained above.

<u>3.3. Simple Test of FIT7.</u> The program d-358 shown below reads the table on page 19, calls FIT7, and finds the linear approximation:

(3.13) $y := 116.72 - 0.23451 \times x1 + 0.08263 \times x2;$

The program is:

```
Program d-358. Test of FIT7 with 2 variables.
begin
   integer i, j;
   array yxdata[1:6, 1:3], xmean[1:2], coef, coeferror[1:2, 1:1],
   ymean, meanerror[1:1], ycalc[1:6, 1:1];
   copy FIT7<
   copy LEQ1<
   select(17);
   writetext({<
Read input to d-358:});
   lyn;
   select(8);
   for i := 1 step 1 until 6 do
   for j := 1 step 1 until 3 do
   yxdata[i, j] := read real;
   FIT7(2, 6, 1, yxdata, xmean, coef, coeferror,
   ymean, meanerror, ycalc, 1⏨-12);
   writetext(⌊<
Output d-358
  yinput  ycalc    error    x1    x2
⌋);
   for i := 1 step 1 until 6 do
   begin
      writecr;
      write(⌊-dddd.dd⌋,
      yxdata[i, 1], ycalc[i, 1], ycalc[i, 1] - yxdata[i, 1]);
      write(⌊-ddddd⌋, yxdata[i, 2], yxdata[i, 3]);
   end for i;
   writecr;
   writecr;
   writetext(⌊<y0: ⌋);
   write(⌊-ddd.ddddd⌋,
   ymean[1] - xmean[1]×coef[1,1] - xmean[2]×coef[2,1]);
   for j := 1, 2 do
   begin
      writecr;
      writetext(⌊<c⌋);
      write(⌊d⌋, j);
      writetext(⌊<: ⌋);
      write(⌊-ddd.ddddd⌋, coef[j,1]);
      writetext(⌊<  ±⌋);
      write(⌊-ddd.ddddd⌋, coeferror[j,1]);
   end for j;
   writecr;
end;
```

In this program we have corrected the calculated value of ymean by subtracting:

xmean[1]×coef[1,1]+xmean[2]×coef[2,1]

The independent variables can then be inserted directly in equation (3.13). This simplification is not recommended for large values of VAR or when higher powers of the independent variables are used as pseudo variables.

The program gave the following output:

```
Output d-358
  yinput  ycalc    error    x1     x2

   43.92  43.71   -0.21    410    280
   38.18  38.33    0.15    440    300
   33.20  33.37    0.17    440    240
   30.57  30.34   -0.23    460    260
   39.68  39.72    0.04    420    260
   38.95  39.02    0.07    430    280

y0:  116.72552
c1:    -0.23451  ±   0.00599
c2:     0.08263  ±   0.00500
```

4. REGRESSION ANALYSIS WITH HIGHER DEGREE PSEUDO VARIABLES

<u>4.1. Use of FIT7 with Pseudo Variables.</u> The procedure FIT7 can give us a linear approximation to a given function, e.g.:

(4.1) $y := y0 + c1\times x1 + c2\times x2 + c3\times x3;$

in which we have three independent variables. If we instead of this wish to find the coefficients in a polynomial of a single variable, but with higher powers:

(4.2) $y := y0 + c1\times x1 + c2\times x1\uparrow 2 + c3\times x1\uparrow 3;$

this may be done by interpretating the higher powers as other independent variables:

(4.3) $x2 := x1\uparrow 2;$
(4.4) $x3 := x1\uparrow 3;$

The program d-359 shown below gives an example of this use of pseudo variables. The program reads Table 1 on page 11 and calculates polynomial approximations of increasing degrees:

(4.5) $y := y0+c1\times x1;$
$y := y0+c1\times x1+c2\times x1\uparrow 2;$
$y := y0+c1\times x1+c2\times x1\uparrow 2+c3\times x1\uparrow 3;$
$y := y0+c1\times x1+c2\times x1\uparrow 2+c3\times x1\uparrow 3+c4\times x1\uparrow 4;$
$y := y0+c1\times x1+c2\times x1\uparrow 2+c3\times x1\uparrow 3+c4\times x1\uparrow 4+c5\times x1\uparrow 5;$
$y := y0+c1\times x1+c2\times x1\uparrow 2+c3\times x1\uparrow 3+c4\times x1\uparrow 4+c5\times x1\uparrow 5+c6\times x1\uparrow 6;$

For each degree, g, the program prints a table of the measured and the calculated y-values, the deviations, the coefficients, and their errors. The mean errors are also printed.

The program is:

Program d-359. Use of FIT7 for one variable and pseudo variables.

```
begin
   integer g, i, j;
   array yxdata[1:13, 1:7], xmean[1:6], coef, coeferror[1:6, 1:1],
   ymean, meanerror[1:1], ycalc[1:13, 1:1];
   copy FIT7<
   copy LEQ1<
   select(17);
   writetext(‹<
Read input to d-359:›);
   lyn;
   select(8);
   for i := 1 step 1 until 13 do
   begin
      yxdata[i, 2] := read real;
      yxdata[i, 1] := read real;
   end for i;
   writetext(‹<
Output d-359
Degree
      x    yinput    ycalc     error            coefficients      errors
›);
   for g := 1 step 1 until 6 do
   begin
      writecr;
      write(‹dddd›, g);
      if g > 1 then
      for i := 1 step 1 until 13 do
      yxdata[i, 1+g] := yxdata[i, 2]↑g;
      FIT7(g, 13, 1, yxdata, xmean, coef, coeferror, ymean,
      meanerror, ycalc, 1₁₀-40);
      for i := 1 step 1 until 13 do
      begin
         writecr;
         write(‹dddddddd›, yxdata[i, 2]);
         write(‹-ddddd.dd›,
         yxdata[i, 1], ycalc[i, 1], ycalc[i, 1]-yxdata[i, 1]);
```

```
        if i ≤ g then
        begin
           write(⟨ddddd⟩, i);
           write(⟨  -d.ddddddd₁₀-dd⟩, coef[i,1], coeferror[i,1]);
        end if i
     end for i;
     writecr;
     writetext(⟨⟨            Mean error:    ⟩);
     write(⟨-ddddd.dddd⟩, meanerror[1]×sqrt(1-g/12));
     writecr;
  end for g;
end;
```

The program gave the following output:

```
Output d-359
Degree
      x   yinput   ycalc    error          coefficients     errors

   1
    300  2413.00  1158.92 -1254.08    1   1.5773461 ₁₀ 1    5.2729666 ₁₀-1
    400  3323.00  2736.27  -586.73
    500  4365.00  4313.62   -51.38
    600  5549.00  5890.96   341.96
    700  6871.00  7468.31   597.31
    800  8321.00  9045.65   724.65
    900  9887.00 10623.00   736.00
   1000 11560.00 12200.35   640.35
   1100 13320.00 13777.69   457.69
   1200 15170.00 15355.04   185.04
   1300 17100.00 16932.38  -167.62
   1400 19090.00 18509.73  -580.27
   1500 21130.00 20087.08 -1042.92
          Mean error:       681.0773
```

2

300	2413.00	2313.24	-99.76	1	6.3290347	3.0307073 $_{10}-1$
400	3323.00	3313.43	-9.57	2	5.2469038 $_{10}-3$	1.6558523 $_{10}-4$
500	4365.00	4418.55	53.55			
600	5549.00	5628.62	79.62			
700	6871.00	6943.62	72.62			
800	8321.00	8363.56	42.56			
900	9887.00	9888.43	1.43			
1000	11560.00	11518.25	-41.75			
1100	13320.00	13253.00	-67.00			
1200	15170.00	15092.69	-77.31			
1300	17100.00	17037.32	-62.68			
1400	19090.00	19086.89	-3.11			
1500	21130.00	21241.40	111.40			
	Mean error:		67.6337			

3

300	2413.00	2420.74	7.74	1	2.7781902	8.4923511 $_{10}-2$
400	3323.00	3313.43	-9.57	2	9.6447373 $_{10}-3$	1.0227702 $_{10}-4$
500	4365.00	4359.92	-5.08	3	-1.6288268 $_{10}-6$	3.7615937 $_{10}-8$
600	5549.00	5550.43	1.43			
700	6871.00	6875.21	4.21			
800	8321.00	8324.46	3.46			
900	9887.00	9888.43	1.43			
1000	11560.00	11557.34	-2.66			
1100	13320.00	13321.41	1.41			
1200	15170.00	15170.88	0.88			
1300	17100.00	17095.96	-4.04			
1400	19090.00	19086.89	-3.11			
1500	21130.00	21133.89	3.89			
	Mean error:		4.6746			

4

300	2413.00	2416.43	3.43	1	3.3581258	2.1710716 $_{10}$-1
400	3323.00	3316.31	-6.69	2	8.4987485 $_{10}$-3	4.1714220 $_{10}$-4
500	4365.00	4364.10	-0.90	3	-7.1349355 $_{10}$-7	3.2867213 $_{10}$-7
600	5549.00	5552.79	3.79	4	-2.5425809 $_{10}$-10	9.0957000 $_{10}$-11
700	6871.00	6874.73	3.73			
800	8321.00	8321.67	0.67			
900	9887.00	9884.77	-2.23			
1000	11560.00	11554.55	-5.45			
1100	13320.00	13320.93	0.93			
1200	15170.00	15173.23	3.23			
1300	17100.00	17100.14	0.14			
1400	19090.00	19089.77	-0.23			
1500	21130.00	21129.58	-0.42			
	Mean error:		3.3203			

5

300	2413.00	2414.57	1.57	1	4.4495858	6.2678886 $_{10}$-1
400	3323.00	3319.08	-3.92	2	5.5209240 $_{10}$-3	1.6698240 $_{10}$-3
500	4365.00	4365.62	0.62	3	3.0348248 $_{10}$-6	2.0709615 $_{10}$-6
600	5549.00	5551.87	2.87	4	-2.4583464 $_{10}$-9	1.2085363 $_{10}$-9
700	6871.00	6872.54	1.54	5	4.8964085 $_{10}$-13	2.6788839 $_{10}$-13
800	8321.00	8319.99	-1.01			
900	9887.00	9884.76	-2.24			
1000	11560.00	11556.22	-3.78			
1100	13320.00	13323.11	3.11			
1200	15170.00	15174.16	4.16			
1300	17100.00	17098.64	-1.36			
1400	19090.00	19087.00	-3.00			
1500	21130.00	21131.41	1.41			
	Mean error:		2.7222			

6

300	2413.00	2408.96	-4.04	1	$1.6310036_{10}1$	5.0760758
400	3323.00	3330.85	7.85	2	$-3.5062186_{10}-2$	$1.7265767_{10}-2$
500	4365.00	4365.78	0.78	3	$7.2276290_{10}-5$	$2.9340247_{10}-5$
600	5549.00	5544.05	-4.95	4	$-6.5117736_{10}-8$	$2.6491558_{10}-8$
700	6871.00	6867.14	-3.86	5	$2.9228845_{10}-11$	$1.2141522_{10}-11$
800	8321.00	8321.96	0.96	6	$-5.2549615_{10}-15$	$2.2211710_{10}-15$
900	9887.00	9891.31	4.31			
1000	11560.00	11560.63	0.63			
1100	13320.00	13320.86	0.86			
1200	15170.00	15167.62	-2.38			
1300	17100.00	17096.50	-3.50			
1400	19090.00	19094.67	4.67			
1500	21130.00	21128.66	-1.34			
	Mean error:		3.8747			

This calculation shows that the mean error is gradually decreased when the polynomial degree is increased until a certain limit. The error for a sixth order polynomial is slightly higher than for a fifth order polynomial. At the same time the coefficient errors have increased to be of the same order of magnitude as the coefficients, indicating that we have found the limit beyond which the polynomial approximation method cannot give better results, at least through the regression analysis with pseudo variables. Slightly better approximations can be found by the use of orthogonal polynomials (see Part 2).

The fourth order polynomial approximation of the enthalpy table is shown in Figure 2 on page 35.

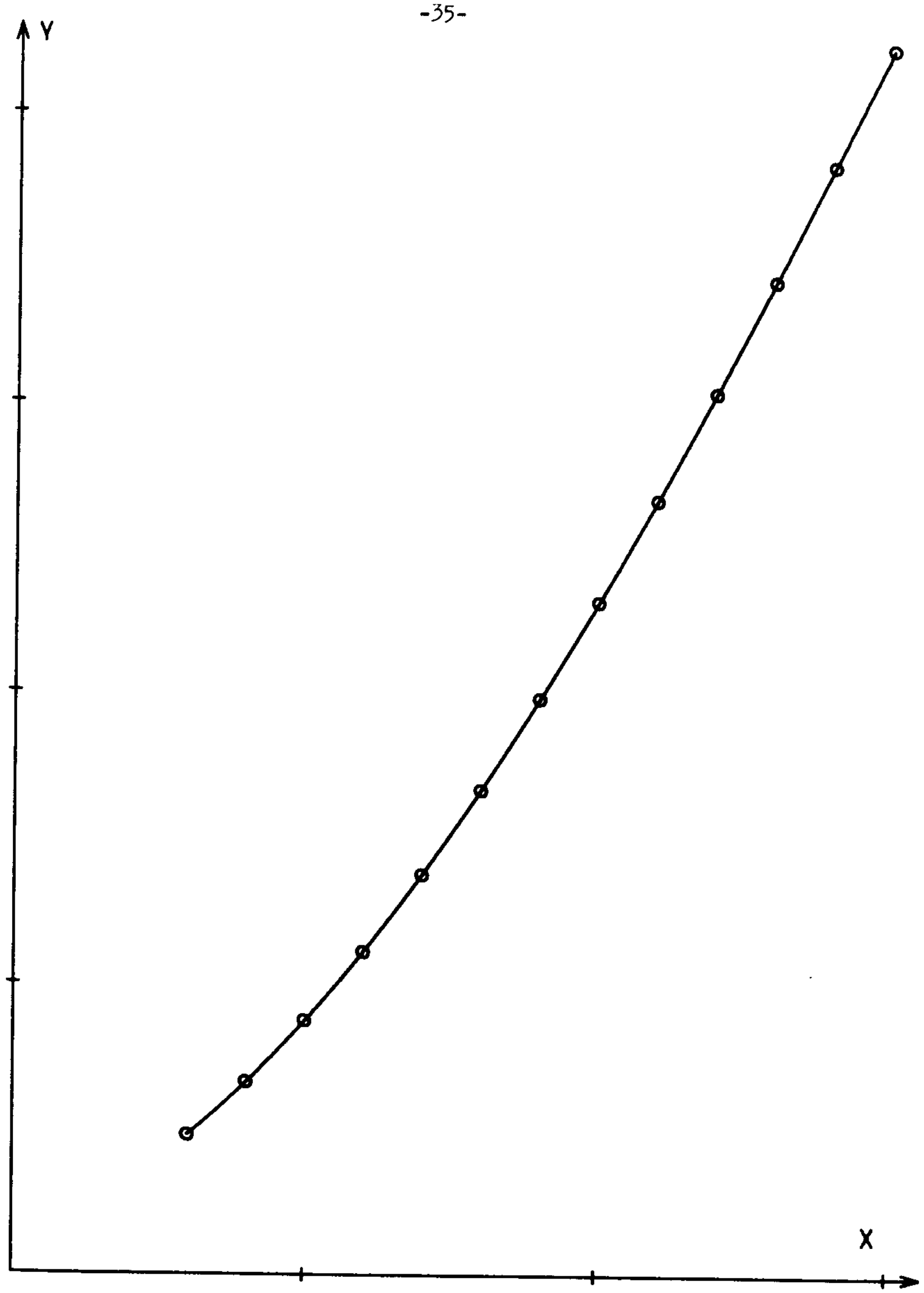

Figure 2

Fourth Degree Approximation

5. INVESTIGATION OF VARIOUS FEATURES

5.1. Centering Around First Observation. A minor improvement in the method used in FIT7 has been suggested by Chr. Gram and P. Mondrup, Regnecentralen, A/S. Instead of scanning the observation material twice to obtain first the mean values of x and y and then to generate the SXX-matrix, a single scanning is made in which the SXX summation is made relative to the first observation and not relative to the mean value. This means that the two equations (3.9) and (3.10) are replaced by:

```
(5.1)     (xobs[i,j]-xobs[1,j])×(xobs[i,k]-xobs[1,k])
(5.2)     (xobs[i,j]-xobs[1,j])×(yobs[i]-yobs[1])
```

and the scanning is made from i = 2 to i = OBS.

Another feature is that the diagonal elements in SXX can be calculated in a more direct way as:

```
(5.3)     SXX[j,j] := SUM(xobs[i,j]↑2) -(SUM(xobs[i,j]))↑2/OBS;
```

where the summation is made from i = 1 to i = OBS.

When the SXX-matrix has been generated, it must be corrected for the effect of using the first observation as basis instead of the mean values. These minor details are explained in the following sections.

The basic idea behind the use of the first observation as basis is that this basis has no inherent rounding error whereas the mean values will contain small rounding errors. Calculations showing this effect are reported in the next section.

5.2. The Procedure FIT8. This is the improved version of FIT7 with the feature of centering around the first observation. The declaration is:

```
integer procedure FIT8(VAR, OBS, FUNC, yxdata, xmean, coef,
coeferror, ymean, meanerror, ycalc, eps);
value VAR, OBS, FUNC, eps;
integer VAR, OBS, FUNC;
real eps;
array yxdata, xmean, coef, coeferror, ymean, meanerror,
ycalc;
```

```
begin
   integer i, j, k;
   real xj;
   array SXX[1:VAR, 1:(2×VAR+FUNC)], yone, yi[1:FUNC];
   for i := 1 step 1 until FUNC do
   begin
      yone[i] := yxdata[1, i];
      ymean[i] := meanerror[i] := 0;
   end for i;
   for j := 1 step 1 until VAR do
   begin
      coeferror[j, 1] := xmean[j] := 0;
      ycalc[j, 1] := yxdata[1, FUNC+j];
      for k := j + 1 step 1 until 2×VAR+FUNC do
      SXX[j, k] := 0;
      SXX[j, j+VAR] := 1;
   end for j;
   for i := 2 step 1 until OBS do
   begin
      for k := 1 step 1 until FUNC do
      begin
         yi[k] := yxdata[i, k] - yone[k];
         ymean[k] := ymean[k] + yi[k];
      end for k;
      for j := 1 step 1 until VAR do
      begin
         xj := yxdata[i, FUNC+j] - ycalc[j, 1];
         xmean[j] := xmean[j] + xj;
         coeferror[j, 1] := coeferror[j, 1] + xj↑2;
         for k := 1 step 1 until FUNC do
         SXX[j, 2×VAR+k] := SXX[j, 2×VAR+k] + yi[k]×xj;
         for k := j + 1 step 1 until VAR do
         SXX[j, k] := SXX[j, k] + (yxdata[i, FUNC+k] -
         ycalc[k, 1])×xj;
      end for j;
   end for i;
   for k := 1 step 1 until FUNC do
   ymean[k] := ymean[k]/OBS;
```

```
    for j := 1 step 1 until VAR do
    begin
       xj := xmean[j];
       SXX[j, j] := coeferror[j, 1] - xj↑2/OBS;
       for k := 1 step 1 until FUNC do
       SXX[j, 2×VAR+k] := SXX[j, 2×VAR+k] - xj×ymean[k];
       xj := xj/OBS;
       for k := j + 1 step 1 until VAR do
       SXX[k, j] := SXX[j, k] := SXX[j, k] - xmean[k]×xj;
       xmean[j] := ycalc[j, 1] + xj;
    end for j;
    FIT8 := i := LEQ1(VAR, VAR+FUNC, SXX, eps);
    if i = 0 then
    begin
       for k := 1 step 1 until FUNC do
       ymean[k] := yone[k] + ymean[k];
       for j := 1 step 1 until VAR do
       for i := 1 step 1 until FUNC do
       coef[j, i] := SXX[j, 2×VAR+i];
       for i := 1 step 1 until OBS do
       begin
          for k := 1 step 1 until FUNC do
          ycalc[i, k] := ymean[k];
          for j := 1 step 1 until VAR do
          begin
             xj := yxdata[i, FUNC+j] - xmean[j];
             for k := 1 step 1 until FUNC do
             ycalc[i, k] := ycalc[i, k] + xj×coef[j, k];
          end for j;
          for k := 1 step 1 until FUNC do
          meanerror[k] := meanerror[k] +
          (ycalc[i, k] - yxdata[i, k])↑2;
       end for i;
       for k := 1 step 1 until FUNC do
       meanerror[k] := sqrt(meanerror[k]/(OBS-VAR-1));
       for j := 1 step 1 until VAR do
       for k := 1 step 1 until FUNC do
       coeferror[j, k] := meanerror[k]×sqrt(abs(SXX[j, VAR+j]));
    end if i = 0;
end FIT8;
```

The following discussion of FIT8 mainly emphasizes the differences between FIT7 and FIT8. The two arrays:

yone, yi[1:FUNC]

are used for storage of the y-values in observation no. 1 and the difference yobs[i]-yobs[1] as it is used during the scanning.

The values of yone[1:FUNC] are inserted at the beginning of the procedure. We also need two new arrays for storage of the x-values in observation no. 1 and the sum of the squares of the x-values (measured relative to the first observation). These arrays have the dimension 1:VAR, but in order to save space we can use ycalc[1:VAR,1] for xobs[1,1:VAR] and coeferror[1:VAR,1] for the sum of the squares. These arrays are not used for their original purpose until after the solution of the equations. The values of ycalc and coeferror are set in the first for-statement counting in j.

Then comes the first scanning in which i goes from 2 to OBS. It has first a for-statement counting in k which sums up the mean values of y relative to observation no. 1 in ymean[1:FUNC]. Then comes a for-statement counting in j from 1 to VAR and which performs five operations:

1. xj is calculated as xobs[i,j] - xobs[1,j].
2. xj is added to xmean[j].
3. The square of xj is added to coeferror[j,1].
4. A for-statement in k going from 1 to FUNC updates row no. j in the right-hand sides of the equations by addition of yi[k]×xj to column no. k (this is the term in equation (5.2) for k = 1).
5. A second for-statement in k updates row no. k in the main part of SXX by adding the term in equation (5.1).

After the main for-statement in i, a small for-statement in k divides the value of ymean by OBS.

Then comes a for-statement counting j from 1 to VAR which corrects the SXX-matrix for the number of observations and for the fact that the centering was made around the first observation instead of around the mean values. This is made in six operations:

1. The previous value of xmean[j] is assigned to xj.

2. The diagonal element is calculated after equation (4.3).

3. A for-statement in k corrects the right-hand sides by subtraction of SUM(xobs[i,j])×SUM(yobs[i,k])/OBS.

4. xj is divided by OBS.

5. The upper half of the main part of the SXX-matrix is corrected by subtraction of SUM(xobs[i,j])×SUM(xobs[i,k])/OBS. The lower half of SXX is assigned symmetrically.

6. xmean is corrected by adding the first observation.

The remaining part of FIT8 is the same as in FIT7, except that the values of ymean are corrected by adding the first observation. In this part of the procedure the arrays ycalc and coeferror are used for their original purpose.

A comparison between FIT7 and FIT8 is given by the program d-360 shown below. We want to approximate the enthalpy table on page 11 by polynomials of increasing degree in the same way as did program d-359 on page 30. A direct comparison of FIT7 and FIT8 on this material shows only a small effect, but if we distort the x-values like this:

Obser- vation no.	Original x-value	Distorted x-value
1	300	333.33333 + delta
2	400	444.44444 + delta
3	500	555.55555 + delta
4	600	666.66666 + delta
etc.		

an effect can be observed. The distortion is equivalent to a change in the temperature scale and is brought about by calculating the x-value of observation no. i from:

(5.4) x[i] := 1000×(i+2)/9 + delta;

The program is run for different values of delta. A print-out of the program is:

Program d-360. Comparison of FIT7 and FIT8.

```
begin
   integer i, j, type;
   real delta, x;
   array yxdata[1:13, 1:7], xmean[1:6], coef, coeferror[1:6, 1:1],
   ymean, meanerror[1:1], ycalc[1:13, 1:1];
   copy FIT7<
   copy FIT8<
   copy LEQ1<
   select(17);
   writetext({<
Read input to d-360:});
   lyn;
   select(8);
   for i := 1 step 1 until 13 do
   begin
      yxdata[i, 2] := read real;
      yxdata[i, 1] := read real;
   end for i;
   writetext({<
Output d-360
delta       Mean error
        FIT7        FIT8
});
   for delta := 0, 1, 10×delta while delta < 1₁₀6 do
   begin
      writecr;
      write({ d₁₀d}, delta);
      for i := 1 step 1 until 13 do
      begin
         x := 1000×(i+2)/9 + delta;
         for j := 1 step 1 until 6 do
         yxdata[i, 1+j] := x↑j;
      end for i;
      for type := 1, 2 do
      begin
         i := case type of
         (FIT7(6, 13, 1, yxdata, xmean, coef, coeferror, ymean,
         meanerror, ycalc, 1⏨-100),
         FIT8(6, 13, 1, yxdata, xmean, coef, coeferror, ymean,
         meanerror, ycalc, 1⏨-100));
         if i ≠ 0 then writetext(⟨< ERROR   ⟩)
         else
         write(⟨ddd.d00000⟩, meanerror[1]×sqrt(0.5));
      end for type;
   end for delta;
   writecr;
end;
```

The program gave the output:

```
Output d-360
delta       Mean error
         FIT7        FIT8

  0    11.44        1.717
  1    12.08        2.161
 1⏨1  15.43        2.134
 1⏨2   2.305       2.485
 1⏨3   2.809       2.905
 1⏨4   3.414       5.804
 1⏨5  70.11       71.43
```

For small values of delta FIT7 gives larger mean errors than FIT8. The reason for this is probably that the infinite decimal fractions in the x-values give loss of accuracy in FIT7 when the mean values of x are calculated. This error is not found in FIT8. For large values of delta the mean error increases again for both procedures. This is very natural because the high value of delta will mask the small differences in the x-values.

Conclusion: FIT8 is better than FIT7 and should be used.

5.3. Linear Equations Versus Matrix Inversion. As mentioned on page 23 the solution of the normal equations in the regression analysis can be made either by direct solution on the augmented matrix:

$$\begin{matrix} SXX & SXX & SXX & SXX \\ SXX & SXX & SXX & SXX \\ SXX & SXX & SXX & SXX \end{matrix}$$

or by inverting the square matrix:

$$\begin{matrix} SXX & SXX & SXX \\ SXX & SXX & SXX \\ SXX & SXX & SXX \end{matrix}$$

giving the inverse:

$$\begin{matrix} INV & INV & INV \\ INV & INV & INV \\ INV & INV & INV \end{matrix}$$

which is then multiplied by the right-hand side:

$$\begin{matrix} SXX \\ SXX \\ SXX \end{matrix}$$

giving the required coefficients. We have performed a comparison of the two methods by taking the calculations carried out by FIT7 in the program d-359 (page 30) and running them on a special program (d-363) which contains a modified version of FIT7. This version generates the SXX-matrix plus the unit matrix:

$$\begin{matrix} SXX & SXX & SXX & 1 & 0 & 0 & SXX \\ SXX & SXX & SXX & 0 & 1 & 0 & SXX \\ SXX & SXX & SXX & 0 & 0 & 1 & SXX \end{matrix}$$

Before the equations are solved, the right-hand side is stored in a local array, SXY. We then solve the equations by means of LEQ1 and get the inverse and the coefficients:

```
--- --- --- INV INV INV coef
--- --- --- INV INV INV coef
--- --- --- INV INV INV coef
```

The calculation of ycalc and the mean error is now carried out in two cycles, the first using the coefficients obtained by multiplication of the inverse, INV, by the right-hand side, SXY. The second cycle uses the coefficients obtained from the direct solution. The result of the calculation was:

Degree	Calculated Mean Error	
	Direct Solution	Matrix Inversion
1	681.08	681.08
2	67.63	67.63
3	4.67	4.67
4	3.32	4.47
5	2.72	69.29
6	3.87	10564.25

This clearly shows how dangerous it can be to use matrix inversion instead of direct solution, although the two methods are equivalent from a mathematical point of view.

5.4. Regression Through First Observation.

It is sometimes required that the approximation expression generated in a regression analysis has certain special properties, such as passing exactly through a specified point. Chapter 12 contains a general discussion of this subject.

If we wish that the linear regression must pass exactly through a given point, this can be obtained in a very simple way when the method of FIT8 is used. The specified point is placed as observation no. 1, and the first part of FIT8 immediately gives the required approximation. The second part of the calculation should then be skipped.

5.5. Scaling Problems in FORTRAN. When FIT8 is translated into FORTRAN and tested on an IBM 360/44 computer, troubles occur because this computer has a smaller range of floating point numbers than has the GIER computer. The high powers in the pseudo variables give overflow in the calculations. This can be avoided by scaling the original x-values before the regression analysis with proper back transformation after the calculation. The scale factors should preferably be powers of 16 in order to avoid the introduction of rounding errors.

PART 2. ORTHOGONAL POLYNOMIAL APPROXIMATION

6. BASIC ON ORTHOGONAL POLYNOMIALS

In Part 1 of this book some examples were given on how to approximate a function:

(6.1) $y = f(x)$

by polynomials of various degrees:

(6.2)
$$y = y0+c1\times x;$$
$$y = y0+c1\times x+c2\times x\uparrow 2;$$
$$y = y0+c1\times x+c2\times x\uparrow 2+c3\times x\uparrow 3;$$
etc.

As illustration to these methods we used an example of the enthalpy of methane as a function of the temperature. In order to illustrate the use of orthogonal polynomials it is more convenient to use a somewhat smaller table and of a function where it is easy to include more independent variables. Table 2 below gives the ammonia mole per cent, Y, in equilibrium as a function of the pressure, X atm. abs. The table is valid for a 3:1 hydrogen-nitrogen mixture at 400 deg. C.

X	Y
200	38.8210
220	40.9274
240	42.9013
260	44.7590
280	46.5139

Table 2

The table can be extended by including the temperature and the inert contents as further independent variables.

When we approximate this function by means of polynomials of increasing degrees (either by regression analysis or by the methods described later) we can get the following results:

(6.3) $y = 19.72+0.09609\times x;$

$y = 11.41+0.16630\times x-1.463_{10}-4\times x\uparrow 2;$

$y = 7.24+0.21934\times x-3.690_{10}-4\times x\uparrow 2+3.094_{10}-7\times x\uparrow 3;$

$y = 4.81+0.26051\times x-6.290_{10}-4\times x\uparrow 2+1.035_{10}-6\times x\uparrow 3-7.561_{10}-10\times x\uparrow 4;$

The coefficients are not written with full accuracy here, but the numbers show one important feature: When the polynomial degree is increased, all the lower degree coefficients change their numerical value. This must be so, of course, but we may put the question if it were possible to perform the generation of the polynomials in such a way that the addition of the next higher term had no influence on the values of the lower degree coefficients calculated already. This is in fact possible by the use of orthogonal polynomials. Instead of the conventional polynomial:

(6.4) $y = c0+c1\times x+c2\times x\uparrow 2+c3\times x\uparrow 3;$

we write the approximation as:

(6.5) $y = a0\times P0(x)+a1\times P1(x)+a2\times P2(x)+a3\times P3(x);$

In this representation the addition of a further term, $a4\times P4(x)$, has no influence upon the values of the numerical constants, a0, a1, a2, and a3. The price for this advantage is that the x-powers:

(6.6) $x,\ x\uparrow 2,\ x\uparrow 3$

are now replaced by polynomials in x:

(6.7) $P0(x),\ P1(x),\ P2(x),\ P3(x)$

of the degrees 0, 1, 2, and 3. P0(x) corresponds to c0. If we want the coefficients in the conventional polynomial, these can easily be calculated from the polynomials, P0, P1, etc.

6.1. Orthogonality. The polynomials P0, P1, P2, etc. introduced above are the so-called orthogonal polynomials. We shall now see what this name means and how the polynomials can be used in the approximation of functions.

In Part 1 of this book the method of linear regression analysis for establishment of the normal equations was described on page 13. The explanation given there refers to a function of VAR independent variables, and is somewhat obscured by the feature of subtracting the mean values from the x-values when the matrix is generated. The element, SXX[j,k], of the square part of the SXX-matrix is found by summation of:

(6.8) (xobs[i,j]-xmean[j])×(xobs[i,k]-xmean[k])

for all observations for i = 1 to i = OBS.

We now consider the case in which a function of a single variable, y = f(x), available as a table of OBS observations:

x, y[1:OBS]

is to be approximated by a conventional polynomial of degree DEG:

(6.9) cpol(x) = c[0]+c[1]×x+c[2]×x↑2+....+c[DEG]×x↑DEG;

The deviation in point no. i can be written as:

(6.10) D := cpol(x[i]) - y[i];

and the square of the deviation:

(6.11) SD := (cpol(x[i]) - y[i])↑2;

or:

(6.12) SD := (cpol(x[i]))↑2 + (y[i])↑2 - 2×y[i]×cpol(x[i]);

Summation of this with i going from 1 to OBS gives the sum of the squares of the deviations:

```
(6.13)     SSD := SUM((cpol(x[i]))↑2) + SUM((y[i])↑2)
                -2×SUM(cpol(x[i])×y[i]);
```

The DEG+1 unknown coefficients, c[0:DEG], are calculated by setting up the DEG+1 normal equations. Equation no. p expresses that the derivative of SSD with respect to c[p] is zero, corresponding to a minimum of SSD. The normal equations are written as the rows in a matrix, SXX, of the dimension:

```
SSX[0:DEG,0:DEG+1]
```

Differentiation of equation (6.13) and some rearrangement gives the result that the element, SXX[j,k], of the square part of SXX can be written as:

```
(6.14)     SUM(x[i]↑j×x[i]↑k);
```

the summation being carried out from i = 1 to i = OBS. The right-hand side of the equation system is the last column in SXX and the element SXX[j,DEG+1] in this column is found by a similar summation:

```
(6.15)     SUM(y[i]×x[i]↑j);
```

We shall now repeat this calculation using the orthogonal polynomials. We first write equation (6.5) in a slightly different way:

```
(6.16)     y := a[0]×P(0,x)+a[1]×P(1,x)+a[2]×P(2,x)+...+a[DEG]×P(DEG,x);
```

The unknown coefficients are now: a[0:DEG]. The generation of the orthogonal polynomials:

```
(6.17)     P(0,x), P(1,x), etc.
```

is explained in the following section. The numbers 0, 1, etc. written as the first argument in P indicate the degree of the polynomial.

The deviation in point no. i becomes:

```
(6.18)    D :=a[0]×P(0,x[i])+a[1]×P(1,x[i])+...+a[DEG]×P(DEG,x[OBS])
               -y[i];
```

Further calculation gives SD and SSD as above. Differentiation of SSD gives the normal equations and the element, SXX[j,k], of the square part of SXX becomes:

```
(6.19)    SUM(P(j,x[i])×P(k,x[i]));
```

The right-hand side element no. j becomes:

```
(6.20)    SUM(y[i]×P(j,x[i]));
```

We now consider the summation in equation (6.19). The whole point in the use of the orthogonal polynomials lies in the fact that these polynomials are generated in such a way that the summation becomes zero for j ≠ k:

```
(6.21)    SUM(P(j,x[i])×P(k,x[i])) = 0 for j ≠ k.
```

The mathematical term: Orthogonal refers to this zero condition or, more correctly, to the similar condition when the summation is replaced by integration over a given interval.

For j = k the summation gives:

```
(6.22)    SUM(P(j,x[i]))↑2);
```

which is different from zero. This means that the elements on the diagonal in SXX are different from zero and those outside the diagonal are zero:

```
(6.23)    SXX  0    0    0    0    SXX
          0    SXX  0    0    0    SXX
          0    0    SXX  0    0    SXX
          0    0    0    SXX  0    SXX
          0    0    0    0    SXX  SXX
```

In other words, the normal equations can be solved immediately by division of the right-hand side elements by the corresponding diagonal element:

```
(6.24)    a[0] := SUM(y[i]×P(0,x[i]))/SUM((P(0,x[i]))↑2);
          a[1] := SUM(y[i]×P(1,x[i]))/SUM((P(1,x[i]))↑2);
          a[2] := SUM(y[i]×P(2,x[i]))/SUM((P(2,x[i]))↑2);
          .....
          .....
          a[DEG]:=SUM(y[i]×P(DEG,x[i]))/SUM((P(DEG,x[i]))↑2);
```

It is easy to understand that the use of orthogonal polynomials will save much computer time because it is not necessary to generate all the matrix terms outside the diagonal and because there is no solution of a set of simultaneous equations. Some additional calculation is required, of course, for the generation and handling of the orthogonal polynomials.

6.2. Generation of Orthogonal Polynomials. We shall now explain how the orthogonal polynomials can be generated by a set of recurrence formulas. The following is taken from Lapidus (1962).

We start by defining the set of polynomials as:

```
(6.25)    P(-1,x) = 0
          P( 0,x) = 1
          P( 1,x) = (x - alfa[1])×P(0,x) - beta[0]×P(-1,x)
          P( 2,x) = (x - alfa[2])×P(1,x) - beta[1]×P( 0,x)
          .....
          .....
          P(j+1,x)= (x - alfa[j+1])×P(j,x) - beta[j]×P(j-1,x)
```

with beta[0] = 0. We must now determine the series alfa and beta in such a way that the orthogonality criterion (6.21) is fulfilled. We can start by assuming that:

```
(6.26)    P(0,x), P(1,x), ....., P(j,x)
```

are all orthogonal to each other. We then multiply the formula for the polynomial P(j+1,x) in equation (6.25) by P(k,x) and perform the summation from i = 1 to i = OBS. This gives:

```
(6.27)    SUM(P(k,x[i])×P(j+1,x[i])) =
          SUM(P(k,x[i])×(x[i]-alfa[j+1])×P(j,x[i])) -
          SUM(P(k,x[i])×beta[j]×P(j-1,x[i]))
```

This is valid for $0 \leq k \leq j$. Insertion of k = j gives:

```
(6.28)    SUM(P(j,x[i])×P(j+1,x[i])) =
          SUM(x[i]×P(j,x[i])↑2) - alfa[j+1]×SUM(P(j,x[i])↑2) -
          beta[j]×SUM(P(j,x[i])×P(j-1,x[i]))
```

The last term on the right-hand side is zero from the orthogonality of P(j,x) and P(j-1,x). In order to secure the orthogonality of P(j,x) and P(j+1,x), we must calculate alfa[j+1] as:

```
(6.29)    alfa[j+1] = SUM(x[i]×P(j,x[i])↑2)/SUM(P(j,x[i])↑2)
```

Insertion of k = j-1 into equation (6.27) gives:

```
(6.30)    SUM(P(j-1,x[i])×P(j+1,x[i])) =
          SUM(x[i]×P(j-1,x[i])×P(j,x[i])) -
          alfa[j+1]×SUM(P(j-1,x[i])×P(j,x[i])) -
          beta[j]×SUM(P(j-1,x[i])↑2)
```

Here, the second term on the right-hand side is zero because of the orthogonality of P(j-1,x) and P(j,x). The orthogonality of P(j+1,x) and P(j-1,x) is then secured by calculating beta[j] as:

```
(6.31)    beta[j] = SUM(x[i]×P(j-1,x[i])×P(j,x[i]))/SUM(P(j-1,x[i])↑2)
```

As the orthogonality of P(-1,x), P(0,x), and P(1,x) is easily verified by direct calculation, we have the material for an induction proof of the orthogonality of all polynomials generated in this way using the equations (6.29) and (6.31) for alfa and beta.

The program d-372 generates the orthogonal polynomials for the 5 x-values in Table 2 on page 49. The program is:

Program d-372. Generation of Orthogonal Polynomials.

```
begin
   integer i, t;
   real alfa, beta, D, OLDSQSUM, XPROD;
   array X[1:5], ORPOL[-2:4, 1:5], SSQPOL[0:4];
   for i := 1 step 1 until 5 do X[i] := 180 + 20×i;
   for i := 1 step 1 until 5 do
   begin
      ORPOL[-2,i] := 1;
      ORPOL[-1,i] := 0;
   end for i;
   alfa := 0;
   beta := OLDSQSUM := 1;
   for t := 0 step 1 until 4 do
   begin
      SSQPOL[t] := XPROD := 0;
      for i := 1 step 1 until 5 do
      begin
         D := ORPOL[t, i] :=
         ORPOL[t-2,i]×beta+ ORPOL[t-1,i]×(X[i]+alfa);
         D := D↑2;
         SSQPOL[t] := SSQPOL[t] + D;
         XPROD := XPROD + D×X[i];
      end for i;
      beta := -SSQPOL[t]/OLDSQSUM;
      OLDSQSUM := SSQPOL[t];
      alfa := -XPROD/OLDSQSUM;
   end for t;
   select(8);
   writetext(⌈<
Output d-372:
>⌉);
   for i := 1 step 1 until 5 do
   begin
      writecr;
      for t := 0 step 1 until 4 do
      write(⌈  -d.dddd10-dd⌉, ORPOL[t,i]);
   end for i;
```

```
    writecr;
    writecr;
    for t := 0 step 1 until 4 do
    write(⟨ -d.dddd₁₀-dd⟩, SSQPOL[t]);
    writecr;
end;
```

Output from the program looks as follows:

Output d-372:

```
1.0000     -4.0000 ₁₀ 1   8.0000 ₁₀ 2  -9.6000 ₁₀ 3   5.4857 ₁₀ 4
1.0000     -2.0000 ₁₀ 1  -4.0000 ₁₀ 2   1.9200 ₁₀ 4  -2.1943 ₁₀ 5
1.0000      0.0000       -8.0000 ₁₀ 2   0.0000        3.2914 ₁₀ 5
1.0000      2.0000 ₁₀ 1  -4.0000 ₁₀ 2  -1.9200 ₁₀ 4  -2.1943 ₁₀ 5
1.0000      4.0000 ₁₀ 1   8.0000 ₁₀ 2   9.6000 ₁₀ 3   5.4857 ₁₀ 4

5.0000      4.0000 ₁₀ 3   2.2400 ₁₀ 6   9.2160 ₁₀ 8   2.1065₁₀ 11
```

The basic statement in the program is that which calculates P(t,x) from the values of P(t-1,x) and P(t-2,x). It has the form:

(6.32) ORPOL[t,i] := ORPOL[t-2,i]×beta+ORPOL[t-1,i]×(X[i]+alfa);

This statement corresponds to the last formula in equation (6.25) when j+1 is replaced by t:

$$P(t,x) = (x - alfa[t])\times P(t-1,x) - beta[t-1]\times P(t-2,x) \tag{6.33}$$

The values of the orthogonal polynomials are written as the array:

ORPOL[-2:DEG,1:OBS]

in which DEG is the highest degree to which we want to calculate the orthogonal polynomials and OBS is the number of observations. The values of x must be available as the array:

X[1:OBS]

The correspondence between P and ORPOL is then:

(6.34) $P(t,X[i]) = ORPOL[t,i]$

Comparison of equations (6.32 and 6.33) shows that alfa and beta are inserted with the opposite sign in equation (6.32). There is no special reason for this change of sign, except for a historical one which goes back to the time when these formulas were evaluated in machine language.

The program d-372 starts by assigning start values to the first items of the orthogonal polynomials:

(6.35)
```
ORPOL[-2,i] := 1;
ORPOL[-1,i] := 0;
```

This is done for i going from 1 to OBS (here 5). In equation (6.25) the initial setting was slightly different:

(6.36) $P(-1,x) = 0$
$P(\ 0,x) = 1$

But the result is the same. In d-372 the start values of alfa and beta are set to 0 and 1, respectively. When the main for-statement in d-372:

(6.37)
```
for t := 0 step 1 until 4 do
```

is carried out for $t = 0$, the result of equation (6.32) becomes:

(6.38) ORPOL[0,i] := ORPOL[-2,i]×beta + ORPOL[-1,i]×(X[i]+alfa)

$= 1\times1 + 0\times(X[i]+0) = 1$

which is the required value.

The local array:

```
SSQPOL[0:DEG]
```

is used for storage of the sum of the squares of the orthogonal polynomials for the degrees 0 to DEG. This corresponds to the expression SUM(P(j,x[i])↑2) found in equation (6.29). For each value of t in the for-statement (6.37) we must also have access to the sum of the squares for one degree lower: SUM(P(t-1,x[i])↑2). This is stored as the simple variable OLDSQSUM. We need this in equation (6.31).

In equation (6.29) we also need the sum: SUM(x[i]×P(j,x[i])↑2). This is calculated in the simple variable: XPROD.

The calculation in d-372 is now clear. After the initial setting of P(-2,x) and P(-1,x), the start value of alfa is set to zero and of beta and OLDSQSUM to 1. Then comes the for-statement in which t is counted from 0 to DEG. For each value of t we set the sum of the squares of the orthogonal polynomials for the degree t to zero (SSQPOL[t]). The cell XPROD used for summation of x[i]×P(t,x[i])↑2 is also set to zero. We then start a for-statement in which i is counted from 1 to OBS (here 5). For each value of i ORPOL[t,i] is calculated from equation (6.32) and it is also assigned to the local simple variable, D. We then square D and add it to SSQPOL[t]. The product of D and X[i] is added to XPROD. After the for-statement in i we calculate beta from:

(6.39)   beta := -SSQPOL[t]/OLDSQSUM;

The presence of the minus sign in this formula was explained above. Apart from this, the formula is equivalent to:

(6.40)   beta := SUM(P(t,x[i])↑2)/SUM(P(t-1,x[i])↑2)

This formula is equivalent to equation (6.31), provided we can prove the identity:

(6.41)   SUM(x[i]×P(j-1,x[i])×P(j,x[i])) = SUM(P(j,x[i])↑2)

As shown by Forsythe (1957) this is easily done by multiplying the last line in equation (6.25) by P(j+1,x):

(6.42)   P(j+1,x)↑2 = (x-alfa[j+1])×P(j,x)×P(j+1,x)-

beta[j]×P(j-1,x)×P(j+1,x)

When the summation of the OBS values is carried out, the terms with alfa and beta as factors disappear because of the orthogonality and the remaining terms are identical to those in equation (6.41).

OLDSQSUM is then put equal to SSQPOL[t] for use in the next cycle for t = t+1. Finally, alfa is found from:

(6.43) alfa := -XPROD/OLDSQSUM;

which is identical to equation (6.29), except for the sign change which was explained above.

The output from program d-372 contains the values of the orthogonal polynomials arranged as one row for each X-value and one column for each degree. The extra row below the ORPOL-values contains SSQPOL for the same degree as the ORPOL-columns.

6.3. Polynomial Coefficients. We have now seen how the orthogonal polynomials can be generated. Evaluation of the regression coefficients a[0:DEG] is then straightforward after equation (6.24) and is further explained in Chapter 7.

The regression coefficients, a[0:DEG], correspond to the expansion in equation (6.16) where a[j] is multiplied by P(j,x). Although this expansion can be used as such, it is more practical to use the conventional expansion in equation (6.9) in which c[j] is multiplied by x↑j. The question is then how to calculate the array c[0:DEG] when the array a[0:DEG] is known.

This is very easy to do because the orthogonal polynomials are also conventional polynomials, i.e. the orthogonal polynomial of degree t can be defined by the t+1 coefficients to the x-powers:

(6.44) P(t,x) = POLC[t,0]+x×POLC[t,1]+x↑2×POLC[t,2]+....
....+x↑t×POLC[t,t]

The required polynomial coefficients are calculated and stored as the array:

(6.45) POLC[0:DEG,0:DEG]

in which POLC[i,j] is the coefficient to x in the power j in the orthogonal polynomial of degree i. When the array POLC is known, the conventional coefficients are calculated by summing up of terms of the same order:

```
(6.46)      c[0] = a[0]×POLC[0,0]+a[1]×POLC[1,0]+...+a[DEG]×POLC[DEG,0]
            c[1] = a[1]×POLC[1,1]+...+a[DEG]×POLC[DEG,1]
            .....
            c[DEG] = a[DEG]×POLC[DEG,DEG]
```

In order to calculate the array POLC we start from the fundamental recursion equation (6.25). Replacing j+1 by t and inserting alfa and beta with opposite signs we get:

```
(6.47)      P(t,x) = x×P(t-1,x)+alfa×P(t-1,x)+beta×P(t-2,x)
```

If we insert the expansion in equation (6.44) into equation (6.47) and the similar expansions of P(t-1,x) and P(t-2,x) and then collect equal powers in x, we get the equations:

```
(6.48) POLC[t,t]   = POLC[t-1,t-1]
       POLC[t,t-1] = POLC[t-1,t-2]+alfa×POLC[t-1,t-1]
       POLC[t,t-2] = POLC[t-1,t-3]+alfa×POLC[t-1,t-2]+beta×POLC[t-2,t-2]
       POLC[t,t-3] = POLC[t-1,t-4]+alfa×POLC[t-1,t-3]+beta×POLC[t-2,t-3]
       .....
       .....
       POLC[t,2]   = POLC[t-1,1]  +alfa×POLC[t-1,2]  +beta×POLC[t-2,2]
       POLC[t,1]   = POLC[t-1,0]  +alfa×POLC[t-1,1]  +beta×POLC[t-2,1]
       POLC[t,0]   =               alfa×POLC[t-1,0]  +beta×POLC[t-2,0]
```

From this we can easily see, that it is possible to generate the polynomial coefficients, POLC[t,0:t] by a for-statement of the form:

```
(6.49)      for e := t-1 step -1 until 0 do
            POLC[t,e] := POLC[t-1,e-1]+alfa×POLC[t-1,e]+beta×POLC[t-2,e];
```

This for-statement must be inserted at a point where alfa and beta are available for the corresponding t-value. It is also necessary to assign certain start values to some of the elements in POLC:

POLC[i,j]:

i:	j:-1	0	1	2	3	4	...	DEG
-1		0						
0	0	1	0					
1	0		1	0				
2	0			1	0			
3	0				1	0		
4	0					1		
.....								
.....								
DEG	0							1

The elements POLC[t,t] are set to 1 because the orthogonal polynomial of degree t just generated is the only source of x to the power of t. For some values of e and t some of the POLC-elements in equation (6.49) will lie outside the limits of POLC[0:DEG,0:DEG]. This can be avoided by enlarging the array to: POLC[-1:DEG,-1:DEG] and assigning zeroes to these elements as shown above. The upper triangle is not used.

6.4. The Procedure ORPOLGEN. If the calculation of the POLC-array is included in the program d-372 after the method described above and we rearrange the calculations into a procedure with suitable formal parameters we get a procedure with the declaration:

```
procedure ORPOLGEN(OBS, DEG, X, ORPOL, POLC, SSQPOL);
value OBS, DEG;
integer OBS, DEG;
array X, ORPOL, POLC, SSQPOL;
begin
   integer e, k, t;
   real alfa, beta, OLDSQSUM, XPROD, D;
   for k := 1 step 1 until OBS do
   begin
      ORPOL[-2,k] := 1;
      ORPOL[-1,k] := 0;
   end for k;
```

```
    for t := 0 step 1 until DEG do
    begin
       POLC[t,t] := 1;
       POLC[t-1,t] := POLC[t,-1] := 0;
    end for t;
    alfa := 0;
    beta := OLDSQSUM := 1;
    for t := 0 step 1 until DEG do
    begin
       SSQPOL[t] := XPROD := 0;
       for k := 1 step 1 until OBS do
       begin
          D := ORPOL[t,k] :=
          ORPOL[t-2,k]×beta + ORPOL[t-1,k]×(X[k] + alfa);
          D := D↑2;
          SSQPOL[t] := SSQPOL[t] + D;
          XPROD := XPROD + D×X[k];
       end for k;
       for e := t - 1 step -1 until 0 do
       POLC[t,e] := POLC[t-1,e]×alfa +
       POLC[t-2,e]×beta + POLC[t-1,e-1];
       beta := -SSQPOL[t]/OLDSQSUM;
       OLDSQSUM := SSQPOL[t];
       alfa := -XPROD/OLDSQSUM;
    end for t;
 end ORPOLGEN;
```

The formal parameters in ORPOLGEN are:

OBS	integer	Number of observations.
DEG	integer	Required polynomial degree.
X	array	[1:OBS]. The list of x-values.

ORPOL array [-2:DEG,1:OBS]. The calculated set of values of the orthogonal polynomials. ORPOL[t,i] is the value of the orthogonal polynomial of degree t for the x-value no. i.

POLC array [-1:DEG,-1:DEG]. The calculated array of polynomial coefficients. POLC[i,j] is the coefficient to x in the power j in the orthogonal polynomial of degree i.

SSQPOL array [0:DEG]. The calculated sum of the squares of the orthogonal polynomials. SSQPOL[t] corresponds to the degree t.

All elements in the procedure have been explained on the preceding pages. The initial setting covers ORPOL[-2,1:OBS], ORPOL[-1,1:OBS], the diagonal in POLC, the first column in POLC, and the line immediately above the diagonal in POLC. We then have the setting of start values of alfa, beta, and SSQSUM and the for-statement in t, exactly as in d-372. The for-statement in e (equation (6.49)) has been added here inside the t-for-statement.

A simple example of the use of the procedure ORPOLGEN is shown in the program d-369 below:

```
Program d-369. Test of ORPOLGEN
begin
   integer i, j, g;
   array X[1:5], ORPOL[-2:4,1:5], POLC[-1:4,-1:4], SSQPOL[0:4];
   copy ORPOLGEN<
   for i := 1 step 1 until 5 do X[i] := 180+20×i;
   select(8);
   writetext(⟨<
Output d-369:
⟩);
   for g := 0 step 1 until 4 do
   begin
      ORPOLGEN(5, g, X, ORPOL, POLC, SSQPOL);
      writetext(⟨<
   Degree:⟩);
      write(⟨dd⟩, g);
      writecr;
      writetext(⟨<
ORPOL:⟩);
      for i := 1 step 1 until 5 do
      begin
         writecr;
         for j := 0 step 1 until g do
         write(⟨  -d.dddd₁₀-dd⟩, ORPOL[j,i]);
      end for i;
      writecr;
      writetext(⟨⟨
POLC:⟩);
      for i := 0 step 1 until g do
      begin
         writecr;
         for j := 0 step 1 until i do
         write(⟨  -d.dddd₁₀-dd⟩, POLC[i,j]);
      end for i;
      writecr;
      writetext(⟨⟨
SSQPOL:⟩);
      writecr;
      for i := 0 step 1 until g do write(⟨  -d.dddd₁₀-dd⟩, SSQPOL[i]);
      writecr;
      if g mod 2 = 1 then writechar(72);
   end for g;
end;
```

The program operates in very nearly the same way as did the program d-372. A for-statement counts the polynomial degree, g, from 0 to 4 and for each value of g a call is made of ORPOLGEN. The program then prints the elements in the three arrays: ORPOL, POLC, and SSQPOL:

Output d-369:

```
   Degree: 0

ORPOL:
   1.0000
   1.0000
   1.0000
   1.0000
   1.0000

POLC:
   1.0000

SSQPOL:
   5.0000
   Degree: 1

ORPOL:
   1.0000        -4.0000 10 1
   1.0000        -2.0000 10 1
   1.0000         0.0000
   1.0000         2.0000 10 1
   1.0000         4.0000 10 1

POLC:
   1.0000
 -2.4000 10 2    1.0000

SSQPOL:
   5.0000         4.0000 10 3

   Degree: 2

ORPOL:
   1.0000        -4.0000 10 1    8.0000 10 2
   1.0000        -2.0000 10 1   -4.0000 10 2
   1.0000         0.0000        -8.0000 10 2
   1.0000         2.0000 10 1   -4.0000 10 2
   1.0000         4.0000 10 1    8.0000 10 2

POLC:
   1.0000
 -2.4000 10 2    1.0000
  5.6800 10 4  -4.8000 10 2    1.0000

SSQPOL:
   5.0000         4.0000 10 3    2.2400 10 6
   Degree: 3

ORPOL:
   1.0000       -4.0000 ₁₀ 1   8.0000 ₁₀ 2  -9.6000 ₁₀ 3
   1.0000       -2.0000 ₁₀ 1  -4.0000 ₁₀ 2   1.9200 ₁₀ 4
   1.0000        0.0000       -8.0000 ₁₀ 2   0.0000
   1.0000        2.0000 ₁₀ 1  -4.0000 ₁₀ 2  -1.9200 ₁₀ 4
   1.0000        4.0000 ₁₀ 1   8.0000 ₁₀ 2   9.6000 ₁₀ 3

POLC:
   1.0000
  -2.4000 ₁₀ 2   1.0000
   5.6800 ₁₀ 4  -4.8000 ₁₀ 2   1.0000
  -1.3498 ₁₀ 7   1.7144 ₁₀ 5  -7.2000 ₁₀ 2   1.0000

SSQPOL:
   5.0000        4.0000 ₁₀ 3   2.2400 ₁₀ 6   9.2160 ₁₀ 8

   Degree: 4

ORPOL:
   1.0000       -4.0000 ₁₀ 1   8.0000 ₁₀ 2  -9.6000 ₁₀ 3   5.4857 ₁₀ 4
   1.0000       -2.0000 ₁₀ 1  -4.0000 ₁₀ 2   1.9200 ₁₀ 4  -2.1943 ₁₀ 5
   1.0000        0.0000       -8.0000 ₁₀ 2   0.0000        3.2914 ₁₀ 5
   1.0000        2.0000 ₁₀ 1  -4.0000 ₁₀ 2  -1.9200 ₁₀ 4  -2.1943 ₁₀ 5
   1.0000        4.0000 ₁₀ 1   8.0000 ₁₀ 2   9.6000 ₁₀ 3   5.4857 ₁₀ 4

POLC:
   1.0000
  -2.4000 ₁₀ 2   1.0000
   5.6800 ₁₀ 4  -4.8000 ₁₀ 2   1.0000
  -1.3498 ₁₀ 7   1.7144 ₁₀ 5  -7.2000 ₁₀ 2   1.0000
   3.2161 ₁₀ 9  -5.4446 ₁₀ 7   3.4383 ₁₀ 5  -9.6000 ₁₀ 2   1.0000

SSQPOL:
   5.0000        4.0000 ₁₀ 3   2.2400 ₁₀ 6   9.2160 ₁₀ 8   2.1065₁₀ 11
```

From this information we can write down how the orthogonal polynomials look like for this data material:

(6.50) $P(0,x) = 1$

$P(1,x) = -240 + x$

$P(2,x) = 56800 - 480 \times x + x\uparrow 2$

$P(3,x) = -1.3498_{10}7 + 1.7144_{10}5 \times x - 720 \times x\uparrow 2 + x\uparrow 3$

$P(4,x) = 3.2161_{10}9 - 5.4446_{10}7 \times x + 3.4383_{10}5 \times x\uparrow 2 - 960 \times x\uparrow 3 + x\uparrow 4$

The coefficients are not given with full accuracy here.

7. FUNCTIONS OF A SINGLE VARIABLE

7.1. Direct Use of ORPOLGEN. When the procedure ORPOLGEN is available the polynomial approximation to a function of a single variable is very simple. After the call of ORPOLGEN we first use equation (6.24) for calculation of the regression coefficients, a[0:DEG], and then convert these coefficients into the conventional polynomial coefficients by use of the array POLC and equation (6.46). The program d-373 shown below has a procedure POLYA which operates according to this method. The program reads the five X-Y-values in Table 2 (page 49) and calculates polynomial approximations to this function of a degree varying from 0 to 4. The procedure POLYA is not intended to be a standard procedure because it is possible to merge ORPOLGEN and POLYA into a single simpler procedure. This merging is explained in section 7.2 and the resulting standard procedure, POLY1, in section 7.3.

The program is:

Program d-373. Test of POLYA.
```
begin
   integer g, i, p;
   real ERROR, YCALC, DEV;
   array X, Y[1:5], COEF[0:4];
   copy ORPOLGEN<
   procedure POLYA(OBS, DEG, X, Z, COEF);
   value OBS, DEG;
   integer OBS, DEG;
   array X, Z, COEF;
   begin
      integer h, i, p;
      real SUM;
      array ORPOL[-2:DEG,1:OBS], POLC[-1:DEG,-1:DEG],
      SSQPOL, a[0:DEG];
      ORPOLGEN(OBS, DEG, X, ORPOL, POLC, SSQPOL);
      for p := 0 step 1 until DEG do
      begin
         SUM := 0;
         for i := 1 step 1 until OBS do
         SUM := SUM + Z[i]×ORPOL[p,i];
         a[p] := SUM/SSQPOL[p];
      end for p;
```

```
    for h := 0 step 1 until DEG do
    begin
      SUM := 0;
      for p := h step 1 until DEG do
      SUM := SUM + a[p]×POLC[p,h];
      COEF[h] := SUM;
    end for h;
  end POLYA;
  for i := 1 step 1 until 5 do
  begin
    X[i] := 180 + 20×i;
    Y[i] := case i of (38.8210, 40.9274, 42.9013, 44.7590, 46.5139);
  end for i;
  select(8);
  writetext(⌊<
Output d-373:
⌋);
  for g := 0 step 1 until 4 do
  begin
    writetext(⌊<
Degree:⌋);
    write(⌊dd⌋, g);
    writecr;
    POLYA(5, g, X, Y, COEF);
    writetext(⌊<
 X      Y, inp.  Y, calc. error
⌋);
    ERROR := 0;
    for i := 1 step 1 until 5 do
    begin
      writecr;
      YCALC := 0;
      for p := g step -1 until 0 do
      YCALC := YCALC×X[i] + COEF[p];
      DEV := YCALC - Y[i];
      ERROR := ERROR + DEV↑2;
      write(⌊ddd.dddd⌋, X[i]);
      write(⌊dddd.dddd⌋, Y[i], YCALC);
```

```
         write(⟨-dd.dddddd⟩, DEV);
      end for i;
      writecr;
      writetext(⟨<                         Mean  ⟩);
      write(⟨ddd.dddddd⟩, sqrt(ERROR/4));
      writecr;
      writetext(⟨<deg:  COEF⟩);
      writecr;
      for p := 0 step 1 until g do
      begin
         writecr;
         write(⟨ddd⟩, p);
         write(⟨  -d.dddddddd₁₀-dd⟩, COEF[p]);
      end for p;
      writecr;
      if g mod 2 = 1 then writechar(72);
   end for g;
end;
```

The procedure POLYA has five formal parameters of which the three first ones (OBS, DEG, and X) are the same as in ORPOLGEN. The other parameters are:

Z array [1:OBS]. This is the set of function values corresponding to the set of values of the independent variables, X[1:OBS]. In the call of POLYA in d-373, Y[1:OBS] is inserted instead of z.

COEF array [0:DEG]. The array of conventional polynomial coefficients generated by the procedure.

POLYA works as follows: Storage is reserved for the usual arrays required by ORPOLGEN: ORPOL, POLC, and SSQPOL. The array of regression coefficients, a[0:DEG], for use with the orthogonal polynomials is also declared. A call is then made of ORPOLGEN. We then have two for-statements of which the first counts p from 0 to DEG and calculates a[p] as given by equation (6.24). The value of SUM(Z[i]×P(p,X[i])) is found by an inner for-statement counting in i. The sum is divided by SSQPOL[p]. The second for-statement counts h from 0 to DEG and calculates COEF[h] after equation (6.46) by summation of a[p]×POLC[p,h] with p going from h to DEG.

Output from d-373 is:

Output d-373:

```
Degree: 0

  X        Y, inp.  Y, calc. error


200.0000  38.8210  42.7845  3.963520
220.0000  40.9274  42.7845  1.857120
240.0000  42.9013  42.7845 -0.116780
260.0000  44.7590  42.7845 -1.974480
280.0000  46.5139  42.7845 -3.729380
                    Mean     3.040513
deg:  COEF

  0   4.27845198 10 1

Degree: 1

  X        Y, inp.  Y, calc. error


200.0000  38.8210  38.9410  0.120040
220.0000  40.9274  40.8628 -0.064620
240.0000  42.9013  42.7845 -0.116780
260.0000  44.7590  44.7063 -0.052740
280.0000  46.5139  46.6280  0.114100
                    Mean     0.109571
deg:  COEF

  0   1.97236404 10 1
  1   9.60869968 10-2
```

```
Degree: 2

  X        Y, inp.  Y, calc. error

200.0000  38.8210  38.8240  0.003011
220.0000  40.9274  40.9213 -0.006106
240.0000  42.9013  42.9015  0.000248
260.0000  44.7590  44.7648  0.005774
280.0000  46.5139  46.5110 -0.002929
                     Mean    0.004699
deg:  COEF

  0    1.14146012 10 1
  1    1.66304229 10-1
  2   -1.46285902 10-4

Degree: 3

  X        Y, inp.  Y, calc. error

200.0000  38.8210  38.8210  0.000041
220.0000  40.9274  40.9272 -0.000166
240.0000  42.9013  42.9015  0.000248
260.0000  44.7590  44.7588 -0.000166
280.0000  46.5139  46.5139  0.000040
                     Mean    0.000173
deg:  COEF

  0    7.23886529
  1    2.19342411 10-1
  2   -3.69031416 10-4
  3    3.09368769 10-7
```

```
Degree: 4

  X        Y, inp.  Y, calc. error

200.0000  38.8210  38.8210 -0.000001
220.0000  40.9274  40.9274 -0.000001
240.0000  42.9013  42.9013 -0.000001
260.0000  44.7590  44.7590 -0.000001
280.0000  46.5139  46.5139 -0.000002
                    Mean     0.000001
deg:  COEF

  0   4.80720784
  1    2.60508787 ₁₀-1
  2  -6.29000026 ₁₀-4
  3    1.03522428 ₁₀-6
  4  -7.56099492₁₀-10
```

As would be expected, the fourth-order polynomial fits closely to the five function values, within the calculation accuracy obtainable in the GIER computer.

7.2. Improved Method. When the procedures ORPOLGEN and POLYA are used on large data materials, the storage requirements for the three arrays: ORPOL, POLC, and SSQPOL become fairly large:

ORPOL	requires	OBS×(DEG+3)	cells
POLC	-	(DEG+2)↑2	-
SSQPOL	-	DEG+1	-

Especially the requirements for the ORPOL array may be prohibitive for use in a small computer. We shall now see how this storage amount can be reduced.

In ORPOLGEN the main for-statement counts t from 0 to DEG. For each value of t the procedure operates on the three rows in ORPOL:

ORPOL[t,1:OBS], ORPOL[t-1,1:OBS], and ORPOL[t-2,1:OBS]

In POLYA the first for-statement also counts from 0 to DEG, here in the variable p, and for each value of p we use the same three rows:

ORPOL[p,1:OBS], ORPOL[p-1,1:OBS], and ORPOL[p-2,1:OBS]

It should be possible to merge the two for-statements in ORPOLGEN and POLYA into a single for-statement and using only 3 rows of the ORPOL array. As the t-for-statement calculates ORPOL[t,k] from ORPOL[t-1,k] and ORPOL[t-2,k], it is actually possible to come through with only two rows of ORPOL. We denote these rows ORPOL1 and ORPOL2:

ORPOL1[1:OBS]	corresponds to	ORPOL[t-1,1:OBS]
ORPOL2[1:OBS]	- -	ORPOL[t-2,1:OBS]

Whenever a new value of ORPOL[t,k] has been calculated, it is stored in a work cell, and we perform the movements:

```
(7.1)     ORPOL2[k] := ORPOL1[k];
          ORPOL1[k] := work cell;
```

A further study of ORPOLGEN and POLYA reveals that it is also sufficient to have available only two rows of the POLC-array. These are denoted:

POLC1[-1:DEG]	corresponds to	POLC[t-1,-1:DEG]
POLC2[-1:DEG]	- -	POLC[t-2,-1:DEG]

Similarly, SSQPOL[0:DEG] can be replaced by a single variable, also denoted SSQPOL, and corresponding to SSQPOL[t].

The procedure POLYB has been prepared by this merging of ORPOLGEN and POLYA. The declaration is:

```
procedure POLYB(OBS, DEG, X, Z, COEF);
value OBS, DEG;
integer OBS, DEG;
array X, Z, COEF;
begin
   integer e, k, t;
   real alfa, beta, D, OLDSQSUM, SSQPOL, XPROD, ZPROD;
   array ORPOL1, ORPOL2[1:OBS], POLC1, POLC2[-1:DEG];
```

```
    for k := 1 step 1 until OBS do
    begin
        ORPOL2[k] := 1;
        ORPOL1[k] := 0;
    end for k;
    alfa := POLC1[-1] := 0;
    beta := OLDSQSUM := 1;
    for t := 0 step 1 until DEG do
    begin
        XPROD := ZPROD := SSQPOL := 0;
        for k := 1 step 1 until OBS do
        begin
            D := ORPOL2[k]×beta;
            ORPOL2[k] := ORPOL1[k];
            D := ORPOL1[k] := D + ORPOL1[k]×(X[k]+alfa);
            ZPROD := ZPROD + Z[k]×D;
            D := D↑2;
            SSQPOL := SSQPOL + D;
            XPROD := XPROD + D×X[k];
        end for k;
        COEF[t] := ZPROD/SSQPOL;
        POLC1[t] := 1;
        POLC2[t] := 0;
        for e := t - 1 step -1 until 0 do
        begin
            D := POLC2[e]×beta;
            POLC2[e] := POLC1[e];
            POLC1[e] := POLC1[e]×alfa + D + POLC1[e-1];
            COEF[e] := COEF[e] + COEF[t]×POLC1[e];
        end for e;
        beta := -SSQPOL/OLDSQSUM;
        OLDSQSUM := SSQPOL;
        alfa := - XPROD/OLDSQSUM;
    end for t;
end POLYB;
```

The procedure POLYB contains the rows ORPOL1 and ORPOL2 of the original ORPOL array in ORPOLGEN and also the rows POLC1 and POLC2 from the POLC array in ORPOLGEN. POLYB starts with setting of the elements in ORPOL2 to 1 and the elements in ORPOL1 to 0. This is equivalent to the setting of the rows ORPOL[-2,k] and ORPOL[-1,k] in ORPOLGEN. Of the initial setting of some of the elements in POLC in ORPOLGEN we only carry out the setting of POLC[-1] to zero in POLYB. The remaining settings are made inside the for-statement. The start values of alfa, beta, and OLDSQSUM are the same in ORPOLGEN and POLYB.

The for-statement in t starts by resetting XPROD and SSQPOL to zero as in ORPOLGEN. SSQPOL is now a simple variable. We also set ZPROD to zero. This corresponds to the variable: SUM in the p-for-statement in POLYA. The for-statement counting k from 1 to OBS is the same as in the procedure ORPOLGEN, except that we now operate on the two actual rows of ORPOL as explained above. The summation of Z[k]×ORPOL[t,k] is also included here. After the k-for-statement we calculate the new regression coefficient as:

```
(7.2)      COEF[t] := ZPROD/SSQPOL;
```

This corresponds to the statement:

```
(7.3)      a[p] := SUM/SSQPOL[p];
```

in POLYA. It is not necessary to have two separate arrays:

```
           a, COEF[0:DEG]
```

because the a-element of highest degree is always equal to the COEF-element of highest degree and the recalculation from a to COEF can be made in a condensed way as shown below.

We then set the values of POLC1[t] and POLC2[t] which were not included in the initial setting. Then comes the for-statement which counts e from t-1 to 0. In ORPOLGEN this contained only a single statement:

```
(7.4)      POLC[t,e] := POLC[t-1,e]×alfa +
           POLC[t-2,e]×beta + POLC[t-1,e-1];
```

In POLYB the statement is broken down into more statements, firstly because we use only two arrays: POLC1, POLC2[-1:DEG] to handle the rows t, t-1, and t-2 in POLC[-1:DEG,-1:DEG] in ORPOLGEN. But we have also included the statement:

```
(7.5)      COEF[e] := COEF[e] + COEF[t]×POLC1[e];
```

which calculates the contribution of the just calculated COEF[t] = a[t] to the lower degree coefficients. If we compare with the last half of POLYA:

```
(7.6)      for h := 0 step 1 until DEG do
           begin
              SUM := 0;
              for p := h step 1 until DEG do
              SUM := SUM + a[p]×POLC[p,h];
              COEF[h] := SUM;
           end for h;
```

we can see, that equation (7.5) performs that part of this double for-statement which can be performed with the value of a[p] just calculated. In this way, no special a-array is necessary.

When POLYB is inserted instead of POLYA in program d-373 the calculation result is exactly the same.

Before we discuss the final version of the polynomial approximation procedure (POLY1) there is one small refinement which requires explanation. On page 50 it was discussed how the expansion in orthogonal polynomials:

```
(7.7)      y = a0×P0(x)+a1×P1(x)+a2×P2(x)+a3×P3(x);
```

has the effect that addition of one more term, here a4×P4(x), does not change the numerical values of the lower degree coefficients, a0, a1, and a3. This means, that if we after the calculation of a0 subtract the value of a0×P0(x[i]) from all the corresponding y[i]-values, this should have no influence upon a1, a2, etc. When a1 has been calculated we can then subtract a1×P1(x[i]) from what remains of the original y[i]-values. If this gradual reduction of the y-values is continued, the order of magnitude of the y-values will become smaller, and we may expect that the accuracy in the summations leading to the higher a-values may be improved. There is, of course, a risk that the subtractions may cause numerical errors. This must be found out from experiments.

The procedure POLYC is similar to POLYB, except that it contains the subtraction explained above. The declaration is:

```
procedure POLYC(OBS, DEG, X, Z, COEF);
value OBS, DEG;
integer OBS, DEG;
array X, Z, COEF;
begin
   integer e, k, t;
   real alfa, beta, D, OLDSQSUM, SSQPOL, XPROD, ZPROD, olda;
   array ORPOL1, ORPOL2, error[1:OBS], POLC1, POLC2[-1:DEG];
   for k := 1 step 1 until OBS do
   begin
      ORPOL2[k] := 1;
      ORPOL1[k] := 0;
      error[k] := Z[k];
   end for k;
   alfa := olda := POLC1[-1] := 0;
   beta := OLDSQSUM := 1;
   for t := 0 step 1 until DEG do
   begin
      XPROD := ZPROD := SSQPOL := 0;
      for k := 1 step 1 until OBS do
      begin
         error[k] := error[k] - olda×ORPOL1[k];
         D := ORPOL2[k]×beta;
         ORPOL2[k] := ORPOL1[k];
         D := ORPOL1[k] := D + ORPOL1[k]×(X[k]+alfa);
         ZPROD := ZPROD + error[k]×D;
         D := D↑2;
         SSQPOL := SSQPOL + D;
         XPROD := XPROD + D×X[k];
      end for k;
```

```
        COEF[t] := olda := ZPROD/SSQPOL;
        POLC1[t] := 1;
        POLC2[t] := 0;
        for e := t - 1 step -1 until 0 do
        begin
           D := POLC2[e]×beta;
           POLC2[e] := POLC1[e];
           POLC1[e] := POLC1[e]×alfa + D + POLC1[e-1];
           COEF[e] := COEF[e] + COEF[t]×POLC1[e];
        end for e;
        beta := -SSQPOL/OLDSQSUM;
        OLDSQSUM := SSQPOL;
        alfa := - XPROD/OLDSQSUM;
     end for t;
  end POLYC;
```

When POLYC is inserted in d-373 instead of POLYA or POLYB we get the same results as regards the mean errors, but the calculated coefficients are slightly different:

deg:	POLYA,POLYB	POLYC
0	4.27845198 $_{10}1$	4.27845198 $_{10}1$
0	1.97236404 $_{10}1$	1.97236400 $_{10}1$
1	9.60869968 $_{10}{-2}$	9.60869992 $_{10}{-2}$
0	1.14146012 $_{10}1$	1.14146110 $_{10}1$
1	1.66304229 $_{10}{-1}$	1.66304145 $_{10}{-1}$
2	-1.46285902 $_{10}{-4}$	-1.46285720 $_{10}{-4}$
0	7.23886529	7.23878530
1	2.19342411 $_{10}{-1}$	2.19343467 $_{10}{-1}$
2	-3.69031416 $_{10}{-4}$	-3.69036025 $_{10}{-4}$
3	3.09368769 $_{10}{-7}$	3.09375424 $_{10}{-7}$

0	4.80720784	4.80988443
1	2.60508787 $_{10}$-1	2.60463176 $_{10}$-1
2	-6.29000026 $_{10}$-4	-6.28709928 $_{10}$-4
3	1.03522428 $_{10}$-6	1.03440809 $_{10}$-6
4	-7.56099492 $_{10}$-10	-7.55242360 $_{10}$-10

From these figures we cannot see which of the two methods is the more accurate. We can check this by taking some data for which the coefficients are known exactly. We take a fourth-order polynomial with the coefificnets: ?

c[0] = 100
c[1] = -1 $_{10}$-1
c[2] = 1 $_{10}$-4
c[3] = -1 $_{10}$-7
c[4] = 1 $_{10}$-10

If we calculate the value of this polynomial for the x-values we have used before we get the values:

X	Y
200	83.360000
220	82.009456
240	80.709376
260	79.459376
280	78.259456

When these data are run on program d-373 with the necessary modification to compare POLYA and POLYC, we get:

deg:	POLYA	POLYC
0	9.99943285 $_{10}$ 1	9.99998579 $_{10}$ 1
1	-9.99036245 $_{10}$-2	-9.99976830 $_{10}$-2
2	9.93889009 $_{10}$-5	9.99858682 $_{10}$-5
3	-9.82865231 $_{10}$-8	-9.99620147 $_{10}$-8
4	9.82072620 $_{10}$-11	9.99620167 $_{10}$-11

These coefficients clearly indicate the improved accuracy of the subtraction feature which is, therefore, included in POLY1.

7.3. The Procedure POLY1. This is the recommended final version of the orthogonal polynomial approximation procedure for a single variable. It contains the additional possibility of weighting the observations. The declaration of POLY1 is:

```
procedure POLY1(OBS, DEG, X, Z, W, COEF, WEIGH);
value OBS, DEG, WEIGH;
integer OBS, DEG;
boolean WEIGH;
array X, Z, W, COEF;
begin
   integer e, k, t;
   real alfa, beta, D, OLDSQSUM, SSQPOL, XPROD, ZPROD, olda;
   array ORPOL1, ORPOL2, error[1:OBS], POLC1, POLC2[-1:DEG];
   for k := 1 step 1 until OBS do
   begin
      ORPOL2[k] := 1;
      ORPOL1[k] := 0;
      error[k] := Z[k];
   end for k;
   alfa := olda := POLC1[-1] := 0;
   beta := OLDSQSUM := 1;
   for t := 0 step 1 until DEG do
   begin
      XPROD := ZPROD := SSQPOL := 0;
      for k := 1 step 1 until OBS do
      begin
         error[k] := error[k] - olda×ORPOL1[k];
         D := ORPOL2[k]×beta;
         ORPOL2[k] := ORPOL1[k];
         D := ORPOL1[k] := D + ORPOL1[k]×(X[k]+alfa);
         if WEIGH then D := D×W[k];
         ZPROD := ZPROD + error[k]×D;
         D := D×ORPOL1[k];
         SSQPOL := SSQPOL + D;
```

```
            XPROD := XPROD + D×X[k];
        end for k;
        COEF[t] := olda := ZPROD/SSQPOL;
        POLC1[t] := 1;
        POLC2[t] := 0;
        for e := t - 1 step -1 until 0 do
        begin
            D := POLC2[e]×beta;
            POLC2[e] := POLC1[e];
            POLC1[e] := POLC1[e]×alfa + D + POLC1[e-1];
            COEF[e] := COEF[e] + COEF[t]×POLC1[e];
        end for e;
        beta := -SSQPOL/OLDSQSUM;
        OLDSQSUM := SSQPOL;
        alfa := - XPROD/OLDSQSUM;
    end for t;
end POLY1;
```

The procedure POLY1 contains two additional formal parameters:

W	array	[1:OBS]. These are the weights to be applied to the observations.
WEIGH	boolean	If WEIGH is true, the weights W[1:OBS] will be used.

If WEIGH is false, the weights are not used. We can then save the storage space for the W-array by inserting X or Z or another array as an actual parameter instead of W.

The weighting method is partly described by Lapidus (1962), page 325. The weight factor, W[i], of observation no. i is used in the following three sums:

(7.8) ZPROD = SUM(W[i]×Z[i]×P(t,X[i]))

(7.9) XPROD = SUM(W[i]×X[i]×P(t,X[i])↑2)

(7.10) SSQPOL = SUM(W[i]×P(t,X[i])↑2)

Here Z[i] is the function value of observation no. i, or rather its residual after subtraction of the lower order contributions. POLY1 is different from POLYC in that we have added the statement:

```
if WEIGH then D := D×W[k];
```

The variable, D, contains P(t,X[k]) before this statement. After addition of D×error[k] to ZPROD, POLY1 calculates:

```
D := D×ORPOL1[k];
```

instead of D := D↑2 in POLYC. The value of D then becomes P(t,X[k])↑2 if no weights are used and W[k]×P(t,X[k])↑2 if weights are used.

The use of POLY1 with weighting has been tested on our standard example, Table 2 on page 49. If we insert the weights 1 for all observations except one, say no. 4, and calculate third order polynomials for different values of the weight on point no. 4, we get the results:

		Calculated deviations		
X	Y	W[4]=0	W[4]=1	W[4]=100
200	38.8210	0.000000	0.000041	0.000053
220	40.9274	0.000000	-0.000166	-0.000215
240	42.9013	-0.000001	0.000248	0.000321
260	44.7590	-0.000726	-0.000166	-0.000003
280	46.5139	-0.000001	0.000041	0.000053
	Mean:	0.000363	0.000173	0.000197

When the weight of point no. 4 is zero we get a close fit to the other points. For the weight unity we get the same result as without weights. Finally, a large weight (100) gives a close fit to point no. 4 with the errors distributed on the other points. If it is required that the approximation must pass exactly through one or more of the points or satisfy other conditions, e.g. concerning the derivatives, it is not recommended to solve this by use of extreme weights, which could give rise to numerical errors. Instead of this, the required polynomial should be written as the sum of two polynomials of which the first satisfies the conditions. The second polynomial is selected as the product of two polynomials: The first polynomial being zero in the special points and the second being calculated in the usual way. This method is further discussed in Chapter 12.

8. FUNCTIONS OF TWO VARIABLES

The use of orthogonal polynomials in the approximation of functions of two or more variables is practically restricted to the case when the function values are available in a regular pattern of the independent variables. For two variables we can select a test example which is an extension of Table 2 on page 49:

X1:

X2:	200	220	240	260	280
400	38.8210	40.9274	42.9013	44.7590	46.5139
408	36.8153	38.8940	40.8475	42.6906	44.4357
416	34.8732	36.9184	38.8457	40.6688	42.3990
424	32.9982	35.0047	36.9007	38.6987	40.4090

Table 3

The table gives the ammonia equilibrium yield as a function of the pressure (X1) and the temperature (X2). It is not necessary that the values of X1 and X2 are equidistant as in this table, but all function values must be available.

When a function of two variables is to be approximated by means of orthogonal polynomials we write the required polynomial as the sum of a number of terms:

$$(8.1)\qquad \begin{array}{lllll} Z = T[0,0] & + T[1,0] & + T[2,0] & + \ldots\ldots\ldots + T[DEG1,0] \\ +T[0,1] & + T[1,1] & + T[2,1] & + \ldots\ldots\ldots + T[DEG1,1] \\ +T[0,2] & + T[1,2] & + T[2,2] & + \ldots\ldots\ldots + T[DEG1,2] \\ +\ldots\ldots & \ldots\ldots & & \\ \ldots\ldots & \ldots\ldots & & \\ \ldots\ldots & \ldots\ldots & & \\ +T[0,DEG2] & +T[1,DEG2] & +T[2,DEG2] & + \ldots\ldots\ldots + T[DEG1,DEG2] \end{array}$$

DEG1 and DEG2 are the maximum degrees in which X1 and X2 should be present in the final polynomial. Each term, T, is written as:

$$\text{(8.2)}\qquad T[p1,p2] = a[p1,p2]\times P1(p1,X1)\times P2(p2,X2)$$

Here, a[p1,p2] is the regression coefficient corresponding to the degrees p1 and p2 of X1 and X2, respectively. P1(p1,X1) is the value of the orthogonal polynomial of degree p1 for a given value of X1. For the data material in Table 3 the orthogonal polynomials P1(p1,X1) are exactly identical to the orthogonal polynomials considered in the two previous chapters. The third factor, P2(p2,X2) is the orthogonal polynomial of degree p2 and calculated on the basis of the values of the second variable, X2.

When the OBS1 values of X1 are known (OBS1 = 5 in Table 3), we can make a call of ORPOLGEN and generate the required values of the orthogonal polynomials, the polynomial coefficients, and the sum of the squares of the orthogonal polynomials. The same can be done for the OBS2 values of X2 (OBS2 = 4 in Table 3).

The regression coefficients, a[p1,p2], can be calculated in the same way as was shown on page 51-54 for the one-dimensional case. Because of the orthogonality all terms outside the diagonal disappear. The formula for a[p1,p2] is very similar to equation (6.24) and becomes:

$$\text{(8.3)}\qquad a[p1,p2] = \begin{array}{l} \mathrm{SUM}(Z[i1,i2]\times P1(p1,X1[i1])\times P2(p2,X2[i2]))/ \\ (\mathrm{SUM}(P1(p1,X1[i1])\uparrow 2)\times \mathrm{SUM}(P2(p2,X2[i2])\uparrow 2)) \end{array}$$

The numerator of this formula contains the summation of all the function values, Z[i1,i2], multiplied by the corresponding values of the two orthogonal polynomials. The summation is made for all the OBS1×OBS2 function values, i1 being counted from 1 to OBS1 and i2 from 1 to OBS2. The denominator of equation (8.3) contains two factors: The sum of the squares of the orthogonal polynomials in the first variable and the similar sum of squares for the second variable.

When the orthogonal polynomial regression coefficients, a[p1,p2], have been found, they must be transformed into the conventional polynomial coefficients, just as for the one-dimensional case.

The expansion in the conventional polynomial can be written as:

```
(8.4)      Z =
    C[0,0]        +    C[1,0]×X1         +   C[2,0]×X1↑2          + .....
                                            +C[DEG1,0]×X1↑DEG1
   +C[0,1]×X2     +    C[1,1]×X1×X2      +   C[2,1]×X1↑2×X2       + .....
                                            +C[DEG1,1]×X1↑DEG1×X2
   +C[0,2]×X2↑2   +    C[1,2]×X1×X2↑2    +   C[2,2]×X1↑2×X2↑2     + .....
                                        .....+C[DEG1,2]×X1↑DEG1×X2↑2

   +.....
    .....
    .....
   +C[0,DEG2]×X2↑DEG2+C[1,DEG2]×X1×X2↑DEG2+C[2,DEG2]×X1↑2×X2↑DEG2+....
                                        .....+C[DEG1,DEG2]×X1↑DEG1×X2↑DEG2
```

We have assumed here, that the coefficients are written as the array C[0:DEG1,0:DEG2], and that all the (DEG1+1)×(DEG2+1) coefficients are used. For DEG1 = 3 and DEG2 = 2 we get the pattern (note that the first subscript here gives the column number and the second subscript the row number):

```
          C    C    C    C
          C    C    C    C
          C    C    C    C
```

The term of highest order contains X1↑3×X2↑2, i.e. the highest sum of the exponents is DEG1+DEG2 = 5. It is possible to restrict this sum of the exponents to a lower value. We use a parameter, DEGLIM, which is the maximum sum of the exponents. Examples are:

```
          C    C    C    C         C    C    C    C
          C    C    C    C         C    C    C
          C    C    C              C    C

             DEGLIM = 4               DEGLIM = 3
```

<u>8.1. The Procedure POLY22.</u> This procedure analyses a two-dimensional table as Table 3 on page 85 and calculates the coefficients in the approximation polynomial. The declaration is:

```
procedure POLY22(OBS1, OBS2, DEG1, DEG2, DEGLIM, X1, X2, Z, COEF);
value OBS1, OBS2, DEG1, DEG2, DEGLIM;
integer OBS1, OBS2, DEG1, DEG2, DEGLIM;
array X1, X2, Z, COEF;
begin
   integer h1, h2, i1, i2, p1, p2;
   real sum1, sum2;
   array ORPOL1[-2:DEG1,1:OBS1], ORPOL2[-2:DEG2,1:OBS2],
   POLC1[-1:DEG1,-1:DEG1], POLC2[-1:DEG2,-1:DEG2],
   SSQPOL1[0:DEG1], SSQPOL2[0:DEG2], a[0:DEG1,0:DEG2];
   ORPOLGEN(OBS1, DEG1, X1, ORPOL1, POLC1, SSQPOL1);
   ORPOLGEN(OBS2, DEG2, X2, ORPOL2, POLC2, SSQPOL2);
   for p1 := 0 step 1 until DEG1 do
   for p2 := 0 step 1 until DEG2 do
   begin
      sum2 := 0;
      if p1 + p2 ≤ DEGLIM then
      begin
         for i2 := 1 step 1 until OBS2 do
         begin
            sum1 := 0;
            for i1 := 1 step 1 until OBS1 do
            sum1 := sum1 + Z[i1,i2]×ORPOL1[p1,i1];
            sum2 := sum2 + sum1×ORPOL2[p2,i2];
         end for i2;
      end if p1 + p2 ≤ DEGLIM;
      a[p1,p2] := sum2/SSQPOL1[p1]/SSQPOL2[p2];
   end for p1 and p2;
   for h1 := 0 step 1 until DEG1 do
   for h2 := 0 step 1 until DEG2 do
   begin
      sum2 := 0;
```

```
        if h1 + h2 ≤ DEGLIM then
        begin
           for p1 := h1 step 1 until DEG1 do
           for p2 := h2 step 1 until DEG2 do
           begin
              if p1 + p2 ≤ DEGLIM then
              sum2 := sum2 + a[p1,p2]×POLC1[p1,h1]×POLC2[p2,h2];
           end for p1 and p2;
        end if h1 + h2 ≤ DEGLIM;
        COEF[h1,h2] := sum2;
     end for h1 and h2;
  end POLY22;
```

The parameters have all been explained above. COEF is the calculated coefficient array, C. The procedure reserves storage space for the two sets of arrays and the a-array:

```
ORPOL1[-2:DEG1,1:OBS1]    ORPOL2[-2:DEG2,1:OBS2]
POLC1[-1:DEG1,-1:DEG1]    POLC2[-1:DEG2,-1:DEG2]
SSQPOL1[0:DEG1]           SSQPOL2[0:DEG2]

a[0:DEG1,0:DEG2]
```

The procedure ORPOLGEN is called twice, once for each of the two variables. We then have a double for-statement in which p1 and p2 are counted from 0 to DEG1 and from 0 to DEG2, respectively. For each set of p1 and p2 the value of a[p1,p2] is found by a summation as given in equation (8.3). If p1+p2>DEGLIM, the summation is skipped, and the value of a[p1,p2] is set to zero.

A new double for-statement is then started in which h1 is counted from 0 to DEG1 and h2 from 0 to DEG2. For each set of h1 and h2 the conventional polynomial coefficient, COEF[h1,h2] is calculated by a summation of the terms:

(8.5) a[p1,p2]×POLC1[p1,h1]×POLC2[p2,h2]

in another double for-statement where p1 is counted from h1 to DEG1 and p2 from h2 to DEG2 with due regard to the limit: DEGLIM.

The program d-375 analyses the data in Table 3 on page 85 by a call of POLY22. The program is:

```
Program d-375. Test of POLY22
begin
   integer OBS1, OBS2, DEG1, DEG2, DEGLIM;
   copy ORPOLGEN<
   copy POLY22<
   copy POLYVAL2<
   select(17);
   writetext(⟨<
Read input to d-375:⟩);
   lyn;
   select(8);
   writetext(⟨<
Output d-375:
⟩);
   OBS1 := read integer;
   OBS2 := read integer;
   DEG1 := read integer;
   DEG2 := read integer;
   DEGLIM := read integer;
   begin comment inner block;
      integer i, j;
      real ERROR, y;
      array X1, E[1:OBS1], X2[1:OBS2], Z[1:OBS1, 1:OBS2],
      COEF[0:DEG1, 0:DEG2];
      for i := 1 step 1 until OBS1 do
      X1[i] := read real;
      for i := 1 step 1 until OBS2 do
      X2[i] := read real;
      for j := 1 step 1 until OBS2 do
      for i := 1 step 1 until OBS1 do
      Z[i,j] := read real;
      POLY22(OBS1, OBS2, DEG1, DEG2, DEGLIM,
      X1, X2, Z, COEF);
      ERROR := 0;
      writecr;
```

```
    for i := 1 step 1 until OBS1 do
    write(⟨-ddddddd⟩, X1[i]);
    writecr;
    for j := 1 step 1 until OBS2 do
    begin
      writecr;
      write(⟨dddd⟩, X2[j]);
      for i := 1 step 1 until OBS1 do
      write(⟨-ddd.dddd⟩, Z[i,j]);
      writecr;
      writetext(⟨<    ⟩);
      for i := 1 step 1 until OBS1 do
      begin
        y := POLYVAL2(DEG1, DEG2, X1[i], X2[j], COEF);
        write(⟨-ddd.dddd⟩, y);
        E[i] := y - Z[i,j];
        ERROR := ERROR + E[i]↑2;
      end for i;
      writecr;
      writetext(⟨<    ⟩);
      for i := 1 step 1 until OBS1 do
      write(⟨-ddd.dddd⟩, E[i]);
      writecr;
    end for j;
    writecr;
    writetext(⟨<Mean error:⟩);
    write(⟨-ddddd.ddddddd⟩, sqrt(ERROR/(OBS1×OBS2-2)));
    writecr;
    writetext(⟨<
Coefficients, COEF[i, j]:
⟩);
    writecr;
    writetext(⟨<j:⟩);
    for i := 0 step 1 until DEG1 do
    begin
      writetext(⟨<       i=⟩);
      write(⟨d⟩, i);
      writetext(⟨<       ⟩);
    end for i;
```

```
        writecr;
        for j := 0 step 1 until DEG2 do
        begin
           writecr;
           write(⟨dd⟩, j);
           for i := 0 step 1 until DEG1 do
           write(⟨  -d.ddddddd⏨-dd⟩, COEF[i, j]);
        end for j;
        writecr;
     end inner block;
  end;
```

Output from the program was:

```
Output d-375:

        200       220       240       260       280


 400   38.8210   40.9274   42.9013   44.7590   46.5139
       38.8212   40.9274   42.9018   44.7591   46.5142
        0.0002    0.0000    0.0005    0.0001    0.0003

 408   36.8153   38.8940   40.8475   42.6906   44.4357
       36.8148   38.8932   40.8470   42.6897   44.4349
       -0.0005   -0.0008   -0.0005   -0.0009   -0.0008

 416   34.8732   36.9184   38.8457   40.6688   42.3990
       34.8737   36.9189   38.8466   40.6694   42.3998
        0.0005    0.0005    0.0009    0.0006    0.0008

 424   32.9982   35.0047   36.9007   38.6987   40.4090
       32.9980   35.0044   36.9006   38.6983   40.4088
       -0.0002   -0.0003   -0.0001   -0.0004   -0.0002

Mean error:     0.0005585
```

Coefficients, COEF[i, j]:

j:	i=0	i=1	i=2	i=3
0	$1.98720552_{10}2$	$5.44347845_{10}-1$	$-2.84282004_{10}-3$	$3.57351551_{10}-6$
1	$-8.17635950_{10}-1$	$-5.24432988_{10}-4$	$9.08899986_{10}-6$	$-1.30466819_{10}-8$
2	$8.47337786_{10}-4$	$-7.20324280_{10}-7$	$-7.26069724_{10}-9$	$1.22148130_{10}-11$

Each observation point is printed as three numbers: The original point, the value calculated from the polynomial, and the deviation. The value of the polynomial is calculated by the procedure POLYVAL2, which is explained in section 11.1.

9. FUNCTIONS OF THREE VARIABLES

In order to illustrate the polynomial approximation method for functions of three variables we extend our Table 3 on page 85 to include a third variable, X3, the percentage of inerts in the gas. At the same time, we reduce the number of values of X1 from 5 to 3. Two values of X3 are used. The table is:

X3 = 0

X2:	X1 200	220	240
400	38.8210	40.9274	42.9013
408	36.8153	38.8940	40.8475
416	34.8732	36.9184	38.8457
424	32.9982	35.0047	36.9007

X3 = 10

X2:	X1: 200	220	240
400	31.6245	33.3405	34.9477
408	29.9900	31.6841	33.2752
416	28.4074	30.0746	31.6449
424	26.8795	28.5155	30.0608

Table 4

The extension of the calculation method to three variables is very simple. In equation (8.3) the following changes are necessary:

Replace a[p1,p2] by a[p1,p2,p3]
Replace Z[i1,i2] by Z[i1,i2,i3]
Add the factor P3(p3,X3[i3]) in the numerator summation.
Add the factor P3(p3,X3[i3])↑2 in the denominator.

Here, P3(p3,X3[i3]) is the orthogonal polynomial of degree p3 calculated on the basis of the OBS3 values of the third variable. The array of function values is written as:

Z[1:OBS1,1:OBS2,1:OBS3]

and the summation is extended over all these values. The coefficient arrays are:

a, COEF[0:DEG1,0:DEG2,0:DEG3]

9.1. The Procedure POLY23. This procedure calculates the three-dimensional coefficient array, COEF, on the basis of the function values of the three variables, X1, X2, and X3. POLY23 is very similar to POLY22 and has the declaration:

```
procedure POLY23(OBS1, OBS2, OBS3, DEG1, DEG2, DEG3,
DEGLIM, X1, X2, X3, Z, COEF);
value OBS1, OBS2, OBS3, DEG1, DEG2, DEG3, DEGLIM;
integer OBS1, OBS2, OBS3, DEG1, DEG2, DEG3, DEGLIM;
array X1, X2, X3, Z, COEF;
begin
   integer h1, h2, h3, i1, i2, i3, p1, p2, p3;
   real sum1, sum2;
   array ORPOL1[-2:DEG1,1:OBS1], ORPOL2[-2:DEG2,1:OBS2],
   ORPOL3[-2:DEG3,1:OBS3], POLC1[-1:DEG1,-1:DEG1],
   POLC2[-1:DEG2,-1:DEG2], POLC3[-1:DEG3,-1:DEG3],
   SSQPOL1[0:DEG1], SSQPOL2[0:DEG2], SSQPOL3[0:DEG3],
   a[0:DEG1,0:DEG2,0:DEG3];
   ORPOLGEN(OBS1, DEG1, X1, ORPOL1, POLC1, SSQPOL1);
   ORPOLGEN(OBS2, DEG2, X2, ORPOL2, POLC2, SSQPOL2);
   ORPOLGEN(OBS3, DEG3, X3, ORPOL3, POLC3, SSQPOL3);
   for p1 := 0 step 1 until DEG1 do
   for p2 := 0 step 1 until DEG2 do
   for p3 := 0 step 1 until DEG3 do
   begin
      sum2 := 0;
      if p1 + p2 + p3 ≤ DEGLIM then
      begin
```

```
            for i3 := 1 step 1 until OBS3 do
            for i2 := 1 step 1 until OBS2 do
            begin
               sum1 := 0;
               for i1 := 1 step 1 until OBS1 do
               sum1 := sum1 + Z[i1,i2,i3]×ORPOL1[p1,i1];
               sum2 := sum2 + sum1×ORPOL2[p2,i2]×ORPOL3[p3,i3];
            end for i3 and i2;
         end if below DEGLIM;
         a[p1,p2,p3] := sum2/SSQPOL1[p1]/SSQPOL2[p2]/SSQPOL3[p3];
      end for p1, p2, p3;
      for h1 := 0 step 1 until DEG1 do
      for h2 := 0 step 1 until DEG2 do
      for h3 := 0 step 1 until DEG3 do
      begin
         sum2 := 0;
         if h1 + h2 + h3 ≤ DEGLIM then
         begin
            for p1 := h1 step 1 until DEG1 do
            for p2 := h2 step 1 until DEG2 do
            for p3 := h3 step 1 until DEG3 do
            begin
               if p1 + p2 + p3 ≤ DEGLIM then
               sum2 := sum2 +
               a[p1,p2,p3]×POLC1[p1,h1]×POLC2[p2,h2]×POLC3[p3,h3];
            end for p1, p2, and p3;
         end if h1 + h2 + h3 ≤ DEGLIM;
         COEF[h1,h2,h3] := sum2;
      end for h1, h2, and h3;
   end POLY23;
```

We have now three calls of ORPOLGEN and all the double for-statements in POLY22 have been extended to be run through three times.

Testing of POLY23 on the data in Table 4 has been made with program d-376 shown on the next pages.

Program d-376. Test of POLY23

```
begin
   integer OBS1, OBS2, OBS3, DEG1, DEG2, DEG3, DEGLIM;
   copy ORPOLGEN<
   copy POLY23<
   copy POLYVAL3<
   select(17);
   writetext({<
Read input to d-376:});
   lyn;
   select(8);
   writetext({<
Output d-376:
});
   OBS1 := read integer;
   OBS2 := read integer;
   OBS3 := read integer;
   DEG1 := read integer;
   DEG2 := read integer;
   DEG3 := read integer;
   DEGLIM := read integer;
   begin comment inner block;
      integer i, j, k;
      real ERROR, y;
      array X1, E[1:OBS1], X2[1:OBS2], X3[1:OBS3],
      Z[1:OBS1, 1:OBS2, 1:OBS3], COEF[0:DEG1, 0:DEG2, 0:DEG3];
      for i := 1 step 1 until OBS1 do
      X1[i] := read real;
      for i := 1 step 1 until OBS2 do
      X2[i] := read real;
      for i := 1 step 1 until OBS3 do
      X3[i] := read real;
      for k := 1 step 1 until OBS3 do
      for j := 1 step 1 until OBS2 do
      for i := 1 step 1 until OBS1 do
      Z[i,j,k] := read real;
      POLY23(OBS1, OBS2, OBS3, DEG1, DEG2, DEG3, DEGLIM,
      X1, X2, X3, Z, COEF);
```

```
ERROR := 0;
writecr;
for k := 1 step 1 until OBS3 do
begin
   writecr;
   writetext({<X3  });
   write({-dd.dd}, X3[k]);
   writecr;
   writecr;
   for i := 1 step 1 until OBS1 do
   write({-dddddddd}, X1[i]);
   writecr;
   for j := 1 step 1 until OBS2 do
   begin
      writecr;
      write({dddd}, X2[j]);
      for i := 1 step 1 until OBS1 do
      write({-ddd.dddd}, Z[i,j,k]);
      writecr;
      writetext({<     });
      for i := 1 step 1 until OBS1 do
      begin
         y := POLYVAL3(DEG1, DEG2, DEG3,
         X1[i], X2[j], X3[k], COEF);
         write({-ddd.dddd}, y);
         E[i] := y - Z[i,j,k];
         ERROR := ERROR + E[i]↑2;
      end for i;
      writecr;
      writetext({<     });
      for i := 1 step 1 until OBS1 do
      write({-ddd.dddd}, E[i]);
      writecr;
   end for j;
end for k;
writecr;
writetext({<Mean error:});
write({-ddddd.ddddddd}, sqrt(ERROR/(OBS1×OBS2×OBS3-3)));
```

```
      writecr;
      writetext({<
Coefficients, COEF[i,j,k]:

});
      for k := 0 step 1 until DEG3 do
      begin
         writecr;
         writetext({<k:});
         write({dd}, k);
         writecr;
         writetext({<j:});
         for i := 0 step 1 until DEG1 do
         begin
            writetext({<       i=});
            write({d}, i);
            writetext({<       });
         end for i;
         writecr;
         for j := 0 step 1 until DEG2 do
         begin
            writecr;
            write({dd}, j);
            for i := 0 step 1 until DEG1 do
            write({  -d.dddddddd10-dd}, COEF[i,j,k]);
         end for j;
         writecr;
      end for k;
      writecr;
   end inner block;
end;
```

The program gave the following output:

Output d-376:

```
X3     0.00

       200       220       240

400   38.8210   40.9274   42.9013
      38.8433   40.8834   42.9236
       0.0223   -0.0440    0.0223

408   36.8153   38.8940   40.8475
      36.8356   38.8516   40.8676
       0.0203   -0.0424    0.0201

416   34.8732   36.9184   38.8457
      34.8934   36.8797   38.8661
       0.0202   -0.0387    0.0204

424   32.9982   35.0047   36.9007
      33.0164   34.9677   36.9189
       0.0182   -0.0370    0.0182

X3    10.00

       200       220       240

400   31.6245   33.3405   34.9477
      31.6428   33.3044   34.9660
       0.0183   -0.0361    0.0183

408   29.9900   31.6841   33.2752
      30.0067   31.6492   33.2918
       0.0167   -0.0349    0.0166
```

```
416  28.4074  30.0746  31.6449
     28.4240  30.0428  31.6616
      0.0166  -0.0318   0.0167

424  26.8795  28.5155  30.0608
     26.8945  28.4851  30.0757
      0.0150  -0.0304   0.0149

Mean error:     0.0280916

Coefficients, COEF[i,j,k]:


k: 0
j:       i=0                 i=1

 0    2.59849150 10 2   -1.86571907 10-1
 1   -9.78806179 10-1    1.57692078 10-3
 2    9.38217185 10-4   -2.13867054 10-6

k: 1
j:       i=0                 i=1

 0   -4.66532244         2.73890510 10-3
 1    1.74736044 10-2   -2.60318059 10-5
 2   -1.66600943 10-5    3.61315906 10-8
```

The values of the generated polynomial are calculated by the procedure POLYVAL3 explained in section 11.2. The printing of the three-dimensional tables of Z and COEF requires much programming, but apart from this the contents of d-376 should be fairly obvious.

10. FUNCTIONS OF MANY VARIABLES

The extension of the previously discussed procedures POLY22 and POLY-23 to cover the general case of VAR independent variables where VAR is a formal parameter presents no new mathematical problems. All we have to do is to perform the analogous changes as when POLY22 was extended from VAR = 2 to VAR = 3 in POLY23. The arrays Z, a, and COEF must now have VAR dimensions and the equation (8.3) for calculation of a must contain the proper product of all orthogonal polynomials in the numerator and all the squares in the denominator. Similarly, all the for-statements must now have VAR controlling variables.

There is, however, a practical difficulty in the fact that neither ALGOL nor FORTRAN permits the notation of VAR-dimensional arrays where VAR is a variable or a parameter, not a known integer. Similarly, a VAR-dimensional for-statement or DO-loop cannot be written in ALGOL or FORTRAN.

This deficiency of the two languages makes it necessary to transform all VAR-dimensional arrays into one-dimensional arrays and to rewrite the set of VAR for-statements as a single for-statement which internally displays the VAR controlling variables in each step.

10.1. The Procedure DISP.

This procedure can be used for transformation between a VAR-dimensional array and a one-dimensional array and for the simulation of VAR-dimensional for-statements. We can illustrate the use of DISP by referring to Table 4 on page 94. This table contains the three-dimensional array:

Z[1:3,1:4,1:2]

containing a total of $3\times4\times2 = 24$ elements. We wish to map this array into a one-dimensional array:

Z1[1:24]

This mapping can be done in different ways, depending on whether the counting starts in the first subscript or the last subscript. We have here selected to start in the first subscript and this gives the correspondence:

Z1[1]	= Z[1,1,1]	COEF1[0]	= COEF[0,0,0]
Z1[2]	= Z[2,1,1]	COEF1[1]	= COEF[1,0,0]
Z1[3]	= Z[3,1,1]	COEF1[2]	= COEF[0,1,0]
Z1[4]	= Z[1,2,1]	COEF1[3]	= COEF[1,1,0]
Z1[5]	= Z[2,2,1]	COEF1[4]	= COEF[0,2,0]
Z1[6]	= Z[3,2,1]	COEF1[5]	= COEF[1,2,0]
Z1[7]	= Z[1,3,1]	COEF1[6]	= COEF[0,0,1]
Z1[8]	= Z[2,3,1]	COEF1[7]	= COEF[1,0,1]
Z1[9]	= Z[3,3,1]	COEF1[8]	= COEF[0,1,1]
Z1[10]	= Z[1,4,1]	COEF1[9]	= COEF[1,1,1]
Z1[11]	= Z[2,4,1]	COEF1[10]	= COEF[0,2,1]
Z1[12]	= Z[3,4,1]	COEF1[11]	= COEF[1,2,1]
Z1[13]	= Z[1,1,2]		
Z1[14]	= Z[2,1,2]		
Z1[15]	= Z[3,1,2]		
Z1[16]	= Z[1,2,2]		
Z1[17]	= Z[2,2,2]		
Z1[18]	= Z[3,2,2]		
Z1[19]	= Z[1,3,2]		
Z1[20]	= Z[2,3,2]		
Z1[21]	= Z[3,3,2]		
Z1[22]	= Z[1,4,2]		
Z1[23]	= Z[2,4,2]		
Z1[24]	= Z[3,4,2]		

The table also shows the correspondence between a three-dimensional coefficient array:

```
COEF[0:1,0:2,0:1]
```

and the one-dimensional array:

```
COEF1[0:11]
```

DISP solves the problem of finding the subscripts in Z which correspond to a given value of the single subscript in Z1 and similarly for COEF and COEF1. To do this, we must know how many possible values there are for each subscript. This is written as the radix-array r[1:3]:

Z-Z1	COEF-COEF1
r[1] = 3	r[1] = 2
r[2] = 4	r[2] = 3
r[3] = 2	r[3] = 2

If we wish to find the subscripts of Z corresponding to Z1[15] and to assign the subscripts to an array, w[1:3], we start by the number 15 and subtract 1 giving 14. This is preliminarily assigned to w[1]:

```
w[1] := 14;
```

We then preliminarily calculate w[2] as $14\underline{:}3$, the first radix:

$$w[2] := 14\underline{:}3 = 4$$

The final value of w[1] is calculated as 14 mod 3 +1 giving:

```
w[1] := 3
```

w[3] is set to $4\underline{:}3 = 1$, w[2] is reset to 4 mod 3 +1 = 2, and we at last add 1 to the last subscript giving w[3] =2. In other words, the original subscript is divided by r[1], r[2], etc. and replaced by the remainders. Mathematically speaking, this is nothing but the transformation from one number system into another, and where a mixed radix is permitted, such as the transformation of pence into pence, shillings, and pounds.

DISP has the six formal parameters:

var	integer	This is the same as VAR, the number of dimensions in the multi-dimensionsional array.
prod	integer	This is the subscript of the given element in the one-dimensional array.
w	integer array	[1:var]. The subscripts are generated in this array.
r	integer array	[1:var]. The radix array explained above.
c1,c2	integer	These corrections are used to correct for the fact that the subscripts may start in 0 or 1.

In transformation of Z1 into Z we must use c1=0 and c2=1 and for the transformation of COEF1 into COEF we must have c1=1 and c2=0.

The declaration of DISP is:

```
procedure DISP(var, prod, w, r, c1, c2);
value var, prod, c1, c2;
integer var, prod, c1, c2;
integer array w, r;
begin
   integer j, k;
   w[1] := prod - 1;
   for j := 2 step 1 until var do
   begin
      k := c1 + r[j-1];
      w[j] := w[j-1]:k;
      w[j-1] := w[j-1] - w[j]×k + c2;
   end for j;
   w[var] := w[var] + c2;
end DISP;
```

It is also convenient to have a small procedure which can calculate the total number of elements in a VAR-dimensional array. This can be done by the procedure PROD:

```
integer procedure PROD(VAR, LIST, k);
value VAR, k;
integer VAR, k;
integer array LIST;
begin
   integer i, factor;
   factor := 1;
   for i := 1 step 1 until VAR do
   factor := factor×(LIST[i] + k);
   PROD := FACTOR;
end PROD;
```

The procedure has the formal parameters:

VAR	integer	The number of dimensions in the array.
LIST	integer array	[1:VAR]. A list of the upper subscript bounds of the array.
k	integer	Correction for the lower subscript bound.

The value of k must be inserted as 0 for the array Z and 1 for COEF.

10.2. The Procedure POLY24. This procedure has the declaration:

```
procedure POLY24(VAR, OBSLIST, OBSMAX, DEGLIST,
DEGMAX, DEGLIM, X, Z, COEF);
value VAR, OBSMAX, DEGMAX, DEGLIM;
integer VAR, OBSMAX, DEGMAX, DEGLIM;
integer array OBSLIST, DEGLIST;
array X, Z, COEF;
begin
   integer NPROD, DEGPROD, i, j, k, deg, obs, out, dprod,
   n1, psum, in, nprod, deg1, h1, dprod2, p1;
   real SUM, FACTOR;
   boolean range;
   integer array hlist, ilist, plist[1:VAR];
   array ORPOL[0:DEGMAX,1:VAR×OBSMAX],
   POLC[0:DEGMAX,0:(VAR×(DEGMAX+1)-1)],
   SSQPOL[0:DEGMAX,1:VAR], a[0:(PROD(VAR,DEGLIST,1)-1)];
   DEGPROD := PROD(VAR, DEGLIST, 1);
   NPROD := PROD(VAR, OBSLIST, 0);
   n1 := OBSLIST[1];
   for i := 1 step 1 until VAR do
   begin
      deg := DEGLIST[i];
      obs := OBSLIST[i];
      begin comment inner block;
         array XV[1:obs], ORPOLV[-2:deg,1:obs],
         POLCV[-1:deg,-1:deg], SSQPOLV[0:deg];
         for j := 1 step 1 until obs do
         XV[j] := X[j+OBSMAX×(i-1)];
         ORPOLGEN(obs, deg, XV, ORPOLV, POLCV, SSQPOLV);
      for j := 0 step 1 until deg do
      begin
         for k := 1 step 1 until obs do
         ORPOL[j,k+OBSMAX×(i-1)] := ORPOLV[j,k];
         for k := 0 step 1 until deg do
         POLC[j,k+(DEGMAX+1)×(i-1)] := POLCV[j,k];
         SSQPOL[j,i] := SSQPOLV[j];
      end for j;
   end inner block;
end for i;
deg := DEGLIST[1];
deg1 := deg+1;
out := 0;
for dprod := 1 step 1 until DEGPROD do
begin
   SUM := 0;
   DISP(VAR, dprod, plist, DEGLIST, 1, 0);
   psum := 0;
   for i := 1 step 1 until VAR do
   psum := psum + plist[i];
   in := 1;
   if psum ≤ DEGLIM then
   for nprod := 1 step n1 until NPROD do
   begin
      DISP(VAR, nprod, ilist, OBSLIST, 0, 1);
      FACTOR := 1;
      for i := 2 step 1 until VAR do
      FACTOR := FACTOR×ORPOL[plist[i], ilist[i]+OBSMAX×(i-1)];
      for i := 1 step 1 until n1 do
      begin
         SUM := SUM + Z[in]×ORPOL[plist[1], i]×FACTOR;
         in := in + 1;
      end for i;
   end for nprod;
   FACTOR := 1;
   for i := 1 step 1 until VAR do
   FACTOR := FACTOR×SSQPOL[plist[i],i];
   a[out] := SUM/FACTOR;
```

```
      out := out + 1;
    end for dprod;
    out := 0;
    for dprod := 1 step deg1 until DEGPROD do
    begin
      DISP(VAR, dprod, hlist, DEGLIST, 1, 0);
      for h1 := 0 step 1 until deg do
      begin
        SUM := 0;
        in := 0;
        for dprod2 := 1 step deg1 until DEGPROD do
        begin
          DISP(VAR, dprod2, plist, DEGLIST, 1, 0);
          FACTOR := 1;
          range := true;
          for i := 2 step 1 until VAR do
          if plist[i] < hlist[i] then range := false;
          if range then
          for i := 2 step 1 until VAR do
          FACTOR := FACTOR×
          POLC[plist[i], hlist[i]+(DEGMAX+1)×(i-1)];
          for p1 := 0 step 1 until deg do
          begin
            if p1 ≥ h1 ∧ range then
            SUM := SUM + FACTOR×a[in]×POLC[p1,h1];
            in := in + 1;
          end for p1
        end for dprod2;
        COEF[out] := SUM;
        out := out + 1;
      end for h1;
    end for dprod;
  end POLY24;
```

The procedure has the following formal parameters:

VAR	<u>integer</u>	The number of independent variables.
OBSLIST	<u>integer</u>	<u>array</u> [1:VAR]. A list of the number of values of each independent variable.
OBSMAX	<u>integer</u>	The largest element in OBSLIST.
DEGLIST	<u>integer</u>	<u>array</u> [1:VAR]. A list of the required polynomial degrees for each variable.
DEGMAX	<u>integer</u>	The largest element in DEGLIST.
DEGLIM	<u>integer</u>	The maximum sum of exponents in the generated polynomial.
X	<u>array</u>	[1:VAR×OBSMAX]. The values of the independent variables are packed in this array as explained below.
Z	<u>array</u>	[1:PROD(VAR,OBSLIST,0)]. This is the array of function values.
COEF	<u>array</u>	[0:PROD(VAR,DEGLIST,1)-1]. This is the array of polynomial coefficients generated by the procedure.

We illustrate the use of POLY24 on the data in Table 4 on page 94. Here we have:

VAR = 3

OBSLIST[1] = 3
OBSLIST[2] = 4
OBSLIST[3] = 2

From this we find that OBSMAX = 4, and we can then pack the three X1-values (200, 220, and 240), the four X2-values (400, 408, 416, and 424), and the two X3-values (0 and 10) into a single array with 12 elements:

X[1]=200	X[2] =220	X[3] =240	X[4] Not used
X[5]=400	X[6] =408	X[7] =416	X[8] = 424
X[9]= 0	X[10]= 10	X[11] Not used	X[12] Not used

This gives a fairly dense packing, provided that no value of OBSLIST is much larger than the others.

For the polynomial degrees we select the values:

DEGLIST[1] = 1
DEGLIST[2] = 2
DEGLIST[3] = 1

This gives DEGMAX = 2. For DEGLIM we select 4. The two arrays Z and COEF have the sizes shown for the one-dimensional arrays Z1 and COEF1 on page 103.

The procedure POLY24 internally reserves storage space for four different arrays:

```
ORPOL[0:DEGMAX,1:VAR×OBSMAX]
POLC[0:DEGMAX,0:(VAR×(DEGMAX+1)-1)]
SSQPOL[0:DEGMAX,1:VAR]
a[0:PROD(VAR,DEGLIST,1)-1]
```

ORPOL contains all the orthogonal polynomials ordered for the first variable in columns 1 to OBSMAX, for the second variable in columns OBSMAX+1 to 2×OBSMAX, etc., just as the arrangement of the X-array. In a similar way, POLC contains the polynomial coefficients for the first variable in columns 0 to DEGMAX, for the second variable in columns DEGMAX+1 to 2×DEGMAX+1, etc. The array SSQPOL contains the sum of the squares of the orthogonal polynomials for variable no. i as column no.i. The a-array has the same size as the COEF1-array.

The procedure starts with a for-statement counting i from 1 to VAR. For each value of i a block is entered in which a call of ORPOLGEN is made. In this block arrays of the proper size and dimensions are extracted from the condensed X-array, and the generated arrays are read back into the condensed arrays.

The second part of the procedure consists of the for-statement:

<u>for</u> dprod := 1 <u>step</u> 1 <u>until</u> DEGPROD <u>do</u>

which simulates the VAR-tuple for-statement in p1, p2, p3, etc., calculating a[p1,p2,p3,etc.] by summation of the product:

Z[in]×ORPOL1[p1,i1]×ORPOL2[p2,i2]×.....

The variation of p1, p2, etc. is obtained by a call of DISP:

```
DISP(VAR, dprod, plist, DEGLIST, 1, 0);
```

displaying the p-values in plist[1:VAR]. The variation of i1, i2, etc. is controlled by an inner for-statement:

```
for nprod := 1 step n1 until NPROD do
```

and a corresponding call of DISP (n1 is OBSLIST[1]):

```
DISP(VAR, nprod, ilist, OBSLIST, 0, 1);
```

displaying the i-values in ilist[1:VAR]. In order to economize the calculation time for the long product with Z, the variation of i1 has been separated as a special for-statement. The scanning of the Z-elements and the a-elements is controlled by two counters, in and out.

The third part of POLY24 has the for-statement:

```
for dprod := 1 step deg1 until DEGPROD do
```

which simulates the VAR-tuple for-statement in h1, h2, etc. For each value of dprod we have the call of DISP:

```
DISP(VAR, dprod, hlist, DEGLIST, 1, 0);
```

displaying the h-values in hlist. The variation of h1 is separated as a special for-statement, so that dprod is counted in steps of deg1 = DEGLIST[1]+1. A similar, inner for-statement counts dprod2 from 1 to DEGPROD, the corresponding p-values being displayed in plist. The kernel of the inner for-statement performs the summation of:

```
a[in]×POLC1[p1,h1]×POLC2[p2,h2]×.....
```

and stores the sum in COEF[out]. The counters, in and out, scan the two arrays: a and COEF.

The program d-377 performs the same calculation as the program d-376 on page 97, but uses POLY24 instead of POLY23. The program is:

```
Program d-377.  Test of POLY24.
begin
   integer i, j, k, VAR, DEGLIM, DEGMAX, OBSMAX, OBS, OBS1;
   real ERROR, y;
   integer array ilist, OBSLIST, DEGLIST[1:5];
   copy DISP<
   copy ORPOLGEN<
   copy POLY24<
   copy POLYVALN<
   copy PROD<
   select(17);
   writetext(⨭<
Read input to d-377:⨮);
   lyn;
   select(8);
   writetext(⨭<
Output d-377:⨮);
   VAR := read integer;
   DEGMAX := OBSMAX := 0;
   for i := 1 step 1 until VAR do
   begin
      OBSLIST[i] := read integer;
      if OBSLIST[i] > OBSMAX then OBSMAX := OBSLIST[i];
      DEGLIST[i] := read integer;
      if DEGLIST[i] > DEGMAX then DEGMAX := DEGLIST[i];
   end for i;
   DEGLIM := read integer;
   OBS1 := OBSLIST[1];
   begin comment inner block;
      array X[1:VAR×OBSMAX], Z[1:PROD(VAR, OBSLIST, 0)],
      COEF[0:(PROD(VAR, DEGLIST, 1)-1)], XACT[1:VAR], E[1:OBS1];
      for i := 1 step 1 until VAR do
      for j := 1 step 1 until OBSLIST[i] do
      X[j+OBSMAX×(i-1)] := read real;
      OBS := PROD(VAR, OBSLIST, 0);
```

```
for i := 1 step 1 until OBS do
Z[i] := read real;
POLY24(VAR, OBSLIST, OBSMAX, DEGLIST,
DEGMAX, DEGLIM, X, Z, COEF);
ERROR := 0;
writecr;
for i := 1 step OBS1 until OBS do
begin
   DISP(VAR, i, ilist, OBSLIST, 0, 1);
   for k := 2 step 1 until VAR do
   XACT[k] := X[ilist[k] + OBSMAX×(k-1)];
   writecr;
   if VAR > 2 ∧ ilist[2] = 1 then
   begin
      for k := 3 step 1 until VAR do
      begin
         writetext({<X});
         write({d}, k);
         write({  -dd.dd}, XACT[k]);
         writecr;
      end for k;
      writecr;
   end if VAR > 2 ∧ ilist[2] = 1;
   if VAR > 1 ∧ ilist[2] = 1 then
   begin
      for k := 1 step 1 until OBS1 do
      write({-dddddddd}, X[k]);
      writecr;
      writecr;
   end if VAR > 1 ∧ ilist[2] = 1;
   if VAR > 1 then write({dddd}, XACT[2])
   else writetext({<    });
   for k := 1 step 1 until OBS1 do
   write({-ddd.dddd}, Z[i+k-1]);
   writecr;
   writetext({<    });
```

```
         for k := 1 step 1 until OBS1 do
         begin
            XACT[1] := X[k];
            y := POLYVALN(VAR, DEGLIST, XACT, COEF);
            write({-ddd.dddd}, y);
            E[k] := y-Z[i+k-1];
            ERROR := ERROR + E[k]↑2;
         end for k;
         writecr;
         writetext({<    });
         for k := 1 step 1 until OBS1 do
         write({-ddd.dddd}, E[k]);
         writecr;
      end for i;
      writecr;
      writetext({<Mean error:});
      write({-ddddd.ddddddd}, sqrt(ERROR/(OBS-VAR)));
      writetext({<
Coefficients:
});
      writecr;
      j := PROD(VAR, DEGLIST, 1)-1;
      for i := 0 step 1 until j do
      begin
         writecr;
         writetext({<COEF[});
         write({dd}, i);
         writetext({<] = COEF[});
         DISP(VAR, i + 1, ilist, DEGLIST, 1, 0);
         for k := 1 step 1 until VAR do
         begin
            write({d}, ilist[k]);
            writetext(if k = VAR then {<]} else {<, });
         end for k;
         write({  -d.dddddddd₁₀-dd}, COEF[i]);
      end for i;
      writecr;
   end inner block;
end;
```

The fact that VAR is unknown when the program starts makes it necessary to have an inner block which is entered when VAR has been read and which contains the necessary arrays. For simplicity, OBSLIST and DEGLIST have been declared in the outermost block with a maximum size of VAR = 5, but this can be changed, if required.

The program gave the following output:

```
Output d-377:

X3      0.00

        200       220       240

 400    38.8210   40.9274   42.9013
        38.8433   40.8834   42.9236
         0.0223   -0.0440    0.0223

 408    36.8153   38.8940   40.8475
        36.8356   38.8516   40.8676
         0.0203   -0.0424    0.0201

 416    34.8732   36.9184   38.8457
        34.8934   36.8797   38.8661
         0.0202   -0.0387    0.0204

 424    32.9982   35.0047   36.9007
        33.0164   34.9677   36.9189
         0.0182   -0.0370    0.0182

X3     10.00

        200       220       240

 400    31.6245   33.3405   34.9477
        31.6428   33.3044   34.9660
         0.0183   -0.0361    0.0183
```

```
408  29.9900  31.6841  33.2752
     30.0067  31.6492  33.2918
      0.0167  -0.0349   0.0166

416  28.4074  30.0746  31.6449
     28.4240  30.0428  31.6616
      0.0166  -0.0318   0.0167

424  26.8795  28.5155  30.0608
     26.8945  28.4851  30.0757
      0.0150  -0.0304   0.0149

Mean error:      0.0280913
Coefficients:


COEF[ 0] = COEF[0, 0, 0]   2.59847985 ₁₀ 2
COEF[ 1] = COEF[1, 0, 0]  -1.86566859 ₁₀-1
COEF[ 2] = COEF[0, 1, 0]  -9.78800505 ₁₀-1
COEF[ 3] = COEF[1, 1, 0]   1.57689625 ₁₀-3
COEF[ 4] = COEF[0, 2, 0]   9.38210287 ₁₀-4
COEF[ 5] = COEF[1, 2, 0]  -2.13864074 ₁₀-6
COEF[ 6] = COEF[0, 0, 1]  -4.66511810
COEF[ 7] = COEF[1, 0, 1]   2.73791276 ₁₀-3
COEF[ 8] = COEF[0, 1, 1]   1.74726039 ₁₀-2
COEF[ 9] = COEF[1, 1, 1]  -2.60269416 ₁₀-5
COEF[10] = COEF[0, 2, 1]  -1.66588699 ₁₀-5
COEF[11] = COEF[1, 2, 1]   3.61256300 ₁₀-8
```

For simplicity, the program prints the subscripts of COEF in two different ways: As a one-dimensional array and a three-dimensional array. Evaluation of the calculated polynomial is made by the procedure POLYVALN described in section 11.3.

11. POLYNOMIAL EVALUATION

The evaluation of a polynomial of degree DEG of a single variable is so simple that a special procedure is normally not required:

```
P := 0;
for j := DEG step -1 until 0 do
P := P×X + COEF[j];
```

When the number of variables is higher than 1, the calculation is more complicated, especially for VAR variables and VAR is a formal parameter. Procedures for 2, 3, and VAR variables are described in the following.

The evaluation procedures can be arranged to yield either the normal polynomial value or a value corresponding to differentiation or integration with respect to one or more of the variables. However, this special possibility is not considered here, because it makes the procedure very slow and complicated. If differentiations or integrations are required, it is recommended to use procedures which transform the coefficient lists, and then apply the normal evaluation procedure.

11.1. The Procedure POLYVAL2. This procedure evaluates a polynomial in two variables. The declaration is:

```
real procedure POLYVAL2(DEG1, DEG2, X1, X2, COEF);
value DEG1, DEG2, X1, X2;
integer DEG1, DEG2;
real X1, X2;
array COEF;
begin
   integer i1, i2;
   real R, S;
   S := 0;
   for i1 := DEG1 step -1 until 0 do
   begin
      R := 0;
      for i2 := DEG2 step -1 until 0 do
      R := R×X2 + COEF[i1,i2];
```

```
       S := SxX1 + R;
    end for i1;
    POLYVAL2 := S;
 end POLYVAL2;
```

The formal parameters are:

DEG1	integer	The highest exponent of the first variable.
DEG2	integer	The highest exponent of the second variable.
X1	real	The actual value of the first variable.
X2	real	The actual value of the second variable.
COEF	array	[0:DEG1,0:DEG2]. The polynomial coefficient array, e.g. as generated by POLY22.

The procedure consists of a double for-statement performing the multiplications and the additions. The maximum sum of degrees, DEGLIM, is not used as a formal parameter, so that the elements in COEF beyond this limit must be zero. This is properly handled by POLY22.

11.2. The Procedure POLYVAL3. This is completely similar to the procedure POLYVAL2, but for 3 dimensions. The declaration is:

```
real procedure POLYVAL3(DEG1, DEG2, DEG3, X1, X2, X3, COEF);
value DEG1, DEG2, DEG3, X1, X2, X3;
integer DEG1, DEG2, DEG3;
real X1, X2, X3;
array COEF;
begin
   integer i1, i2, i3;
   real R, S, T;
   T := 0;
   for i1 := DEG1 step -1 until 0 do
   begin
      S := 0;
      for i2 := DEG2 step -1 until 0 do
      begin
         R := 0;
         for i3 := DEG3 step -1 until 0 do
         R := RxX3 + COEF[i1,i2,i3];
```

```
            S := S×X2 + R
         end for i2;
         T := T×X1 + S;
      end for i1;
      POLYVAL3 := T;
   end POLYVAL3;
```

The formal parameters DEG3 and X3 have been added, and the procedure contains 3 for-statements inside another.

11.3. The Procedure POLYVALN. This is the general type procedure which evaluates a polynomial of VAR variables. The declaration is:

```
real procedure POLYVALN(VAR, DEGLIST, X, COEF);
value VAR;
integer VAR;
integer array DEGLIST;
array X, COEF;
begin
   integer cell, cycle, deg1, i;
   real SUMROW, X1;
   integer array plist[1:VAR];
   array SUM[1:VAR];
   cell := 1;
   for i := 1 step 1 until VAR do
   begin
      SUM[i] := 0;
      plist[i] := DEGLIST[i];
      cell := cell×(1+plist[i]);
   end for i;
   deg1 := plist[1];
   X1 := X[1];
   for cycle := cell:(deg1+1) step -1 until 1 do
   begin
      SUMROW := 0;
      for i := deg1 step -1 until 0 do
      begin
         cell := cell -1;
         SUMROW := SUMROW×X1 + COEF[cell];
      end for i;
```

```
        if VAR > 1 then
        SUM[2] := SUMROW := SUM[2]×X[2] + SUMROW
        else go to EX;
        plist[2] := plist[2] -1;
        for i := 2 step 1 until VAR do
        begin
           if plist[i] < 0 then
           begin
              plist[i] := DEGLIST[i];
              if i < VAR then
              begin
                 plist[i+1] := plist[i+1] - 1;
                 SUM[i+1] := SUMROW := SUM[i+1]×X[i+1]+SUMROW;
                 SUM[i] := 0;
              end if i < VAR;
           end if plist[i] < 0;
        end for i;
     end for cycle;
     EX:
     POLYVALN := SUMROW;
  end POLYVALN;
```

The formal parameters in POLYVALN are:

VAR	integer	The number of independent variables.
DEGLIST	integer array	[1:VAR]. This is the list of polynomial degrees as in POLY24.
X	array	[1:VAR]. The list of actual values of the independent variables.
COEF	array	[0:PROD(VAR,DEGLIST,1)-1]. The one-dimensional array of polynomial coefficients as in POLY24.

The procedure contains a counter, cell, used in the scanning of COEF and a basic for-statement:

```
for i := deg1 step -1 until 0 do
begin
   cell := cell - 1;
   SUMROW := SUMROW×X1 + COEF[cell];
end for i;
```

The simple variable, deg1, is equal to DEGLIST[1], and X1 is equal to X[1]. If VAR = 1 no further calculation is required, but if VAR > 1, the nested polynomial evaluation is continued by multiplying the previous sum by X[2] and adding the new sum. The old sums are stored in the local array:

SUM[1:VAR]

which is initially set to zero.

Control of the multiplications by the other variables is made by another local array:

plist[1:VAR]

of type integer, and in which the elements are first set equal to the elements in DEGLIST. Whenever a multiplication by X[2] has been made, plist[2] is counted 1 downwards. A check is then made on all the items in plist (except plist[1] which is not used). If an element, plist[i], becomes negative, it is reset to DEGLIST[i], and 1 is subtracted from the next element, plist[i+1], except when i = VAR. At the same time, the contents of the sum cell, SUM[i+1], is multiplied by X[i+1], and SUMROW is added to it. SUM[i] is reset to zero.

This calculation method is the generalized Horner scheme. As in the procedures POLYVAL2 and POLYVAL3 we assume that any coefficients that might have been cut off because of the DEGLIM parameter are actually present as zero.

12. SPECIAL APPROXIMATION POLYNOMIALS

The following discussion is mainly concerned with approximation of functions of a single variable.

In some cases we require that the polynomial which reproduces a given set of table values must satisfy special conditions. We may wish that the polynomial passes exactly through one or more points without any error. Similarly, the derivatives of the polynomial may have to assume known values at given points.

As an example we take a function of a single variable:

(12.1) $Y = F(X)$

which is given as a table. We wish that the polynomial passes exactly through the two points:

Point 1: (XA, YA)
Point 2: (XB, YB)

We must then have:

(12.2) $Y = YA$ for $X = XA$
(12.3) $Y = YB$ for $X = XB$

In order to solve this, we write the required polynomial, Y, as the sum of two terms:

(12.4) $Y = MODPOL + NODE \times MAINPOL$

Here, MODPOL, NODE, and MAINPOL are all polynomials. We then have to arrange it so that the node polynomial, NODE, is exactly zero at the two special points, XA and XB, and that the modifier polynomial, MODPOL, has the required values: YA at XA and YB at XB. The conditions are then satisfied. It then remains to adjust the polynomial, MAINPOL, to fit the given table values as accurately as possible.

As NODE must be zero at X = XA and X = XB, we can write it as:

(12.5) $NODE = (X - XA) \times (X - XB)$

As MODPOL must pass exactly through the two points (XA,YA) and (XB,YB), it must be a straight line, and we can write it as:

(12.6) $Y = C0 + C1 \times X$

We can then find C0 and C1 from the two equations:

(12.7) $YA = C0 + C1 \times XA$

(12.8) $YB = C0 + C1 \times XB$

to:

(12.9) $C0 = (YB \times XA - YA \times XB)/(XA - XB)$

(12.10) $C1 = (YA - YB)/(XA - XB)$

Insertion of (12.9) and (12.10) into (12.6) and rearrangement gives:

(12.11) $MODPOL = YA + (YA - YB)/(XA - XB) \times (X - XB)$

We now know MODPOL and NODE and may calculate their value in each of the specified table points. If equation (12.4) is solved with respect to MAINPOL:

(12.12) $MAINPOL = (Y - MODPOL)/NODE$

we can easily see, that all we have to do is to calculate a new set of Y-values as:

(12.13) $YNEW = (Y - MODPOL)/NODE$

and then to calculate the coefficients in MAINPOL by means of the procedure POLY1 using the original X-values together with the Y-values of YNEW.

When the coefficients in MAINPOL have been found, we can either insert these coefficients into equation (12.4) together with the coefficients from NODE and MODPOL, giving a normal polynomial, or we can retain the form of equation (12.4). The latter is probably more accurate.

12.1. Calculation Example. As a slightly more complicated example we consider the following approximation problem.

The boiling point, Y, of a mixture of water and ethanol is a function of the mole fraction, X, of ethanol in the liquid mixture. Values of X and Y are shown in Table 5 below.

Mole fraction of ethanol X	Boiling point deg.C Y
0.01	97.41
0.02	95.16
0.04	91.60
0.06	89.17
0.08	87.56
0.10	86.38
0.14	84.73
0.18	83.62
0.25	82.26
0.35	81.00
0.45	80.03
0.55	79.33
0.65	78.80
0.75	78.36
0.85	78.18
0.95	78.20

Table 5

Boiling points of ethanol-water mixtures

The special requirements to the approximation polynomial are:

1. The boiling point of pure water must be reproduced exactly, i.e. for X = 0 we must have Y = 100.00.

2. The boiling point of the azeotropic mixture must also be exact. This means that for X = 0.89404 we must have Y = 78.15.

3. The boiling point curve must have a horizontal tangent at the azeotropic point:

(12.14) dY/dX = 0 for X = 0.89404

We must now use the same expression for Y as in equation (12.4) and its derivative:

(12.15) Y = MODPOL + NODE×MAINPOL
(12.16) dY/dX = dMODPOL/dX + NODE×dMAINPOL/dX + MAINPOL×dNODE/dX

At the two special points, X = 0 and 0.89404, the NODE-polynomial is zero. This gives:

(12.17) dY/dX = dMODPOL/dX + MAINPOL×dNODE/dX

But as we wish to have a special value of dY/dX at a special point, we also arrange dNODE/dX to be zero at X = 0.89404. This is obtained by writing NODE as:

(12.18) NODE = (X - XA)×(X - XB)↑2

Here, XA = 0 and XB = 0.89404. The modifying polynomial, MODPOL, must satisfy the conditions:

(12.19) MODPOL = 100 at X = XA
MODPOL = 78.15 at X = XB
dMODPOL/dX = 0 at X = XB

These conditions require a second order polynomial:

(12.20) MODPOL = C0 + C1×X + C2×X↑2
dMODPOL/dX = C1 + 2×C2×X

Insertion of the three conditions gives:

(12.21) $$\begin{aligned} 100 &= C0 \\ 78.15 &= C0 + C1\times0.89404 + C2\times0.89404\uparrow2 \\ 0 &= C1 + 2\times C2\times0.89404 \end{aligned}$$

from which we find:

(12.22) $$\text{MODPOL} = 100 - 48.8792\times X + 27.3362\times X\uparrow2$$

We now have the necessary expressions for NODE and MODPOL. The old Y-values, Y, are then replaced by the new Y-values, YNEW, calculated after eqaution (12.13). When the set of YNEW is treated on POLY1, we get the final results as shown in Table 6 below.

		No weighing		Automatic weighing		
X	Y,meas.	Y,calc.	error	Y,calc.	error	weight
0.01	97.41	97.387	-0.023	97.338	-0.072	0.00006
0.02	95.16	95.171	0.011	95.103	-0.057	0.00023
0.04	91.60	91.712	0.112	91.661	0.061	0.00085
0.06	89.17	89.249	0.079	89.247	0.077	0.00174
0.08	87.56	87.496	-0.064	87.542	-0.018	0.00281
0.10	86.38	86.240	-0.140	86.317	-0.063	0.00398
0.14	84.73	84.631	-0.099	84.708	-0.022	0.00634
0.18	83.62	83.633	0.013	83.649	0.029	0.00842
0.25	82.26	82.389	0.129	82.293	0.033	0.01075
0.35	81.00	81.014	0.014	80.945	-0.055	0.01073
0.45	80.03	80.028	-0.002	80.064	0.034	0.00787
0.55	79.33	79.310	-0.020	79.341	0.011	0.00424
0.65	78.80	78.762	-0.038	78.759	-0.041	0.00150
0.75	78.36	78.389	0.029	78.399	0.039	0.00024
0.85	78.18	78.178	-0.002	78.182	0.002	0.000003
0.95	78.20	78.201	0.001	78.189	-0.011	0.000009
	Mean:		0.070		0.046	

Table 6. Approximation of

Boiling points of ethanol-water mixtures

The table gives two columns of calculated Y-values. Both correspond to a MAINPOL of degree 6, but the first column is calculated without weights, whereas the second column of Y,calc is found with the weights shown in the last column.

The weights are here calculated as the square of the NODE polynomial giving very small weights close to the special points, where the main part of Y comes from MODPOL.

13. REFERENCES

Ascher, M. and Forsythe, G.E.: Journ. ACM, 5, 9-21 (1958).

Cadwell, J. H.: Comp. Journ. 3, 266-269 (1960).

Clenshaw, C. W.: Comp. Journ. 2, 170-173 (1959).

Clenshaw, C. W., and Hayes, J. J.: J.Inst.Maths.Applics., 1, 164-183 (1965).

Fisher, R. A.: Statistical Methods for Research Workers (1948).

Forsythe, G. E.: J. Soc.Indust.Appl.Math., 5, 74-88 (1957).

Frøberg, C.-E.: Introduction to Numerical Analysis, Addison-Wesley Publ. Co. (1966).

Kantorowitz, E.: Mathematical Definition of Ship Surfaces, Copenhagen (1967).

Kjær, J.: Calculation of Ammonia Converters on an Electronic Digital Computer (1963).

Kjær, J.: Computer Methods in Linear and Quadratic Models (1970).

Lapidus, L.: Digital Computation for Chemical Engineers, McGraw-Hill Book Co. (1962).

14. ALPHABETIC INDEX