ORTHOGONAL POLYNOMIALS

A thesis submitted in partial satisfaction of the requirements for the degree of Master of Science in Mathematics by

Donald James Lund, Jr.

May 1983
The Thesis of Donald James Lund, Jr. is approved:

William Karush

E. Muriel Wright

John W. Blattner, Chairman

California State University, Northridge
ACKNOWLEDGMENTS

I would like to express my thanks to the instructors in the mathematics department: Dr. Blattner, Dr. Wright, Dr. Karush, and Dr. Potts. Without their patience and continual support, I would never have been able to complete this paper. At times when I was having difficulties in completing this paper, the people in the mathematics department always seemed to be willing to offer encouragement and help. Finally, I would like to thank my parents, who were willing to help in so many ways.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgments</td>
<td>iii</td>
</tr>
<tr>
<td>Abstrace</td>
<td>v</td>
</tr>
<tr>
<td>Chapter</td>
<td></td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2. Polynomial Approximations</td>
<td>3</td>
</tr>
<tr>
<td>3. Orthogonal Polynomials Approximations</td>
<td>9</td>
</tr>
<tr>
<td>Gram Polynomials</td>
<td>11</td>
</tr>
<tr>
<td>Gram-Schmidt Orthogonalization Process</td>
<td>16</td>
</tr>
<tr>
<td>Forsythe Recurrence Relation</td>
<td>18</td>
</tr>
<tr>
<td>4. Errors in Least Squares Approximations</td>
<td>22</td>
</tr>
<tr>
<td>5. Applications</td>
<td>32</td>
</tr>
<tr>
<td>6. Summary</td>
<td>37</td>
</tr>
<tr>
<td>Bibliography</td>
<td>39</td>
</tr>
<tr>
<td>Appendixes</td>
<td></td>
</tr>
<tr>
<td>A. Program Gram</td>
<td>40</td>
</tr>
<tr>
<td>B. Program Schmidt</td>
<td>43</td>
</tr>
<tr>
<td>C. Program Forsythe</td>
<td>47</td>
</tr>
<tr>
<td>D. Sample Solutions</td>
<td>51</td>
</tr>
</tbody>
</table>
ABSTRACT

ORTHOGONAL POLYNOMIALS

by

Donald James Lund, Jr.

Master of Science in Mathematics

This paper discusses how to use orthogonal polynomials to find a polynomial least squares approximation of a function which is defined by experiment over a discrete set of points. After discussing the advantages of using orthogonal polynomials over the more traditional approach of using normal equations to find a polynomial least squares approximation, this paper presents three methods of deriving a set of orthogonal polynomials: the Gram polynomials, the Gram-Schmidt orthogonalization process, and the Forsythe recurrence relation. These methods are compared, and the Forsythe recurrence is shown to be the most efficient of these methods for constructing a set of orthogonal polynomials. Next, the paper discusses how to calculate the errors involved in finding a polynomial least squares approximation and how a study of these errors can be used to determine the degree of the polynomial least squares approximation which will give the best results. Finally, this paper shows how polynomial least squares approximations can be used for data smoothing and approximate differentiation.
The appendix of this paper contains three computer programs which demonstrate how each of the three methods of deriving a set of orthogonal polynomials can be used by a digital computer. Each of these programs is used to find a polynomial least squares approximation of a function defined by the same set of data, and a comparison of resulting approximations is given.
CHAPTER ONE

Introduction

Most mathematicians know that it is possible to use polynomials to approximate a given function. Methods to accomplish this task have been known for a very long time. However, until recently it has been practical to use only low degree polynomials for these approximations, and some practical applications which demanded the use of high degree polynomials were avoided. But the present availability of high speed digital computers has changed this situation. Calculations which used to take hours to perform manually can now be performed in a matter of seconds. Many applications and methods which were considered impractical can now be handled satisfactorily.

Unfortunately, the general literature on polynomial approximations has not kept up with the advance in new technology. Most textbooks on numerical analysis discuss methods using Legendre, Chebyshev, and Laguerre polynomials to approximate a continuous function over an interval. Often, however, we must deal with functions defined by experiment over a discrete set of points, and not nearly as many textbooks discuss this subject. Those books which do discuss the subject usually rely upon using tables and tabulation procedures to perform the needed calculations. Few books discuss methods to approximate these functions which can be readily adapted for use by digital computers. The paper by Forsythe [3] revolutionized the whole subject by showing how to use a digital computer to make polynomial approximations of functions whose values are only known over a discrete set
of points.

This paper will discuss methods using polynomials to approximate functions defined by experiment over a discrete set of points, and will show how these methods can be adapted for use by digital computers.
CHAPTER TWO

Polynomial Approximations

Let \( \{x_i\} \) for \( i = 1, \ldots, n \) be a sequence of distinct real numbers, called data points, at which we have observed values of some function, and denote the observed value at \( x_i \) by \( f(x_i) \). Let \( \{y_j(x)\} \) for \( j = 0, \ldots, m \) be a finite sequence of linearly independent functions defined for every \( x_i \). (Throughout this paper, \( n \) will represent the number of distinct data points used, and \( m + 1 \) will represent the number of the linearly independent functions used.)

We desire to approximate \( f \) by a linear combination of these \( m + 1 \) functions so that

\[
f(x_i) = \sum_{j=0}^{m} c_j^{(m)} y_j(x_i) \quad \text{for} \quad i = 1, \ldots, n \quad (2-1)
\]

with the \( c_j^{(m)} \)'s to be determined so that

\[
S = \sum_{i=1}^{n} \left[ f(x_i) - \sum_{j=0}^{m} c_j^{(m)} y_j(x_i) \right]^2 \quad (2-2)
\]

is minimized. The superscript \( m \) on \( c_j^{(m)} \) denotes the fact that the coefficient of \( y_j(x_i) \) will generally depend on \( m \). This approximation is called the least squares approximation of \( f \) over \( \{x_i\} \) \([5,7]\).

There are other expressions which could be used as the one to be minimized in place of (2-2) such as

\[
S = \sum_{i=1}^{n} \left| f(x_i) - \sum_{j=0}^{m} c_j^{(m)} y_j(x_i) \right| \quad (2-3)
\]

or
\[
S = \max \left\{ f(x_i) - \sum_{j=0}^{m} c_j^{(m)} y_j(x_1) \right\} \quad \text{for } i = 1, \ldots, n. 
\] (2-4)

There are some advantages in using one of these expressions as the one to be minimized. For example, using expression (2-4) has the advantage of providing an absolute upper bound on the error of our approximation. But for general applications, the difficulty in determining the \( c_j^{(m)} \)'s needed to minimize one of these expressions leads us to selecting the method of least squares.

Also there are many sequences of \( \{y_j(x)\} \) which could be used in equation (2-1). If \( \{y_j(x)\} = \{\sin(jx), \cos(jx)\} \), then equation (2-1) gives us a Fourier approximation of \( f \). This approximation is particularly useful in approximating functions which are periodic. If \( y_j(x) \) is a polynomial of degree \( j \), then equation (2-1) will give us a polynomial approximation of \( f \). A major advantage of this approximation is its ease of implementation.

It is possible to select a sequence of polynomial functions such that (2-2) can be made zero, and we can obtain an exact polynomial approximation. For example, if we choose a sequence of polynomial functions in which the number of coefficients in (2-1) is greater than or equal to the number of data points, that is \( m + 1 \geq n \), then we may use Lagrange's interpolation formula to find an exact polynomial approximation. However, this procedure is not often desirable. We must assume that the observed values \( f(x_1) \) were obtained experimentally, and that some error exists between the observed values and the true values of the function. If we look at Figure 1, we see that an exact polynomial approximation may be significantly different from
the true function. And in particular, the derivative of the exact polynomial approximation at a given point will usually be much different from the derivative of the true function at the same point. On the other hand, if we choose a sequence of polynomial functions in which \( m + 1 < n \), we can "smooth" the observed values and can obtain a function whose derivative at a given point may not be significantly different from the derivative of the true function.

![Diagram showing true function and polynomial approximations](image)

**Figure 1**

Exact Polynomial Approximations

Thus, for most practical applications, a polynomial least squares approximation in which \( m + 1 < n \) will give us a good approximation of a function defined by experiment over a discrete set of points, and the \( a_j^{(m)} \)'s needed to minimize (2.2) should not be extremely difficult to calculate.

If \( y_j(x) = x^j \), we can easily derive a system of linear equations called **normal equations** which can be used to calculate the needed
If we take the partial derivatives of \( S \) in (2-2) with respect to \( c_{k}^{(m)} \) for \( k = 0, \ldots, n \) and set them equal to zero, we obtain

\[
\frac{\partial S}{\partial c_{k}^{(m)}} = -2 \sum_{i=1}^{n} \left[ f(x_{i}) - \sum_{j=0}^{m} c_{j}^{(m)} y_{j}(x_{i}) \right] y_{k}(x_{i}) = 0. \tag{2-5}
\]

After substituting \( y_{j}(x) = x^{j} \) into (2-5) and canceling the \(-2\), we have

\[
\sum_{i=1}^{n} \left[ f(x_{i}) - \sum_{j=0}^{m} c_{j}^{(m)} x_{i}^{j} \right] x_{i}^{k} = 0. \tag{2-6}
\]

Interchanging summations, we can write

\[
\sum_{j=0}^{m} c_{j}^{(m)} \sum_{i=1}^{n} x_{i}^{j+k} = \sum_{i=1}^{n} f(x_{i}) x_{i}^{k}. \tag{2-7}
\]

Hence, using the notation

\[
s_{jk} = \sum_{i=1}^{n} x_{i}^{j+k} \quad \text{and} \quad t_{k} = \sum_{i=1}^{n} f(x_{i}) x_{i}^{k}, \tag{2-8}
\]

the normal equations can be written as

\[
\sum_{j=0}^{m} c_{j}^{(m)} s_{jk} = t_{k} \quad \text{for} \quad k = 0, \ldots, m \tag{2-9}
\]

or in expanded form as

\[
\begin{align*}
c_{0} s_{00} + c_{1} s_{10} + \cdots + c_{m} s_{m0} &= t_{0}, \\
c_{0} s_{01} + c_{1} s_{11} + \cdots + c_{m} s_{m1} &= t_{1}, \\
&\vdots \\
c_{0} s_{0m} + c_{1} s_{1m} + \cdots + c_{m} s_{mm} &= t_{m}.
\end{align*} \tag{2-10}
\]
This gives us a system of \( m + 1 \) linear equations for \( m + 1 \) unknown \( c_j \)'s. This system has a unique solution which minimizes \( S \) [7,8].

At this point, it seems that we have solved the polynomial least squares approximation problem. In fact, if \( m = 1 \), the normal equations are easily solved and yield the solution

\[
f(x) = c_0 + c_1 x
\]  

(2-11)

where

\[
c_0 = \left( s_{11}t_0 - s_{10}t_1 \right) / \left( s_{00}s_{11} - s_{01}s_{10} \right) \quad \text{and} \quad c_1 = \left( s_{00}t_1 - s_{01}t_0 \right) / \left( s_{00}s_{11} - s_{01}s_{10} \right).
\]

However, there is a problem in using the above normal equations. To illustrate this problem, let us assume that all the data points \( \{x_i\} \) are equally spaced such that \( x_i = i/n \) for \( i = 1, \ldots, n \).

Then for large \( n \),

\[
s_{jk} = \sum_{i=1}^{n} x_i^{j+k} = n \int_0^1 x^{j+k} dx = n/(j+k+1) \quad (2-12)
\]

and the matrix of coefficients in (2-10) is approximately \( n \) times

\[
\begin{bmatrix}
1 & 1/2 & 1/3 & \cdots & 1/(m+1) \\
1/2 & 1/3 & 1/4 & \cdots & 1/(m+2) \\
1/3 & 1/4 & 1/5 & \cdots & 1/(m+3) \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1/(m+1) & 1/(m+2) & 1/(m+3) & \cdots & 1/(2m+1)
\end{bmatrix}
\]  

(2-13)

This matrix is the Hilbert matrix of order \( m + 1 \), which is a classical example of an ill-conditioned matrix [7]. This means that we will have difficulties in calculating the required \( c_j \)'s when \( m \)
is large. For example, when \( m = 9 \), then the inverse of \((2-13)\)
has elements of magnitude \( 3 \times 10^{12} \). The result is that any roundoff
errors committed in calculating the \( c_j^{(m)} \)'s will be greatly magnified.
Therefore, in order to obtain an accurate solution, a very large
number of decimal places must be used in making the various calcu-
lations. For even larger \( m \), the number of decimal places needed
to obtain an accurate solution soon becomes prohibitively large.
Thus, we are led to using \textit{orthogonal polynomials}. 
By definition, a set of polynomials \( \{p_j(x)\} \) of degrees \( j = 0, \ldots, m \) is orthogonal over a set of points \( \{x_i\} \) for \( i = 1, \ldots, n \) if

\[
\sum_{i=1}^{n} p_j^{(n)}(x_i) p_k^{(n)}(x_i) = 0 \quad \text{whenever} \quad j \neq k. \tag{3-1}
\]

The superscript \( n \) on the polynomials denotes the fact that each polynomial will depend on the number of points \( n \) [7].

If we let \( y_j(x) = p_j^{(n)}(x) \) in equation (2-1), then the equations used to determine the \( c_j^{(m)} \)'s prove to be extremely easy to solve.

We proceed as we did previously by taking the partial derivatives of \( S \) in (2-2) with respect to \( c_k^{(m)} \) for \( k = 0, \ldots, m \) and setting them to zero so that

\[
\frac{\delta S}{\delta c_k^{(m)}} = -2 \sum_{i=1}^{n} \left[ f(x_i) - \sum_{j=0}^{m} c_j^{(m)} y_j(x_i) \right] y_k^{(n)}(x_i) = 0. \tag{3-2}
\]

After substituting \( y_j(x) = p_j^{(n)}(x) \) into (3-2) and canceling the \(-2\), we have

\[
\sum_{i=1}^{n} \left[ f(x_i) - \sum_{j=0}^{m} c_j^{(m)} p_j^{(n)}(x_i) \right] p_k^{(n)}(x_i) = 0. \tag{3-3}
\]

But the orthogonal property (3-1) makes most terms zero. Hence, (3-3) simplifies to

\[
\sum_{i=1}^{n} \left[ f(x_i) - c_k^{(m)} p_k^{(n)}(x_i) \right] p_k^{(n)}(x_i) = 0. \tag{3-4}
\]
Solving for the \( c_k^{(m)} \)'s, we have

\[
c_k^{(m)} = \frac{\sum_{i=1}^{n} f(x_i) p_k^{(n)}(x_i)}{\sum_{i=1}^{n} [p_k^{(n)}(x_i)]^2} \text{ for } k = 0, \ldots, m. \tag{3-5}
\]

Notice that the \( c_k^{(m)} \)'s are independent of \( m \) so that we can drop the superscript. This means that we can easily calculate the least squares approximation of degree \( m + 1 \) simply by using the coefficients obtained for the approximation of degree \( m \) and calculating one additional coefficient \( c_{m+1} \).

The above property of orthogonal polynomials greatly simplifies our calculations when we desire least squares approximations for many different degrees. However, in practice, if we are interested in finding the best least squares approximation of some unknown degree where we will have to compare several approximations of different degrees, we will use a high speed digital computer and will solve the problem only once. Thus, the ease of calculating the coefficients is not nearly as important as the accuracy obtained in using orthogonal polynomials over the normal equation discussed in the last chapter.

All that we need is a set of orthogonal polynomials. There are three methods commonly used to find a set of orthogonal polynomials. We will look at each of these methods separately.
Gram Polynomials

To construct a set of orthogonal polynomials, we assume that we have n equally spaced data points such that

\[ x_i = x_1 + (i - 1)h \text{ for } i = 1, \ldots, n. \]  

(3-6)

(In the next two sections, we will discuss methods of constructing orthogonal polynomials where the points need not be equally spaced.)

To make the notation more convenient, let us define a new variable t such that \( t = i - 1 \) and let

\[ p_j^{(n)}(x_1) = p_j^{(n)}(x_1 + th) = p_j(t, n). \]  

(3-7)

Now suppose that we let

\[ p_j(t, n) = a_0 + a_1 t + a_2 t^2 + \cdots + a_j t^j \]  

(3-8)

where \( t^r \) is the factorial \( t(t - 1) \ldots (t - r + 1) \), and we determine \( a_r \) for \( r = 0, \ldots, j \) so that \( p_j(t, n) \) is orthogonal to the factorials \( 1, (t + 1)(t + 2)(t + 1), \ldots, (t + j - 1)(j - 1) \).

It can be shown that the powers \( 1, t, t^2, \ldots, t^{j-1} \) can be written as a linear combination of these factorials. Hence, \( p_j(t, n) \) is also orthogonal to these powers. Furthermore, if \( k < j \), it can be shown that \( p_k(t, n) \) can be written as a linear combination of the above powers. Thus, \( p_j(t, n) \) and \( p_k(t, n) \) are themselves orthogonal, and we can use equation (3-8) to determine a set of orthogonal polynomials over a set of equally spaced data points [8].

In order for \( p_j(t, n) \) to be orthogonal to the factorials \( (t + s)^m \) for \( s = 0, \ldots, j - 1 \), we must have
Further, since

\[ (t + s)(s) \frac{p_j(t,n)}{t \in 0} = a_0(t + s)(s) + a_1(t + s)(s+1) + \ldots + a_j(t + s)(s+j) \] (3-10)

and

\[ \Delta t(k) = (t + 1)(k) - t(k) = t(t - 1) \ldots (t - k + 2)((t + 1) - (t - k + 1)) = \] (3-11)

\[ k t(k-1) \]

where \( \Delta \) is the forward difference operator, we can write (3-9) as

\[ \sum_{t=0}^{n-1} (t + s)(s) \frac{p_j(t,n)}{t \in 0} = \] (3-12)

\[ \sum_{t=0}^{n-1} [a_0(t + s)(s) + a_1(t + s)(s+1) + \ldots + a_j(t + s)(s+j)] = \]

\[ \sum_{t=0}^{n-1} [a_0 \Delta(t + s)(s+1)/(s + 1) + a_1 \Delta(t + s)(s+2)/(s + 2) + \ldots + a_j \Delta(t + s)(s+j+1)/(s + j + 1)] = \]

\[ a_0 [(n + s)(s+1) - (s)(s+1)]/(s + 1) + a_1 [(n + s)(s+2) - (s)(s+2)]/(s + 2) + \ldots + a_j [(n + s)(s+j+1) - (s)(s+j+1)]/(s + j + 1) = \]

\[ a_0(n + s)(s+1)/(s + 1) + a_1(n + s)(s+2)/(s + 2) + \ldots + a_j(n + s)(s+j+1)/(s + j + 1) = \]
\((n + s)^{(s+1)}[a_0/(s + 1) + a_1(n - 1)/(s + 2) + \ldots + a_j(n - 1)/(s + j + 1)] = 0.\)

After removing the factor \((n + s)^{(s+1)}\) and setting \(b_k = a_k(n - 1)^{(k)}\), we see that equation (3-12) simplifies to

\[b_0/(s + 1) + b_1/(s + 2) + \ldots + b_j/(s + j + 1) = 0\]  

for \(s = 0, \ldots, j - 1.\)

The Hilbert matrix again appears in the set of equations given by (3-13); however, solving the system exactly still leads us to a useful algorithm. Let

\[Q(s)/(s + j + 1)^{(j+1)} = b_0/(s + 1) + b_1/(s + 2) + \ldots + b_j/(s + j + 1).\]  

(3-14)

Now since \(Q(s)\) is at most a polynomial of degree \(j\), and since \(Q(s)\) must be zero at \(s = 0, \ldots, j - 1,\) we must have \(Q(s) = B_j(s)\) where \(B\) is independent of \(s\). Hence, we have

\[B_j(s)/(s + j + 1)^{(j+1)} = b_0/(s + 1) + b_1/(s + 2) + \ldots + b_j/(s + j + 1).\]  

(3-15)

To determine \(B\), multiply both sides of (3-15) by \((s + 1)\) so that

\[B_j(s)/(s + j + 1)^{(j+1)} = b_0 + (s + 1)[b_1/(s + 2) + \ldots + b_j/(s + j + 1)].\]  

(3-16)

This equation must be true for all values of \(s\) except for those where the denominator is zero. Thus, letting \(s = -1,\) we see that

\[B(-1)^{(j)}/j!(j) = b_0\]  

(3-17)
and
\[ B = j! b_0 / [( -1)( -2) \ldots ( -j)] = ( -1)^j b_0. \quad (3-18) \]

Furthermore, since there are no restrictions on \( a_0 \) in equation (3-8), we can simplify our calculations by choosing \( b_0 = a_0 = 1 \). Thus, equation (3-18) becomes
\[ B = ( -1)^j, \quad (3-19) \]
and we can rewrite equation (3-15) as
\[ ( -1)^j s(j) / (s + j + 1)(j+1) = 1 / (s + 1) + b_1 / (s + 2) + \cdots + b_j / (s + j + 1). \quad (3-20) \]

Now we can uniquely determine \( b_k \) for \( k = 1 \) to \( j \). Multiplying both sides of equation (3-20) by \( (s + j + 1)(j+1) \), we obtain
\[ ( -1)^j s(j) = (3-21) \]
\[ (s + j + 1)(j+1) \left[ 1 / (s + 1) + b_1 / (s + 2) + \cdots + b_j / (s + j + 1) \right] = (s + 1) \cdots (s + j + 1) \left[ 1 / (s + 1) + b_1 / (s + 2) + \cdots + b_j / (s + j + 1) \right]. \]

Thus, letting \( s = -k - l \), we see that
\[ ( -1)^j (-k - l)(j) = \]
\[ (-k) \cdots ( -1)(1) \ldots (j - k) b_k = (3-22) \]
\[ ( -1)^k k! (j - k)! b_k \]
and
\[ b_k = \]
\[ ( -1)^j (-k - l)(j) / (-k)^k k! (j - k)! = (3-23) \]
\[ (j + k)(j) / ( -1)^k k! (j - k)! = \]
\[ (-1)^k (j + k)(j) / k! (j - k)! \]
for \( k = 1, \ldots, j \). Recalling that \( b_k = a_k(n - 1)(k) \) and \( b_0 = a_0 = 1 \), we can write equation (3-8) as

\[
\begin{align*}
p_j(t, n) &= \sum_{k=0}^{n-1} b_k(n-1)(k) = \\
1 + b_1 t/(n-1) + b_2 t(2)/(n-1)(2) + \cdots + b_j t(j)/(n-1)(j) = \\
1 + \sum_{k=1}^{n-1} b_k(n-1)(k) = \\
1 + \sum_{k=1}^{n-1} [(-1)^k(j+k)/k! (j-k)!][t(k)/(n-1)(k)] = \\
\sum_{k=0}^{n-1} [(-1)^k(j+k)/k! (j-k)!][t(k)/(n-1)(k)]
\end{align*}
\]

Using equation (3-24), we can now determine a set of orthogonal polynomials over a set of equally spaced data points. These polynomials are usually referred to as Gram polynomials (or sometimes, confusingly, as Chebyshev polynomials). The first five of these polynomials are:

\[
\begin{align*}
p_0(t, n) &= 1, \\
p_1(t, n) &= 1 - 2t/(n - 1), \\
p_2(t, n) &= 1 - 6t/(n - 1) + 6t(t - 1)/(n - 1)(n - 2), \\
p_3(t, n) &= 1 - 12t/(n - 1) + 30t(t - 1)/(n - 1)(n - 2) - 20t(t - 1)(t - 2)/(n - 1)(n - 2)(n - 3), \\
p_4(t, n) &= 1 - 20t/(n - 1) + 90t(t - 1)/(n - 1)(n - 2) - 140t(t - 1)(t - 2)/(n - 1)(n - 2)(n - 3) + 70t(t - 1)(t - 2)(t - 3)/(n - 1)(n - 2)(n - 3)(n - 4).
\end{align*}
\]

These polynomials can be used with equations (2-1) and (3-5) to solve
the polynomial least squares approximation problem for equally spaced data points. For hand computing, tables of \( p_j(t,n) \) exist and should be used [2,4].

**Gram-Schmidt Orthogonalization Process**

The Gram-Schmidt orthogonalization process is a convenient method to generate a set of orthogonal polynomials over a set of data points \( \{x_i\} \) for \( i = 1, \ldots, n \) which need not be equally spaced [4,7]. We begin with a set of \( m+1 \) polynomials \( \{q_j(x)\} \), where \( q_j(x) \) is a polynomial of degree \( j \) and \( m+1 < n \). Furthermore, the polynomials in this set must be linearly independent over the set of data points; that is, there exist no constants \( \{a_j\} \) other than zero such that

\[
\sum_{j=0}^{m} a_j q_j(x_i) = 0 \quad \text{for} \quad i = 1, \ldots, n. \quad (3-26)
\]

A convenient choice for \( \{q_j(x)\} \) is often \( 1, x, x^2, \ldots, x^m \). In any case, let

\[
p_0(x) = q_0(x) \quad \text{and} \quad p_j(x) = q_j(x) - \sum_{r=0}^{j-1} b_{jr} p_r(x) \quad \text{for} \quad j = 1, \ldots, m. \quad (3-27)
\]

where the \( b_{jr} \)'s will be determined so that \( p_j(x) \) will be orthogonal to the function space generated by \( q_0(x), \ldots, q_{j-1}(x) \). Suppose that we have determined \( p_0(x), \ldots, p_{j-1}(x) \) so that they form an orthogonal basis for the above function space. We desire to determine \( p_j(x) \) so that it will also be orthogonal to this basis. Thus, we want
for \( k = 0, \ldots, j - 1 \). By orthogonality, all terms in the double summation on the bottom line of (3-28) are zero except the term in which \( r = k \). Therefore, equation (3-28) can be written

\[
\sum_{i=1}^{n} a_j(x_i) p_k(x_i) - \sum_{r=0}^{j-1} b_{jr} \sum_{i=1}^{n} p_r(x_i) p_k(x_i) = 0
\]

for \( k = 0, \ldots, j - 1 \). Solving for \( b_{jk} \), we have

\[
b_{jk} = \frac{\sum_{i=1}^{n} a_j(x_i) p_k(x_i)}{\sum_{i=1}^{n} [p_k(x_i)]^2}
\]

for \( k = 0, \ldots, j - 1 \). Thus, using equation (3-27) and equation (3-30), we can determine \( p_j(x) \) which will be orthogonal to all lower degree polynomials.

The Gram-Schmidt orthogonalization process can easily be used by high speed digital computers to generate a set of orthogonal polynomials over any set of points. However, there is an even more convenient and efficient method which uses a recurrence relation.
Forsythe Recurrence Relation

The method discussed in this section was given by Forsythe. This method, in one form or another, is becoming the most general method used by industry to approximate functions defined on a discrete set of points \([1,3]\).

For this method, suppose that we have a set of polynomials \([p_j(x)]\) such that

\[
\begin{align*}
p_{-1}(x) &= 0 \\
p_0(x) &= 1, \text{ and} \\
p_{j+1}(x) &= (x - a_{j+1})p_j(x) - b_j p_{j-1}(x).
\end{align*}
\]

We can show by induction that the constants \(a_{j+1}\) and \(b_j\) can be determined so that \([p_j(x)]\) is orthogonal over a set of points \([x_i]\).

For \(j = 0\), we see that

\[
p_1(x) = (x - a_1).
\]

In order for this polynomial to satisfy the orthogonality property (3-31), we must have

\[
\sum_{i=1}^{n} p_0(x_i)p_1(x_i) = \sum_{i=1}^{n} (x_i - a_1) = 0.
\]

Hence, we see that

\[
a_1 = \frac{\sum_{i=1}^{n} x_i}{n}.
\]

Next suppose that we have determined the constants \(a_1, \ldots, a_j\) and \(b_1, \ldots, b_{j-1}\) so that the set of polynomials \([p_j(x)]\) for
j = 0, ..., k is orthogonal over the set of points \{x_i\}. We must show that we can choose \(a_{k+1}\) and \(b_k\) so that \(p_{k+1}(x)\) will also satisfy the orthogonality property.

Using the recurrence relation in (3-31), we can write

\[
\sum_{i=1}^{n} p_j(x_i)p_{k+1}(x_i) = \sum_{i=1}^{n} \left[ p_j(x_i)[(x_i - a_{k+1})p_k(x_i) - b_kp_{k-1}(x_i)] \right]
\]

where

\[
\sum_{i=1}^{n} x_i p_j(x_i)p_k(x_i) - a_{k+1} \sum_{i=1}^{n} p_j(x_i)p_k(x_i) - b_k \sum_{i=1}^{n} p_j(x_i)p_{k-1}(x_i).
\]

But by our induction hypothesis,

\[
a_{k+1} \sum_{i=1}^{n} p_j(x_i)p_k(x_i) = 0 \text{ whenever } j \neq k, \quad (3-36)
\]

\[
b_k \sum_{i=1}^{n} p_j(x_i)p_{k-1}(x_i) = 0 \text{ whenever } j \neq k - 1. \quad (3-37)
\]

Moreover, for \(j = 0, ..., k - 2\), \([x p_j(x)]\) is a polynomial of degree no greater than \(k - 1\). Hence, \([x p_j(x)]\) can be expressed as a linear combination of the polynomials \(p_0(x), ..., p_{k-1}(x)\). And, again by our induction hypothesis,

\[
\sum_{i=1}^{n} x_i p_j(x_i)p_k(x_i) = 0 \text{ for } j = 0, ..., k - 2. \quad (3-38)
\]

Thus, equation (3-35) simplifies to
\[ \sum_{i=1}^{n} p_j(x_1)p_{k+1}(x_1) = 0 \text{ for } j = 0, \ldots, k - 2. \quad (3-39) \]

Therefore, for \( j = 0, \ldots, k - 2 \), \( p_{k+1}(x) \) will satisfy the orthogonality property regardless of the values chosen for \( a_{k+1} \) and \( b_k \).

In order for equation (3-35) to equal zero when \( j = k - 1 \), we must have

\[ \sum_{i=1}^{n} x_1 p_{k-1}(x_1)p_k(x_1) - b_k \sum_{i=1}^{n} [p_{k-1}(x_1)]^2 = 0 \quad (3-40) \]

which gives us the requirement that

\[ b_k = \frac{\sum_{i=1}^{n} x_1 p_{k-1}(x_1)p_k(x_1)}{\sum_{i=1}^{n} [p_{k-1}(x_1)]^2}. \quad (3-41) \]

And in order for equation (3-35) to equal zero when \( j = k \), we must have

\[ \sum_{i=1}^{n} x_1 [p_k(x_1)]^2 - a_{k+1} \sum_{i=1}^{n} [p_k(x_1)]^2 = 0 \quad (3-42) \]

which gives us the requirement that

\[ a_{k+1} = \frac{\sum_{i=1}^{n} x_1 [p_k(x_1)]^2}{\sum_{i=1}^{n} [p_k(x_1)]^2}. \quad (3-43) \]

Thus, if \( a_{k+1} \) and \( b_k \) are chosen according to (3-41) and (3-43), then \( p_{k+1}(x) \) will satisfy the orthogonality property for all \( j \leq k \).

This completes our proof by induction that it is possible to choose \( a_{k+1} \) and \( b_k \) so that the set of polynomials \( \{ p_j(x) \} \) generated by (3-31) will be orthogonal over the set of points \( \{ x_1 \} \).

Using the recurrence relation in (3-31), we can derive an
alternative formula for \( b_k \). We can write

\[
\sum_{i=1}^{n} \left[ \frac{p_k(x_i)}{x_i} \right]^2 = \sum_{i=1}^{n} p_k(x_i)p_k(x_i) = \sum_{i=1}^{n} p_k(x_i)[(x_i - a_k)p_{k-1}(x_i) - b_{k-1}p_{k-2}(x_i)] = \sum_{i=1}^{n} x_i p_{k-1}(x_i)p_k(x_i) - a_k \sum_{i=1}^{n} p_{k-1}(x_i)p_k(x_i) - b_{k-1} \sum_{i=1}^{n} p_{k-2}(x_i)p_k(x_i) = \sum_{i=1}^{n} x_i p_{k-1}(x_i)p_k(x_i).
\]

Hence, we can write equation (3-41) as

\[
b_k = \frac{\sum_{i=1}^{n} \left[ p_k(x_i) \right]^2}{\sum_{i=1}^{n} \left[ p_{k-1}(x_i) \right]^2}. \quad (3-45)
\]

Thus, using (3-31), (3-43), and (3-45), we can easily generate a set of orthogonal polynomials over any discrete set of points. Further, since these equations are not difficult to mechanize, we can use this method to solve the least squares approximation problem on a digital computer.
Now that we have solved the least squares approximation problem, we need to investigate how accurate we can expect this solution to be. A standard measure of error is the root-mean-square error $[6]$. The root-mean-square error in an approximation over a set of points $\{x_1\}$ is defined to be

$$\text{RMS error} = \left\{ \sum_{i=1}^{n} \left[ f(x_1) - \sum_{j=0}^{m} c_j y_j(x_1) \right]^2 / N \right\}^{1/2}. \quad (4-1)$$

Notice that the numerator in the above fraction is just the quantity that was minimized when obtaining the least squares approximation of the function $f$.

The interpretation of the root-mean-square error depends upon the context in which it is used. If the values $f(x_1)$ are considered to be the exact values of a true function, then this error represents the actual deviation of the approximation from the true function over $\{x_1\}$. On the other hand, if the values $f(x_1)$ are empirical values which correspond to an observed function, then this error only represents the deviation of the approximation from the observed function. And unless additional information is supplied, nothing can be determined about the actual deviation between the approximation and the true function.

Generally, however, it is assumed that the true function is such that it can be represented exactly by a linear combination of $\{y_j(x)\}$,
but due to the presence of random errors in the observed values, it
is impossible to obtain this representation. Using this assumption,
it is possible to estimate the root-mean-square deviation between
the observed function and the true function and to estimate the errors
in the calculated coefficients.

If we let \( \overline{c}_0, \ldots, \overline{c}_m \) denote the calculated coefficients, then
the observed function \( \overline{f}(x) \) can be approximated by

\[
\overline{f}(x) = \sum_{j=0}^{m} \overline{c}_j y_j(x).
\]

From (2-5), we see that

\[
\frac{\delta S}{\delta \overline{c}_k} = -2 \sum_{i=1}^{n} \left[ \overline{f}(x_i) - \sum_{j=0}^{m} \overline{c}_j y_j(x_i) \right] y_k(x_i) = 0
\]

for \( k = 0, \ldots, m \). And after canceling the \(-2\) and interchanging
the summations, we have

\[
\sum_{j=0}^{m} \overline{c}_j \sum_{i=1}^{n} y_j(x_i) y_k(x_i) = \sum_{i=1}^{n} y_k(x_i) \overline{f}(x_i).
\]

Hence, the normal equations can be written in the form

\[
\sum_{j=0}^{m} \overline{c}_j s_{jk} = t_k \text{ for } k = 0, \ldots, m
\]

where

\[
s_{jk} = \sum_{i=1}^{n} y_j(x_i) y_k(x_i) \quad \text{and} \quad t_k = \sum_{i=1}^{n} y_k(x_i) \overline{f}(x_i).
\]

(These equations are similar to the normal equations given by (2-8).)
and (2-9); however, they are in a slightly more general form so that the following results will be applicable to all least squares approximations.

Letting $S_{jk}$ be the cofactor of $s_{jk}$ in the matrix of coefficients of the normal equations given by (4-5) and letting $D$ be the determinant of this matrix, we can use Cramer's rule to write

$$
\bar{c}_k = (4-7)
$$

If we define

$$
Y_k(x) = \sum_{j=0}^{m} \frac{S_{jk}/D}{y_j(x)} f(x),
$$

then equation (4-7) can be written as

$$
\bar{c}_k = \sum_{i=1}^{n} Y_k(x_i) f(x). \quad (4-9)
$$

Furthermore, if we assume that the true function can be written as

$$
f(x) = \sum_{j=0}^{m} c_j y_j(x), \quad (4-10)
$$

then in a similar fashion, we can derive the equation

$$
c_k = \sum_{i=1}^{n} Y_k(x_i) f(x). \quad (4-11)
$$
Hence, if \( E(x_1) = f(x_1) - \bar{f}(x_1) \), then the actual error in the calculated value for \( c_k \) is

\[
c_k - \bar{c}_k = \sum_{i=1}^{n} y_k(x_1)f(x_1) - \sum_{i=1}^{n} y_k(x_1)\bar{f}(x_1) = \sum_{i=1}^{n} y_k(x_1)E(x_1).
\]

Generally, there is no way to determine \( E(x_1) \). However, if we assume that errors are randomly distributed so that the mean of the product \( E(x_1)E(x_j) \) is zero when \( i \neq j \), then the expected value or mean \( M \) of \( (c_k - \bar{c}_k)^2 \) can be written as

\[
M[(c_k - \bar{c}_k)^2] = \left\{ \left( \sum_{i=1}^{n} y_k(x_1)E(x_1) \right)^2 \right\} = M \left\{ \sum_{i=1}^{n} \sum_{j=1}^{n} y_k(x_1)y_k(x_j)E(x_1)E(x_j) \right\} = \sum_{i=1}^{n} \sum_{j=1}^{n} y_k(x_1)y_k(x_j)M[E(x_1)E(x_j)] = \sum_{i=1}^{n} y_k(x_1)^2 M[E(x_1)^2].
\]

Letting \( E_m^2 \) equal the mean of \( [E(x_0)]^2, \ldots, [E(x_n)]^2 \), we can write equation (4-13) as

\[
M[(c_k - \bar{c}_k)^2] = E_m^2 \sum_{i=1}^{n} [y_k(x_1)]^2.
\]

Equation (4-14) can be written in a more convenient form. Notice from (4-2) that if \( f(x) = y_s(x) \) for \( 0 \leq s \leq m \), then \( \tilde{c}_k = \delta_{sk} \).
where $\delta_{sk} = 1$ when $s = k$ and 0 otherwise. Thus, we can deduce from (4-9) that

$$\sum_{i=1}^{n} Y_k(x_i)y_s(x_i) = \delta_{sk}.$$  

(4-15)

Using equation (4-8), we can write

$$\sum_{i=1}^{n} [Y_k(x_i)]^2 =$$  

(4-16)

$$\sum_{i=1}^{n} Y_k(x_i)x_k(x_i) =$$  

$$\sum_{i=1}^{n} Y_k(x_i) \left[ \sum_{j=0}^{m} S_{jk}y_j(x_i)/D \right] =$$  

$$\sum_{j=0}^{m} [S_{jk}/D] \sum_{i=1}^{n} Y_k(x_i)y_j(x_i) =$$  

$$\sum_{j=0}^{m} [S_{jk}/D] \delta_{jk} = S_{kk}/D.$$  

Hence, equation (4-14) can be written as

$$Mf(c_k - \bar{c}_k)^2 = \frac{\sigma^2}{m} S_{kk}/D.$$  

(4-17)

It is possible to obtain an estimate of $\frac{\sigma^2}{m}$ in terms of the residuals $R(x_1), \ldots, R(x_n)$ where

$$R(x_i) = \bar{F}(x_i) - \sum_{k=0}^{m} \bar{c}_k y_k(x_i).$$  

(4-18)

Using this definition and equations (4-10) and (4-12), we can write

$$B(x_1) + R(x_1) =$$  

(4-19)

$$f(x_1) - \sum_{k=0}^{m} \bar{c}_k y_k(x_1) =$$  

$$\sum_{k=0}^{m} c_k y_k(x_1) - \sum_{k=0}^{m} \bar{c}_k y_k(x_1) =$$
Also from equation (4-3), we see that

\[ \sum_{i=1}^{n} R(x_i) y_k(x_i) = \sum_{i=1}^{n} \left[ \bar{F}(x_i) - \sum_{j=0}^{m} \bar{C}_j y_j(x_i) \right] y_k(x_i) = 0 \quad (4-20) \]

for \( k = 0, \ldots, m \). Hence, using equations (4-19) and (4-20), we obtain

\[ \sum_{i=1}^{n} R(x_i) [E(x_i) + R(x_i)] = \sum_{i=1}^{n} \left[ \bar{F}(x_i) - \sum_{j=0}^{m} \bar{C}_j y_j(x_i) \right] = 0. \]  

And we have

\[ \sum_{i=1}^{n} R(x_i) E(x_i) = - \sum_{i=1}^{n} [R(x_i)]^2. \quad (4-22) \]

Using this result, we can write

\[ M \left\{ \sum_{i=1}^{n} E(x_i) [E(x_i) + R(x_i)] \right\} = M \left\{ \sum_{i=1}^{n} [E(x_i)]^2 + \sum_{i=1}^{n} R(x_i) E(x_i) \right\} = \]

\[ M \left\{ \sum_{i=1}^{n} [E(x_i)]^2 - \sum_{i=1}^{n} [R(x_i)]^2 \right\} = \sum_{i=1}^{n} M([E(x_i)]^2) - \sum_{i=1}^{n} M([R(x_i)]^2) = \]

\[ \frac{nE^2}{m} - \sum_{i=1}^{n} M([R(x_i)]^2). \quad (4-23) \]

Furthermore, again using the assumption that the mean product
\( E(x_i)E(x_j) \) is zero when \( i \neq j \) and using equation (4-19), we can write

\[
M\left\{ \sum_{i=1}^{n} E(x_i)[E(x_i) + R(x_i)] \right\} = \tag{4-24}
\]

\[
M\left\{ \sum_{i=1}^{n} E(x_i)[ \sum_{k=0}^{m} \sum_{v=1}^{n} y_k(x_i) y_k(x_i) E(x_v)E(x_i)] \right\} = \tag{4-25}
\]

\[
M\left\{ \sum_{i=1}^{n} \sum_{v=1}^{n} y_k(x_i) y_k(x_i) E(x_v)E(x_i) \right\} = \tag{4-26}
\]

\[
M\left[ \sum_{i=1}^{n} \sum_{v=1}^{n} y_k(x_i) y_k(x_i) M[E(x_v)E(x_i)] \right] = \tag{4-27}
\]

\[
\sum_{k=0}^{m} \sum_{i=1}^{n} y_k(x_i) y_k(x_i) M[E(x_i)^2] = \tag{4-28}
\]

\[
\sum_{k=0}^{m} \sum_{i=1}^{n} y_k(x_i) y_k(x_i) E_m^2. \tag{4-29}
\]

Hence, using equation (4-15), we can rewrite (4-24) as

\[
M\left\{ \sum_{i=1}^{n} E(x_i)[E(x_i) + R(x_i)] \right\} = (m + 1)E_m^2. \tag{4-25}
\]

Combining equations (4-23) and (4-25), we obtain

\[
n E_m^2 - \sum_{i=1}^{n} M[R(x_i)]^2 = (m + 1)E_m^2. \tag{4-26}
\]

And solving for \( E_m^2 \), we have

\[
E_m^2 = \sum_{i=1}^{n} M[R(x_i)]^2 / (n - m - 1). \tag{4-27}
\]

Thus, since for each data point we can calculate only a single residual, the best approximation of \( M[R(x_i)]^2 \) is \( [R(x_i)]^2 \).

Hence, we can estimate the root-mean-square deviation between the
observed function and the true function as

$$\text{RMS error} = \left[ \frac{E_m^2}{n} \right]^{1/2} = \left\{ \frac{\sum_{i=1}^{n} [R(x_i)]^2}{(n - m - 1)} \right\}^{1/2}, \quad (4-28)$$

and using equation (4-17), we can estimate the errors in the calculated coefficients by

$$M[(c_k - \bar{c}_k)^2] = [S_{kk}/D] \sum_{i=1}^{n} [R(x_i)]^2/(n - m - 1). \quad (4-29)$$

Furthermore, if we let $y_j(x) = p_j(x)$, then using the orthogonality property (3-1) and equation (4-6), we can write the matrix of coefficients of the normal equations given by (4-5) as

$$\begin{pmatrix}
s_{00} & 0 & 0 & \cdots & 0 \\
0 & s_{11} & 0 & \cdots & 0 \\
0 & 0 & s_{22} & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & s_{mm}
\end{pmatrix} \quad (4-30)$$

where

$$s_{kk} = \sum_{i=1}^{n} [p_k(x_i)]^2. \quad (4-31)$$

Hence, we see that

$$[S_{kk}/D] = 1/s_{kk} = 1/\sum_{i=1}^{n} [p_k(x_i)]^2. \quad (4-32)$$

One application of the above results is that we can determine the degree of the polynomial least squares approximation which will give the best results [5,7]. Our initial hypothesis is that the true
function can be represented exactly by a linear combination of
\[ \{y_j(x)\} \]
such that
\[ f(x) = \sum_{j=0}^{M} c_j y_j(x) \]  
(4-33)

for some \( M \). Thus, if we knew \( M \) and used the observed data to
calculate the \( \bar{c}_j \)'s for \( j = 0, \ldots, m \) where \( m > M \), the \( \bar{c}_j \)'s
should be equal to zero for all \( j > M \). In practice, however, it is
unlikely that any of the calculated coefficients will actually be
zero because of errors in the observed data. But we can use the
maximum likelihood statistical method to test the likelihood that
the \( \bar{c}_j \)'s are zero for all \( j > M \) \([9]\). This method is not discussed
in this paper. Here only the results of this test are stated. If
the \( \bar{c}_j \)'s for all \( j > M \) are statistically zero, then the expected
values of the RMS errors will be independent of \( m \) for \( m = M, M + 1, \ldots, n - 1 \). Thus, to find \( M \), all that we need to do is to calculate
the RMS errors when \( m = 1, 2, \ldots \) and to continue as long as the RMS
errors significantly decrease as \( m \) increases. As soon as a value
of \( m \) is obtained after which no significant decrease occurs in the
RMS error, we can conclude that this value of \( m \) is \( M \).

To calculate the RMS errors for \( m = 1, 2, \ldots \), we must calculate
the residuals defined by equation (4-8). These residuals depend upon
the calculated coefficients \( \bar{c}_0, \ldots, \bar{c}_m \), which generally depend upon
\( m \). However, when orthogonal polynomials are used, the calculated
coefficients are independent of \( m \). Thus, we can usually simplify
the calculations needed to find the degree of the polynomial least
squares approximation when we use orthogonal polynomials. Furthermore,
if the polynomial least squares approximation is a high degree polynomial, we must use orthogonal polynomials in order to avoid the ill-conditioned nature of the normal equations. But we should not arbitrarily use orthogonal polynomials to solve every polynomial least squares approximation problem. There are some applications in which using the normal equations gives satisfactory results.
CHAPTER FIVE

Applications

There are two major applications of least squares polynomials for discrete data: data smoothing and approximate differentiation [8]. Both applications are based upon the assumption that values obtained from using the least squares polynomial will be closer to the exact values of the true function than the empirical values of the observed function. Thus, we can smooth the empirical data by accepting the least squares polynomial

\[ f(x) = c_0 + c_1 x + \cdots + c_m x^m \]  \hspace{1cm} (5-1)

in place of the observed function. And we can obtain an approximation of the derivatives of the true function by differentiating (5-1).

Because of the commonness of these applications, it is often convenient to derive formulas which give smoothed values and derivatives in terms of the empirical data. For illustration, let us derive formulas when \( m = 2 \) and \( n = 5 \). Assuming that the five data points \((x_1, y_1)\) are equally spaced such that

\[ x_i = x_1 + (i - 1)h \text{ for } i = 1, \ldots, 5, \]  \hspace{1cm} (5-2)

we can define a new variable \( r \) such that \( r = (x - x_3)/h \), and we can express the second degree approximation as

\[ f(r) = c_0 + c_1 r + c_2 r^2. \]  \hspace{1cm} (5-3)
For this arrangement, the normal equations can be written as

\[
\begin{align*}
\cos\theta & + c_1 s_1 + c_2 s_2 = t_0, \\
\cos\theta & + c_1 s_1 + c_2 s_2 = t_1, \\
\cos\theta & + c_1 s_1 + c_2 s_2 = t_2,
\end{align*}
\]

where

\[
 s_{jk} = \sum_{i=1}^{5} r_{i+k} \quad \text{and} \quad t_k = \sum_{i=1}^{5} y_i r_k.
\]

Since \( r_1 = -2, \ r_2 = -1, \ r_3 = 0, \ r_4 = 1, \) and \( r_5 = 2, \) we see that the normal equations simplify to

\[
\begin{align*}
5c_0 + 10c_2 &= t_0, \\
10c_1 &= t_1, \\
10c_0 + 34c_2 &= t_2,
\end{align*}
\]

which are easily solved. Using Cramer's rule, we have

\[
c_0 = \frac{(340t_0 - 100t_2)}{700} = \frac{(34t_0 - 10t_2)}{70} = \frac{(34y_1 + 34y_2 + 34y_3 + 34y_4 + 34y_5 - 40y_1 - 10y_2 - 10y_4 - 10y_5)}{70},
\]

\[
c_1 = \frac{(170t_1 - 100t_2)}{700} = \frac{7t_1}{70} = \frac{(-14y_1 - 7y_2 + 7y_4 + 14y_5)}{70}, \quad \text{and}
\]

\[
c_2 = \frac{(50t_2 - 100t_0)}{700} = \frac{(5t_2 - 10t_0)}{70} = \frac{(20y_1 + 5y_2 + 5y_4 + 20y_5 - 10y_1 - 10y_2 - 10y_3 - 10y_4 - 10y_5)}{70}.
\]
(10y_1 - 5y_2 - 10y_3 - 5y_4 + 10y_5)/70. 

Hence, using equation (5-3), we obtain

\[ f(-2) = c_0 - 2c_1 + 4c_2 = \]
\[ (6y_1 + 18y_2 - 6y_3 - 10y_4 + 6y_5)/70 = \]
\[ (3y_1 + 9y_2 - 3y_3 - 5y_4 + 3y_5)/35, \]

\[ f(-1) = c_0 + c_1 + c_2 = \]
\[ (18y_1 + 26y_2 + 24y_3 + 12y_4 - 10y_5)/70 = \]
\[ (9y_1 + 13y_2 + 12y_3 + 6y_4 - 5y_5)/35, \]

\[ f(0) = c_0 = \]
\[ (-3y_1 + 12y_2 + 17y_3 + 12y_4 - 3y_5)/35, \]

\[ f(1) = c_0 + c_1 + c_2 = \]
\[ (-10y_1 + 12y_2 + 24y_3 + 26y_4 + 18y_5)/70 = \]
\[ (-5y_1 + 6y_2 + 13y_3 + 9y_4 + 9y_5)/35, \text{ and} \]

\[ f(2) = c_0 + 2c_1 + 4c_2 = \]
\[ (6y_1 - 10y_2 - 6y_3 + 18y_4 + 62y_5)/70 = \]
\[ (3y_1 - 5y_2 - 3y_3 + 9y_4 + 31y_5)/35. \]

Equations (5-9) through (5-13) are called smoothing formulas.

(Actually, because of symmetry, it was not necessary to derive equations for \( f(1) \) and \( f(2) \).) Given a set of data points, we do not need to fit one polynomial to all points. Instead, we can fit part to one polynomial and another part to another polynomial. Thus, we can use equation (5-11) to smooth all data points except the end points, and we can use the other equations to smooth the end points.
In this way, each data point is replaced by a smoothed value obtained by fitting every five points to a different parabola.

Furthermore, we can use the above results to obtain formulas for approximate differentiation. Differentiating equation (5-3), we have

\[ f'(x) = f'(r) \frac{dr}{dx} = \left( c_1 + 2c_2 r \right) / h. \]  
(5-14)

Hence,

\[ f'(0) = \frac{c_1}{h} = \frac{(-14y_1 - 7y_2 + 7y_4 + 14y_5)}{70h}, \]  
(5-15)

\[ f'(1) = \frac{c_1 + 2c_2}{h} = \frac{(6y_1 - 19y_2 - 20y_3 - 3y_4 + 34y_5)}{70h}, \text{ and} \]  
(5-16)

\[ f'(2) = \frac{c_1 + 4c_2}{h} = \frac{(26y_1 - 27y_2 - 40y_3 - 13y_4 + 54y_5)}{70h}. \]  
(5-17)

Thus, equations (5-15) through (5-19) provide us with formulas to approximate the derivatives of the true function. This approximation should give better estimates of the true derivatives than estimates obtained from differentiating an exact polynomial approximation which passes through each observed data point. Even small errors between the observed values and the true values of a function are greatly magnified when differentiating an exact polynomial.
approximation. However, a least squares polynomial does not pass through each data point. It passes between the observed data points and provides a smoothing effect. This smoothing effect usually leads to better estimates of the derivatives.

In order to see how well these equations work, let \( y = x^{1/2} \) be the true function and add a random error between -0.05 and 0.05 to each exact value to obtain the corresponding observed value. Using the equations for data smoothing and approximate differentiation, we obtain the values shown in Table 1.

Notice in Table 1 that the approximated values are better in the middle of the table than they are at the end points. This phenomenon is typical with numerical approximations.

Table 1

Examples of data smoothing and approximate differentiation

<table>
<thead>
<tr>
<th>( x_i )</th>
<th>True ( y_1 )</th>
<th>Observed ( y_1 )</th>
<th>Smoothe ( y_1 )</th>
<th>True ( y'_1 )</th>
<th>Computed ( y'_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
<td>0.97</td>
<td>0.98</td>
<td>0.50</td>
<td>0.45</td>
</tr>
<tr>
<td>2</td>
<td>1.41</td>
<td>1.40</td>
<td>1.39</td>
<td>0.35</td>
<td>0.38</td>
</tr>
<tr>
<td>3</td>
<td>1.73</td>
<td>1.76</td>
<td>1.74</td>
<td>0.29</td>
<td>0.31</td>
</tr>
<tr>
<td>4</td>
<td>2.00</td>
<td>1.97</td>
<td>2.00</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>5</td>
<td>2.24</td>
<td>2.24</td>
<td>2.22</td>
<td>0.22</td>
<td>0.21</td>
</tr>
<tr>
<td>6</td>
<td>2.45</td>
<td>2.41</td>
<td>2.42</td>
<td>0.20</td>
<td>0.21</td>
</tr>
<tr>
<td>7</td>
<td>2.65</td>
<td>2.61</td>
<td>2.61</td>
<td>0.19</td>
<td>0.20</td>
</tr>
<tr>
<td>8</td>
<td>2.83</td>
<td>2.82</td>
<td>2.83</td>
<td>0.18</td>
<td>0.19</td>
</tr>
<tr>
<td>9</td>
<td>3.00</td>
<td>3.04</td>
<td>3.01</td>
<td>0.17</td>
<td>0.16</td>
</tr>
<tr>
<td>10</td>
<td>3.16</td>
<td>3.14</td>
<td>3.15</td>
<td>0.16</td>
<td>0.13</td>
</tr>
</tbody>
</table>
CHAPTER SIX

Summary

In this paper, we have seen that orthogonal polynomials can be used to determine polynomial least squares approximations of functions whose values are known only over a discrete set of data points. However, orthogonal polynomials are not needed in every case. In deriving the formulas for smoothing data and approximate differentiation, we are interested in fitting one polynomial to one group of points and another polynomial to another group of points. In this case, since we are generally interested in looking only at a small number of points at one time, we can find a polynomial of low degree to fit each group of points. Thus, we can use the normal equations to derive formulas which give satisfactory results. In this case, there would be no benefit in going through the more complicated computations needed to use orthogonal polynomials. On the other hand, when we desire to fit a polynomial to a large number of points, we may need to use a polynomial of high degree. Since the normal equations prove to be badly ill-conditioned for high degree polynomials, we are led to using orthogonal polynomials.

This paper has discussed three methods for obtaining a set of orthogonal polynomials. Each of these methods basically solves the same problem in an identical manner, so that we would not expect one method to be any more accurate than another method. However, in a particular problem, one method may be more convenient than another. For hand computing, we can use existing tables for Gram polynomials.
when our data points are equally spaced. Otherwise, we must use the
Gram-Schmidt orthogonalization process or the Forsythe recurrence
relation. Both of these methods can be easily adapted for use by a
digital computer to simplify calculations. In using the Gram-Schmidt
orthogonalization process, we must use all previously determined ortho-
gonal polynomials to find the next orthogonal polynomial of higher
degree. In using the Forsythe recurrence relation, we use only the
two most recently determined orthogonal polynomials to find the next
orthogonal polynomial of higher degree. Thus, we should be able to
write more efficient computer programs using the Forsythe recurrence
relation than using the Gram-Schmidt orthogonalization process.

There is one additional advantage in using orthogonal polynomials
besides avoiding the ill-conditioned nature of the normal equations.
When we desire to calculate the polynomial least squares approximations
for many different degrees over the same set of points, it is con-
venient to use orthogonal polynomials. When using orthogonal poly-
nomials, each new approximation of degree \( m + 1 \) is obtained simply
by adding the appropriate multiple of the orthogonal polynomial in
the set of degree \( m + 1 \) to the previously determined approximation
of degree \( m \). Had we used the normal equations, each approximation
must be determined independently of every other approximation. Thus,
while it is not always necessary or desirable to use orthogonal pol-
nomials in obtaining a polynomial least squares approximation, there
are some circumstances in which using orthogonal polynomials provides
us with a better method for determining the polynomial least squares
approximation of a function.


APPENDIX A

(This program is written in UCSD Pascal. It uses Gram polynomials to determine the least squares approximation of a function whose values are known only over a discrete set of equally spaced data points.)

Program Gram;
Uses Transcend;
Const
M=4;
N=9;
Var
RMS:Real;
DATA:Text;
I,J,T:Integer;
X:Array[0..8] of Real;
F:Array[0..8] of Real;
R:Array[0..8] of Real;
B:Array[0..4] of Real;
C:Array[0..4] of Real;
D:Array[0..4] of Real;
E:Array[0..4] of Real;
S:Array[0..4] of Real;
P:Array[0..4,0..8] of Real;
Q:Array[0..4,0..4] of Real;
Procedure Getdata;
(This section reads the required empirical data.)
Begin
Reset(DATA,'DATA.TEXT');
For T:=0 to N-1 do
  Read(DATA,X[T],F[T])
End;
Procedure Constants;
(This section defines the initial constants for the Gram polynomials.)
Begin
P[0,0]:=1; P[1,0]:=1;  P[2,0]:=1;  P[3,0]:=1;  P[4,0]:=1;
P[0,1]:=1; P[1,1]:=3/4; P[2,1]:=1/4; P[3,1]:=-1/2; P[4,1]:=-3/2;
P[0,2]:=1; P[1,2]:=1/2; P[2,2]:=-2/7; P[3,2]:=-13/14; P[4,2]:=-11/14;
P[0,3]:=1; P[1,3]:=1/4; P[2,3]:=-17/28; P[3,3]:=-9/14; P[4,3]:=9/14;

Procedure Coefficients;

(This section calculates the coefficients needed to determine the least squares approximation.)

Begin
For J:=0 to M do
BC,J:=0;
For J:=0 to M do
For T:=0 to N-1 do
BC,J:=BC,J+FCT,J*F'E,J,T;
End;

Procedure Approximation;

(This section calculates the actual least squares approximation.)

Var
A, H: Real;

Begin
Q[0,0]:=1; Q[1,0]:=4; Q[2,0]:=28; Q[3,0]:=84; Q[4,0]:=168;
Q[0,1]:=0; Q[1,1]:=-1; Q[2,1]:=-24; Q[3,1]:=-181; Q[4,1]:=-872;
Q[0,2]:=0; Q[1,2]:=1; Q[2,2]:=3; Q[3,2]:=60; Q[4,2]:=557;
Q[0,3]:=0; Q[1,3]:=0; Q[2,3]:=0; Q[3,3]:=-5; Q[4,3]:=-112;
Q[0,4]:=0; Q[1,4]:=0; Q[2,4]:=0; Q[3,4]:=0; Q[4,4]:=7;

For I:=0 to 4 do
Begin
DI:=0;
EIJ:=0;
End;

A:=-X[0];
H:=X[I] - X[0];

For I:=0 to M do
For J:=0 to M do
D[I]:=D[I] + EIJ*Q[J, I]/Q[I, 0];
EC3 := (1/(H*H*H))*D[3] + (A/H)*4*D[4])
EC4 := (1/(H*H*H*H))*D[4]

Procedure Error:
(This section calculates the RMS error.)
Begin
For T := 0 to N-1 do
Begin
R[T] := F[T];
For J := 0 to M do
End;
RMS := 0;
For T := 0 to N-1 do
RMS := RMS + Sqr(R[T]);
RMS := Sqrt(RMS/(N-M-1))
End;

Procedure Print:
(This section prints the solution to the least squares approximation problem.)
Begin
For I := 0 to M do
WriteIn('D('I,') = ',D[I]:10:5);
WriteIn;
For I := 0 to M do
WriteIn('E('I,') = ',E[I]:10:5);
WriteIn;
WriteIn('RMS Error = ',RMS)
End;

Begin (Main Program)
GetData;
Constants;
Coefficients;
Approximation;
Error;
Print
End.
APPENDIX B

(This program is written in UCSD Pascal. It uses the Gram–Schmidt orthogonalization process to determine a set of orthogonal polynomials which are used to determine the least squares approximation of a function whose values are known only over a discrete set of data points.)

Program Schmidt;

Uses Transcend;

Const
M=4;
N=9;

Var
RMS:Real;
DATA:Text;
I,J,K:Integer;
X:Array[1..N] of Real;
F:Array[1..N] of Real;
R:Array[1..N] of Real;
C:Array[0..M] of Real;
D:Array[0..M] of Real;
S:Array[0..M] of Real;
B:Array[1..M,0..M] of Real;
P:Array[0..M,1..N] of Real;
Q:Array[0..M,0..M] of Real;

Procedure Getdata;

(This section reads the required empirical data.)

Begin
Reset(DATA,'DATA.TEXT');
For I:=1 to N do
Read(DATA,X[I],F[I])
End;

Function Power(X:Real;R:Integer):Real;

Begin
If R=0 then
Power:=1
Else if R=1 then
Power:=X
Else
Power:=X*Power(X,R-1)
End:
Procedure Coefficients;
{This section calculates the coefficients needed to generate the }
{set of orthogonal polynomials.}
Begin
CCO[0]:=0; S[0]:=N;
For I:=1 to N do
Begin
PC[0,I]:=1;
CCO[I]:=CCO[I] + F[I]
End;
CCO[N]:=CCO[N]/S[0];
For J:=1 to M do
Begin
For K:=0 to J-1 do
Begin
BC[0,K]:=0;
For I:=1 to N do
BC[0,K]:=BC[0,K] + Power(X[I],J)*P[K,I];
BC[0,K]:=BC[0,K]/S[K];
End;
S[J]:=0;
For I:=1 to N do
Begin
P[J,I]:=Power(X[I],J);
For K:=0 to J-1 do
P[J,I]:=P[J,I] - BC[0,K]*P[K,I];
S[J]:=S[J] + Sqr(P[J,I])
End;
End;
Procedure Polynomials;
{This section calculates the set of orthogonal polynomials.}
Begin
Q[0,0]:=1;
For I:=1 to M do
Q[0,I]:=0;
For J:=1 to M do
  Begin
    For I:=0 to M do
      Q[J,I]:=0;
    Q[J,J]:=1;
    For K:=0 to J-1 do
      For I:=0 to M do
        Q[J,I]:=Q[J,I] - B[J,K]*Q[K,I];
  End;

Procedure Approximation;
{This section calculates the actual least squares approximation.}
  Begin
    For I:=0 to M do
      D[I]:=0;
    For I:=0 to M do
      For J:=0 to M do
        D[I]:=D[I] + C[J]*Q[I,J];
  End;

Procedure Error;
{This section calculates the RMS error.}
  Begin
    For I:=1 to N do
      Begin
        R[I]:=F[I];
        For J:=0 to M do
          R[I]:=R[I] - C[J]*P[I,J];
      End;
    RMS:=0;
    For I:=1 to N do
      RMS:=RMS+Sqr(R[I]);
    RMS:=Sqrt(RMS/(N-M-1));
  End;

Procedure Print;
{This section prints the solution to the least squares approximation problem.}
  Begin
    For I:=0 to M do
      WriteN('D(',I,') = ',D[I]:10:5);
    WriteN('RMS Error = ',RMS);
  End;
Begin (Main Program)
  Getdata;
  Coefficient;
  Polynomials;
  Approximation;
  Error;
  Print
End.
This program is written in UCSD Pascal. It uses the Forsythe recurrence relation to determine a set of orthogonal polynomials which are used to determine the least squares approximation of a function whose values are known only over a discrete set of data points.

Program Forsythe;
Uses Transcend;
Const
M=4;
N=9;
Var
RMS:Real;
DATA:Text;
I,J:Integer;
X:Array[1..N] of Real;
F:Array[1..N] of Real;
R:Array[1..N] of Real;
A:Array[0..M] of Real;
B:Array[0..M] of Real;
C:Array[0..M] of Real;
D:Array[0..M] of Real;
P:Array[-1..M,1..N] of Real;
Q:Array[-1..M,0..M] of Real;

Procedure Getdata;
(This section reads the required empirical data.)
Begin
Reset(DATA,'DATA.TEXT');
For I:=1 to N do
  Read(DATA,X[I],F[I])
End;

Procedure Coefficients;
(This section calculates the coefficients needed to generate the set of orthogonal polynomials.)
Var
S1,S2,S3,S4,S5,S6:Real;
Begin
For I:=1 to N do
Begin
PC-1,IJ:=O;
P(+1,1):=1
End;
AC[O]:=O; B(O):=O;
For J:=0 to M do
Begin
S1:=0; S2:=0; S3:=0;
S4:=0; S5:=0; S6:=0;
{Calculate C[J]}
For I:=1 to N do
Begin
S1:=S1 + F[I]*P[J,I];
S2:=S2 + Sqr(P[J,I])
End;
C[J]:=S1/S2;
If J=M then exit(Coefficients);
{Calculate A[J+1]}
For I:=1 to N do
Begin
S3:=S3 + X[I]*Sqr(P[J,I]);
S4:=S4 + Sqr(P[J,I])
End;
A[J+1]:=S3/S4;
{Calculate P[J+1]}
For I:=1 to N do
Begin
End;
{Calculate B[J+1]}
For I:=1 to N do
Begin
S5:=S5 + Sqr(P[J+1,I]);
S6:=S6 + Sqr(P[J,I])
End;
B[J+1]:=S5/S6
End
Procedure Polynomials;
{This section calculates the set of orthogonal polynomials.}
Begin
For I:=0 to M do
Begin
Q[-1,I]:=0;
Q[0,I]:=0
End;
Q[0,0]:=1;

For J:=0 to M-1 do
For I:=0 to M do
If I=0
then Q[J+1,0]:= - A[J+1]*Q[J,0] - B[J]*Q[J-1,0]
End;

Procedure Approximation;
(This section calculates the actual least squares approximation.)

Begin
For I:=0 to M do
D[I]:=0;
For I:=0 to M do
For J:=0 to M do
D[I]:=D[I] + C[J]*Q[I,J]
End;

Procedure Error;
(This section calculates the RMS error.)

Begin
For I:=1 to N do
Begin
R[I]:=F[I];
For J:=0 to M do
R[I]:=R[I] - C[J]*P[I,J]
End;
RMS:=0;
For I:=1 to N do
RMS:=RMS + Sqr(R[I]);
RMS:=Sqrt(RMS/(N-M-1))
End;

Procedure Print;
(This section prints the solution to the least squares approximation problem.)

Begin
For I:=0 to M do
Write("D('",I,"') = ",D[I]:10:5);
Writeln;
Writeln('RMS Error = ', RMS)
End;

Begin (Main program)
Getdata;
Coefficients;
Polynomials;
Approximation;
Error;
Print
End.
APPENDIX D
Sample Solutions

To illustrate the concepts developed in this paper, let us derive three least squares approximations using the preceding computer programs. Suppose that the true function is

\[ f(x) = x^4 + 3x^3 + 2x^2 + x + 5. \]  \hspace{1cm} (D-1)

Hence, if we add a random error between -0.0005 and 0.0005 to the corresponding true values of this function, we are able to generate the following empirical data:

<table>
<thead>
<tr>
<th>( x_i )</th>
<th>.1</th>
<th>.2</th>
<th>.3</th>
<th>.4</th>
<th>.5</th>
<th>.6</th>
<th>.7</th>
<th>.8</th>
<th>.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(x_i) )</td>
<td>5.1234</td>
<td>5.3057</td>
<td>5.5687</td>
<td>5.9378</td>
<td>6.4370</td>
<td>7.0978</td>
<td>7.9493</td>
<td>9.0253</td>
<td>10.3627</td>
</tr>
</tbody>
</table>

(For comparison purposes, the above data was made identical to the data used by Ralston [7].)

**Gram Polynomials**

Using the set of orthogonal polynomials given in (3-25) with the above data, we can write

\[ p_0(t,9) = 1, \]  \hspace{1cm} (D-2)

\[ p_1(t,9) = (4 - t)/4, \]

\[ p_2(t,9) = (28 - 24t + 3t^2)/28, \]

\[ p_3(t,9) = (84 - 181t + 60t^2 - 5t^3)/84, \]

\[ p_4(t,9) = (168 - 872t + 557t^2 - 112t^3 + 7t^4)/168, \]
where \( t = (x - x_1)/0.1 \). We can use these polynomials to find the values in Table 2. Let

\[
s_j = \sum_{t=0}^{s} [p_j(t,9)]^2 \quad \text{for} \quad j = 0, \ldots, 4.
\]  

**Table 2**

The Values of \( p_j(t,9) \)

<table>
<thead>
<tr>
<th>t</th>
<th>( p_0(t,9) )</th>
<th>( p_1(t,9) )</th>
<th>( p_2(t,9) )</th>
<th>( p_3(t,9) )</th>
<th>( p_4(t,9) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1/4</td>
<td>1</td>
<td>1/2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1/2</td>
<td>-2/7</td>
<td>-3/2</td>
<td>-13/14</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1/4</td>
<td>-17/28</td>
<td>-9/14</td>
<td>9/14</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>-5/7</td>
<td>9/7</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>-1/4</td>
<td>-17/28</td>
<td>9/14</td>
<td>9/14</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>-1/2</td>
<td>-2/7</td>
<td>13/14</td>
<td>-11/14</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>-3/4</td>
<td>1/4</td>
<td>1/2</td>
<td>-3/2</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
</tbody>
</table>

Letting

\[
b_j = \sum_{t=0}^{s} f(t)p_j(t,9) \quad \text{and} \\
\]

\[
c_j = \sum_{t=0}^{s} f(t)p_j(t,9)/ \sum_{t=0}^{s} [p_j(t,9)]^2 = b_j/s_j,
\]

and using the values in Table 2, we have

\[
b_0 = 62.8077, \quad c_0 = 6.9786, \\
b_1 = -9.5093, \quad c_1 = -2.5358, \\
b_2 = 2.6942, \quad c_2 = 0.7620, \\
b_3 = -0.4232, \quad c_3 = -0.0838,
\]
Thus, using the results given in (D-2) and (D-5) and equation (2-1), we obtain the following approximation

\[ f(t) = 0.001t^4 + 0.0034t^3 + 0.0297t^2 + 0.1489t + 5.1234 \quad (D-6) \]

or

\[ f(x) = 0.9988x^4 + 2.9900x^3 + 2.0171x^2 + 0.9920x + 5.0009. \quad (D-7) \]

The RMS error for this approximation is \( 0.0035 \).

**Gram-Schmidt Orthogonalization Process**

Using equations (3-27) and (3-30) where \( \{q_j(x)\} \) is \( 1, x, x^2, \ldots, x^m \), we obtain the values in Table 3.

**Table 3**

<table>
<thead>
<tr>
<th>( r )</th>
<th>( b_1r )</th>
<th>( b_2r )</th>
<th>( b_3r )</th>
<th>( b_4r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.5000</td>
<td>0.3167</td>
<td>0.2250</td>
<td>0.1704</td>
</tr>
<tr>
<td>1</td>
<td>1.0000</td>
<td>1.0000</td>
<td>0.8680</td>
<td>0.7360</td>
</tr>
<tr>
<td>2</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.5000</td>
<td>1.6643</td>
</tr>
<tr>
<td>3</td>
<td>1.0000</td>
<td>1.0000</td>
<td>2.0000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( p_0(x_1) )</td>
<td>( p_1(x_1) )</td>
<td>( p_2(x_1) )</td>
<td>( p_3(x_1) )</td>
</tr>
<tr>
<td>1</td>
<td>1.0000</td>
<td>-0.4000</td>
<td>0.0933</td>
<td>-0.0168</td>
</tr>
<tr>
<td>2</td>
<td>1.0000</td>
<td>-0.3000</td>
<td>0.0233</td>
<td>0.0084</td>
</tr>
<tr>
<td>3</td>
<td>1.0000</td>
<td>-0.2000</td>
<td>-0.0267</td>
<td>0.0156</td>
</tr>
<tr>
<td>4</td>
<td>1.0000</td>
<td>-0.1000</td>
<td>-0.0567</td>
<td>0.0108</td>
</tr>
<tr>
<td>5</td>
<td>1.0000</td>
<td>0.0000</td>
<td>-0.0667</td>
<td>0.0000</td>
</tr>
<tr>
<td>6</td>
<td>1.0000</td>
<td>0.1000</td>
<td>-0.0567</td>
<td>-0.0108</td>
</tr>
<tr>
<td>7</td>
<td>1.0000</td>
<td>0.2000</td>
<td>-0.0267</td>
<td>-0.0156</td>
</tr>
<tr>
<td>8</td>
<td>1.0000</td>
<td>0.3000</td>
<td>0.0233</td>
<td>-0.0084</td>
</tr>
<tr>
<td>9</td>
<td>1.0000</td>
<td>0.4000</td>
<td>0.0933</td>
<td>0.0168</td>
</tr>
</tbody>
</table>
And using the values in Table 3 and equation (3-5), we have

\[c_0 = 6.9786, \quad (D-8)\]
\[c_1 = 6.3395,\]
\[c_2 = 8.1643,\]
\[c_3 = 4.9889,\]
\[c_4 = 0.9414.\]

Furthermore, we can use the values of \(b_{jr}\) in Table 3 and equation (3-27) to derive the following set of orthogonal polynomials:

\[p_0(x) = 1, \quad (D-9)\]
\[p_1(x) = x - 0.5000,\]
\[p_2(x) = x^2 - 1.0000x + 0.1833,\]
\[p_3(x) = x^3 - 1.5000x^2 + 0.6320x - 0.0660,\]
\[p_4(x) = x^4 - 2.0002x^3 + 1.3359x - 0.3358x + 0.0245.\]

Thus, using the results given in (D-8) and (D-9) and equation (2-1), we obtain the following approximation

\[f(x) = 0.9414x^4 + 3.1061x^3 + 1.939x^2 + 1.0121x + 4.9995 \quad (D-10)\]

The RMS error for this approximation is .00041.

**Forsythe Recurrence Relation**

Using equations (3-31), (3-43), and (3-45), we obtain the values in Table 4.
Table 4

The Values of $p_j(x_1)$, $a_j$, and $b_j$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$p_0(x_1)$</th>
<th>$p_1(x_1)$</th>
<th>$p_2(x_1)$</th>
<th>$p_3(x_1)$</th>
<th>$p_4(x_1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000</td>
<td>-0.4000</td>
<td>0.0933</td>
<td>-0.0168</td>
<td>0.0024</td>
</tr>
<tr>
<td>2</td>
<td>1.0000</td>
<td>-0.3000</td>
<td>0.0233</td>
<td>0.0084</td>
<td>-0.0036</td>
</tr>
<tr>
<td>3</td>
<td>1.0000</td>
<td>-0.2000</td>
<td>-0.0267</td>
<td>0.0156</td>
<td>-0.0019</td>
</tr>
<tr>
<td>4</td>
<td>1.0000</td>
<td>-0.1000</td>
<td>-0.0567</td>
<td>-0.0108</td>
<td>0.0015</td>
</tr>
<tr>
<td>5</td>
<td>1.0000</td>
<td>0.0000</td>
<td>0.00667</td>
<td>0.0000</td>
<td>0.0031</td>
</tr>
<tr>
<td>6</td>
<td>1.0000</td>
<td>0.0000</td>
<td>-0.0567</td>
<td>-0.0108</td>
<td>0.0015</td>
</tr>
<tr>
<td>7</td>
<td>1.0000</td>
<td>0.2000</td>
<td>-0.0267</td>
<td>-0.1560</td>
<td>-0.0019</td>
</tr>
<tr>
<td>8</td>
<td>1.0000</td>
<td>0.3000</td>
<td>0.0233</td>
<td>-0.0084</td>
<td>-0.0036</td>
</tr>
<tr>
<td>9</td>
<td>1.0000</td>
<td>0.4000</td>
<td>0.0933</td>
<td>0.0168</td>
<td>0.0024</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$j$</th>
<th>$a_j$</th>
<th>$b_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>1</td>
<td>0.5000</td>
<td>0.6667</td>
</tr>
<tr>
<td>2</td>
<td>0.5000</td>
<td>0.0513</td>
</tr>
<tr>
<td>3</td>
<td>0.5000</td>
<td>0.0463</td>
</tr>
<tr>
<td>4</td>
<td>0.5000</td>
<td>0.0413</td>
</tr>
</tbody>
</table>

And using the values in Table 4 and equation (3-5), we have

\[
c_0 = 6.9786, \quad \text{(D-11)}
\]
\[
c_1 = 6.3395, \quad c_2 = 8.1644, \quad c_3 = 4.9875, \quad c_4 = 0.9985.
\]

Furthermore, using the values of $a_j$ and $b_j$ in Table 4, we can derive the following set of orthogonal polynomials:

\[
p_0(x) = 1, \quad \text{(D-12)}
\]
\[
p_1(x) = x - 0.5000, \quad p_2(x) = x^2 - 1.0000x + 0.18333,
\]
\[
p_3(x) = x^3 - 1.5000x^2 + 0.6320x - 0.0660, \quad p_4(x) = x^4 - 2.0000x^3 + 1.3357x^2 - 0.3357x + 0.02451.
\]
Thus, using the results given in (D-11) and (D-12) and equation (2-1), we obtain the following approximation

\[
f(x) = 0.9985x^4 + 2.9904x^3 + 2.0169x^2 + 0.9920x + 5.0001.
\]  
(D-13)

The RMS error for this approximation is 0.00035.

**Conclusions**

All of the above approximations are basically the same; however, there are some differences which might be puzzling to the casual reader. Observe that the values of \( p_j(x_1) \) listed in Table 3 seem to be identical to the values of \( p_j(x_1) \) listed in Table 4, but due to different round off errors which the computer makes in calculating these values, the actual values that are obtained by the computer are slightly different. (The computer used to calculate these values maintains six significant figures, and not all of these significant figures are shown in the tables.) This difference becomes apparent when the computer uses these values to calculate the coefficients given in (D-8) and (D-11) and obtains strikingly different coefficients.

In deriving the above approximations on an Apple II computer, it took 5.8 seconds to run program Gram, 9.8 seconds to run program Schmidt, and 8.2 seconds to run program Forsythe. The fastest program was program Gram; however, this program is not general. The values of \( p_j(t,9) \) in Table 2 were calculated by hand and placed in the program. Thus, the usefulness of program Gram is greatly limited. On the other hand, program Schmidt and program Forsythe are general and can be used in many different situations, but program Forsythe is
the more efficient method for general use.