The chess-board is the world; the pieces are the phenomena of the universe; the rules of the game are what we call the laws of Nature. The player on the other side is hidden from us. We know that his play is always fair, just, and patient. But also we know, to our cost, that he never overlooks a mistake, or makes the smallest allowance for ignorance.

— T.H. HUXLEY

Chapter 5

Functional approximations

In previous chapters we have developed algorithms for solving systems of linear and nonlinear *algebraic* equations. Before we undertake the development of algorithms for *differential* equations, we need to develop some basic concepts of *functional approximations*. In this respect the present chapter is a bridge between the realms of *lumped parameter* models and *distributed* and/or *dynamic* models.

There are at least two kinds of *functional approximation* problems that we encounter frequently. In the first class of problem, a known function $f(x)$ is approximated by another function, $P_n(x)$ for reasons of computational necessity or expediency. As modelers of physical phenomena, we often encounter a second class of problem in which there is a need to represent an experimentally observed, discrete set of data of the form $\{x_i, f_i | i = 1, \cdots n\}$ as a function of the form $f(x)$ over the domain of the independent variable x.

5.1 Approximate representation of functions

5.1.1 Series expansion

As an example of the first class of problem, consider the evaluation of the *error function* given by,

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-\xi^2} \, d\xi$$

Since the integral does not have a closed form expression, we have to use a series expansion for,

$$e^{-\xi^2} = \sum_{k=0}^{\infty} \frac{(-1)^k \xi^{2k}}{k!}$$

Note that this expansion is around $\xi = 0$. We can integrate the series expansion term-by-term to obtain,

$$erf(x) = \frac{2}{\sqrt{\pi}} \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k+1}}{(2k+1)k!}$$

We can now choose to approximate this function as,

$$erf(x) \approx P_{2n+1}(x) = \frac{2}{\sqrt{\pi}} \sum_{k=0}^{n} \frac{(-1)^k x^{2k+1}}{(2k+1)k!} + R(x)$$

by truncating the infinite series to n terms. The error introduced by truncating such a series is called the *truncation error* and the magnitude of the *residual function*, $R(x)$ represents the magnitude of the truncation error. For x close to zero a few terms of the series (small n) are adequate. The convergence of the series is demonstrated in Table 5.1. It is clear that as we go farther away from $x = 0$, more terms are required for $P_{2n+1}(x)$ to represent $erf(x)$ accurately. The error distribution, defined as $\epsilon(x,n) := |erf(x) - P_{2n+1}(x)|$, is shown in figure 5.1. It is clear from figure 5.1a, that for a fixed number of terms, say $n = 8$, the error increases with increasing values of x. For larger values of x, more terms are required to keep the error small. For $x = 2.0$, more than 10 terms are required to get the error under control.

5.1.2 Polynomial approximation

In the above example we chose to construct an approximate function to represent $f(x) = erf(x)$ by expanding $f(x)$ in Taylor series around $x =$

n	$P_{2n+1}(x = 0.5)$	$P_{2n+1}(x = 1.0)$	$P_{2n+1}(x = 2.0)$
2	0.5207	0.865091	2.85856
4	0.5205	0.843449	2.09437
6	0.5205	0.842714	1.33124
8	0.5205	0.842701	1.05793
10	0.5205	0.842701	1.00318
20	0.5205	0.842701	0.995322
Exact	0.5205	0.842701	0.995322

Table 5.1: Convergence of $P_{2n+1}(x)$ to $erf(x)$ at selected values of x

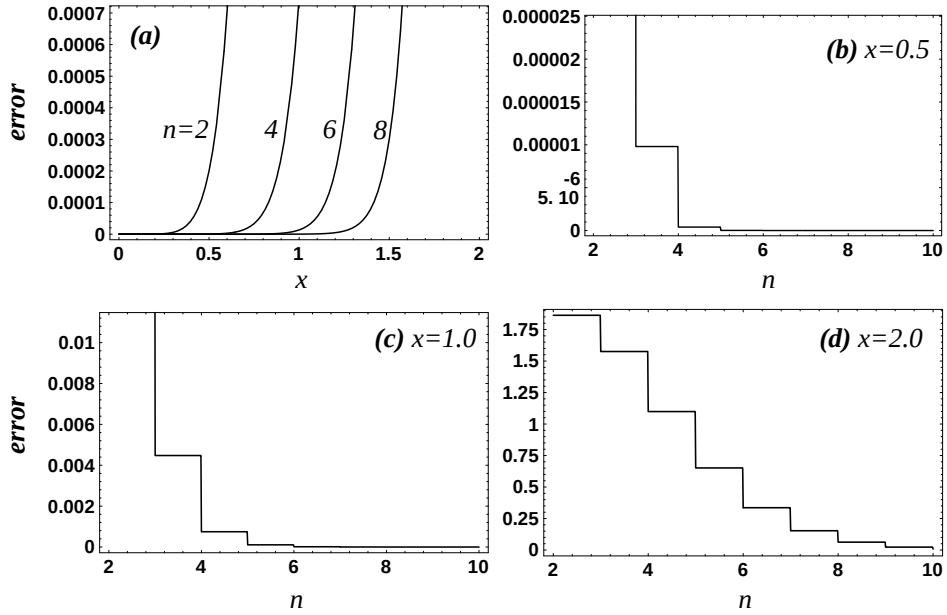


Figure 5.1: Error distribution of $\epsilon(x,n) := |erf(x) - P_{2n+1}(x)|$ for different levels of truncation

0. This required that all the higher order derivative be available at $x = 0$. Also, since the expansion was around $x = 0$, the approximation fails increasingly as x moves away from zero. In another kind of functional approximation we can attempt to get a good representation of a given function $f(x)$ over a range $x \in [a, b]$. We do this by choosing a set of n basis functions, $\{\phi_i(x)|i = 1 \cdots n\}$ that are linearly independent and representing the approximation as,

$$f(x) \approx P_n(x) = \sum_{i=1}^{n} a_i \phi_i(x)$$

Here the basis functions $\phi_i(x)$ are known functions, chosen with care to form a linearly independent set and a_i are unknown constants that are to be determined in such a way that we can make $P_n(x)$ as good an approximation to $f(x)$ as possible - *i.e.*, we can define an error as the difference between the exact function and the approximate representation,

$$\epsilon(x; a_i) = |f(x) - P_n(x)|$$

and device a scheme to select a_i such that the error is minimized.

Example

So far we have outlined certain general concepts, but left open the choice of a specific basis functions $\phi_i(x)$, the definition of the norm $|.|$ in the error or the minimization procedure to get a_i.

Let the basis functions be

$$\phi_i(x) = x^{i-1} \qquad i = 1, \cdots n$$

which, incidentally is a poor choice, but one that is easy to understand. Hence the approximate function will be a polynomial of degree $(n-1)$ of the form,

$$P_{n-1}(x) = \sum_{i=1}^{n} a_i x^{i-1}$$

Next, let us introduce the idea of *collocation* to evaluate the error at n selected points in the range of interest $x \in [a, b]$. We choose n points $\{x_k|k = 1, \cdots n\}$ because we have introduced n degrees of freedom (unknowns) in a_i. A naive choice would be to space these collocation points equally in the interval $[a, b]$ - *i.e.*,

$$x_k = a + (k-1)\frac{(b-a)}{(n-1)} \qquad k = 1, \cdots, n$$

Finally we can require the error at these points to be exactly equal to zero - *i.e.*,

$$\epsilon(x_k; a_i) = f(x_k) - P_{n-1}(x_k) = 0$$

or

$$\sum_{i=1}^{n} a_i x_k^{i-1} = f(x_k) \qquad k = 1, \cdots, n \tag{5.1}$$

which yields n linear equations in n unknowns a_i. This can be written in matrix form

$$\boldsymbol{P}\boldsymbol{a} = \boldsymbol{f}$$

where the elements of matrix $\boldsymbol{P}$ are given by, $P_{k,i} = x_k^{i-1}$ and the vectors are $\boldsymbol{a} = [a_1, \cdots, a_n]$ and $\boldsymbol{f} = [f(x_1), \cdots, f(x_n)]$. Thus we have reduced the functional approximation problem to one of solving a system of linear algebraic equations and tools of chapter 3 become useful!

Let us be even more specific now and focus on approximating the error function $f(x) = erf(x)$ over the interval $x \in [0.1, 0.5]$. Let us also choose $n = 5$ - *i.e.*, a quartic polynomial. This will allow us to write out the final steps of the approximation problem explicitly. The equally spaced collocation points are,

$$x_k = \{0.1, 0.2, 0.3, 0.4, 0.5\}$$

and the error function values at the collocation points are

$$\boldsymbol{f} = f(x_k) = [0.1125, 0.2227, 0.3286, 0.4284, 0.5205]$$

Thus, equation (5.1) yields the following system

$$\boldsymbol{P} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 & x_1^4 \\ 1 & x_2 & x_2^2 & x_2^3 & x_2^4 \\ 1 & x_3 & x_3^2 & x_3^3 & x_3^4 \\ 1 & x_4 & x_4^2 & x_4^3 & x_4^4 \\ 1 & x_5 & x_5^2 & x_5^3 & x_5^4 \end{bmatrix} = \begin{bmatrix} 1.0 & 0.10 & 0.010 & 0.0010 & 0.0001 \\ 1.0 & 0.20 & 0.040 & 0.0080 & 0.0016 \\ 1.0 & 0.30 & 0.090 & 0.0270 & 0.0081 \\ 1.0 & 0.40 & 0.160 & 0.0640 & 0.0256 \\ 1.0 & 0.50 & 0.250 & 0.1250 & 0.0625 \end{bmatrix}$$

Solution of the linear system yields the unknown coefficients as

$$\boldsymbol{a} = \{0.0001, 1.1262, 0.0186, -0.4503, 0.1432\}$$

A MATLAB function that shows the implementation of the above procedure for a specified degree of polynomial n is given in figure 5.2. Recall that we had made a comment earlier that the basis function $\phi_i(x) = x^{i-1} \qquad i = 1, \cdots n$ is a poor choice. We can understand why this is so,

```
function a=erf_apprx(n)
%  Illustration functional (polynomial) approximation
%  fits error function in (0.1, 0.5) to a
%  polynomial of degree n

%define interval
a = 0.1;   b=0.5;

%pick collocation points
x=a + [0:(n-1)] *(b-a)/(n-1);

%Calculate the error function at collocation points
f=erf(x);   %Note that erf is a MATLAB function

%Calculate the matrix
for k=1:n
 P(k,:) = x(k).^[0:n-1];
end

%Print the determinant of P
fprintf(1,'Det. of P for deg. %2i is = %12.5e\n', n,det(P) );

%Determine the unknown coefficients a_i
a=P\f';
```

Figure 5.2: MATLAB implementation illustrating steps of functional approximation

by using the function shown in figure 5.2 for increasing degree of polynomials. The matrix $\boldsymbol{P}$ becomes poorly scaled and nearly singular with increasing degree of polynomial as evidence by computing the determinant of $\boldsymbol{P}$. For example the determinant of $\boldsymbol{P}$ is 1.60000×10^{-2} for $n = 3$ and it goes down rapidly to 1.21597×10^{-12} for $n = 6$. Selecting certain orthogonal polynomials such as Chebyshev polynomials and using the roots of such polynomials as the collocation points results in well conditioned matrices and improved accuracy. More on this in section §5.8.

Note that MATLAB has a function called `polyfit(x,y,n)` that will accept a set of pairwise data $\{x_k, y_k = f(x_k) \mid k = 1, \cdots, m\}$ and produce a polynomial fit of degree n (which can be different from m) using a least-squares minimization. Try using the function `polyfit(x,y,n)` for the above example and compare the polynomial coefficients $\boldsymbol{a}$ produced by the two approaches.

```
»x=[0.1:0.1:0.5]  % Define Collocation Points
»y=erf(x)         % Calculate the function at Collocation Points
»a=polyfit(x,y,4)% Fit 4th degree polynomial. Coefficients in a
»polyval(a,x)     % Evaluate the polynomial at collocation pts.
»erf(x)           % Compare with exact values at the same pts.
```

5.2 Approximate representation of data

The concepts of polynomial approximation were discussed in section §5.1.2 in the context of constructing approximate representations of complicated functions (such as the error function). We will develop and apply these ideas further in later chapters for solving differential equations. Let us briefly explore the problem of constructing approximate functions for representing a discrete set of m pairs of data points

$$\{(x_k, f_k) \mid k = 1, \cdots, m\}$$

gathered typically from experiments. As an example, let us consider the saturation temperature vs. pressure data taken from steam tables and shown in Table 5.2. Here the functional form that we wish to construct is to represent pressure as a function of temperature, $P(T)$ over the temperature range $T \in [220, 232]$. A number of choices present themselves.

$T(^oF)$	$P(psia)$
220.0000	17.1860
224.0000	18.5560
228.0000	20.0150
232.0000	21.5670

Table 5.2: Saturation temperature vs. pressure from steam tables

- We can choose to fit a cubic polynomial, $P_3(T)$ that will pass through each of the four data points over the temperature range $T \in [220, 232]$. This will be considered as a *global polynomial* as it covers the entire range of interest in T.

- Alternately we can choose to construct *piecewise polynomials* of a lower degree with a limited range of applicability. For example, we can take the first three data points and fit a quadratic polynomial, and the last three points and fit a different quadratic polynomial.

- As a third alternative, we can choose to fit a global polynomial of degree less than three, that will not pass through any of the given data points, but will produce a function that *minimizes* the error over the entire range of $T \in [220, 232]$.

The procedures developed in section §5.1.2 are directly applicable to the first two choices and hence they warrant no further discussion. Hence we develop the algorithm only the third choice dealing with the least-squares minimization concept.

5.2.1 Least squares approximation

Suppose there are m independent experimental observations ($m = 4$ in the above example) and we wish to fit a global polynomial of degree n ($n < m$) we define the error at every observation point as,

$$\epsilon_k = (P_{n-1}(x_k) - f_k) \qquad k = 1, \cdots, m$$

The basis functions are still the set, $\{x^{i-1} \mid i = 1, \cdots n\}$ and the polynomial is

$$P_{n-1}(x) = \sum_{i=1}^{n} a_i x^{i-1}$$

Here a_i are the unknowns that we wish to determine. Next we construct an objective function which is the sum of squares of the error at every observation point - viz.

$$J(\boldsymbol{a}) = \frac{\sum_{k=1}^{m} \epsilon_k^2}{m} = \frac{\sum_{k=1}^{m} (P_{n-1}(x_k) - f_k)^2}{m} = \frac{\sum_{k=1}^{m} (\sum_{i=1}^{n} a_i x_k^{i-1} - f_k)^2}{m}$$

The scalar objective function $J(\boldsymbol{a})$ is a function of n unknowns a_i. From elemetary calculus, the condition for the function $J(\boldsymbol{a})$ to have a minimum is,

$$\frac{\partial J(\boldsymbol{a})}{\partial \boldsymbol{a}} = 0$$

This condition provides n linear equations of the form $\boldsymbol{Pa} = \boldsymbol{b}$ that can be solved to obtain $\boldsymbol{a}$. The expanded form of the equations are,

$$\begin{bmatrix} \sum_{k=1}^{m} 1 & \sum_{k=1}^{m} x_k & \sum_{k=1}^{m} x_k^2 & \cdots & \sum_{k=1}^{m} x_k^{n-1} \\ \sum_{k=1}^{m} x_k & \sum_{k=1}^{m} x_k^2 & \sum_{k=1}^{m} x_k^3 & \cdots & \sum_{k=1}^{m} x_k^{n} \\ \sum_{k=1}^{m} x_k^2 & \sum_{k=1}^{m} x_k^3 & \sum_{k=1}^{m} x_k^4 & & \sum_{k=1}^{m} x_k^{n+1} \\ \vdots & \vdots & & \ddots & \vdots \\ \sum_{k=1}^{m} x_k^{n-1} & \sum_{k=1}^{m} x_k^{n} & \sum_{k=1}^{m} x_k^{n+1} & \cdots & \sum_{k=1}^{m} x_k^{2(n-1)} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{m} f_k \\ \sum_{k=1}^{m} f_k x_k \\ \sum_{k=1}^{m} f_k x_k^2 \\ \vdots \\ \sum_{k=1}^{m} f_k x_k^{n-1} \end{bmatrix}$$

Observe that the equations are not only linear, but the matrix is also symmetric. Work through the following example using MATLAB to generate a quadratic, least-squares fit for the data shown in Table 5.2. Make sure that you understand what is being done at each stage of the calcualtion. This example illustrates a cubic fit that passes through each of the four data points, followed by use of the cubic fit to interpolate data at intermediate temperatures of $T = [222, 226, 230]$. In the last part the least squares solution is obtained using the procedure developed in this section. Finally MATLAB's `polyfit` is used to generate the same least squares solution!

```
»x=[220,224,228,232]              % Define temperatures
»f=[17.186,18.556,20.015,21.567] % Define pressures
»a3=polyfit(x,f,3)                % Fit a cubic. Coefficients in a3
»polyval(a3,x)                    % Check cubic passes through pts.
»xi=[222,226,230]                 % Define interpolation points
»polyval(a3,xi)                   % Evaluate at interpolation pts.
»%get ready for least square solution!
»x2=x.^2                          % Evaluate x^2
»x3=x.^3                          % Evaluate x^3
»x4=x.^4                          % Evaluate x^4
```

```
»P=[4,sum(x),sum(x2); ...          % Define matrix P over next 3 lines
»sum(x), sum(x2), sum(x3); ...
»sum(x2), sum(x3), sum(x4) ]
»b=[sum(f), f*x', f*x2']           % Define right hand side
»a = P\b'                          % ans: (82.0202,-0.9203,0.0028)
»c=polyfit(x,f,2)                  % Let MATLAB do it! compare c & a
»norm(f-polyval(a3,x))             % error in cubic fit 3.3516 × 10^-14
»norm(f-polyval(c,x))              % error in least squares fit 8.9443 × 10^-4
```

5.3 Difference operators

In the previous sections we developed polynomial approximation schemes in such a way that they required a solution of a system of linear algebraic equation. For uniformly spaced data, introduction of *difference operators* and difference tables, allows us to solve the same polynomial approximation problem in a more elegant manner without the need for solving s system of algebraic equations. This difference operator approach also lends itself naturally to recursive construction of higher degree polynomials with very little additional computation as well as extension to numerical differentiation and integration of discrete set of data.

Consider the set of data $\{(x_i, f_i) \mid i = 1, \cdots, m\}$ where the independent variable, x is varied uniformly generating an equally spaced data - *i.e.*,

$$x_{i+1} = x_i + h, \qquad i = 1, \cdots m \quad \text{or} \quad x_i = x_1 + (i-1)h$$

The forward difference operator, as introduced already in section §2.8, is defined by,

Forward difference operator

$$\Delta f_i = f_{i+1} - f_i \tag{5.2}$$

In a similar manner we can define a backward difference, central difference and shift operators as shown below.

Backward difference operator

$$\nabla f_i = f_i - f_{i-1} \tag{5.3}$$

Central difference operator

$$\delta f_i = f_{i+1/2} - f_{i-1/2} \tag{5.4}$$

Shift operator

$$Ef_i = f_{i+1} \tag{5.5}$$

We can also add the differential operator to the above list.

Differential operator

$$Df(x) = \frac{df(x)}{dx} = f'(x) \tag{5.6}$$

The difference operators are nothing but rules of calculations, just like a differential operator defines a rule for differentiation. Clearly these rules can be applied repeatedly to obtaind higher order differences. For example a second order forward difference with respect to reference point i is,

$$\Delta^2 f_i = \Delta(\Delta f_i) = \Delta(f_{i+1} - f_i) = f_{i+2} - 2f_{i+1} + f_i$$

5.3.1 Operator algebra

Having introduced some new definitions of operators, we can discover some interesting relationships between various operators such as the following.

$$\Delta f_i = f_{i+1} - f_i \quad \text{and} \quad Ef_i = f_{i+1}$$

Combining these two we can write,

$$\Delta f_i = Ef_i - f_i = (E-1)f_i$$

Since the operand f_i is the same on both sides of the equation, the operators (which define certain rules and hence have certain effects on the operand f_i) have an equivalent effect given by,

$$\Delta = (E-1) \quad \text{or} \quad \boxed{E = (1+\Delta)} \tag{5.7}$$

Equation (5.7) can then be applied on any other operand like f_{i+k}. All of the operators satisfy the distributive, commutative and associative rules of algebra. Also, repeated application of the operation can be represented by,

$$E^\alpha = (1+\Delta)^\alpha$$

Note that $E^\alpha f(x)$ simply implies that the function f is evaluated after shifting the independent variable by α - *i.e.*,

$$E^\alpha f(x) = f(x + \alpha h)$$

Hence α can be an integer or any real number. Similarly, we have

$$\nabla f_i = f_i - f_{i-1} \quad \text{and} \quad E f_{i-1} = f_i \quad \text{and} \quad f_{i-1} = E^{-1} f_i$$

where we have introduce the inverse of the shift operator E to shift backwards. Combining these we can write,

$$\nabla f_i = f_i - E^{-1} f_i = (1 - E^{-1}) f_i$$

Once again recognizing that the operand f_i is the same on both sides of the equation, the operators are related by,

$$\nabla = (1 - E^{-1}) \quad \text{or} \quad E^{-1} = (1 - \nabla) \quad \text{or} \quad \boxed{E = (1 - \nabla)^{-1}} \tag{5.8}$$

Yet another relation between the shift operator E and the differential operator D can be developed by considering the Taylor series expansion of $f(x + h)$,

$$f(x + h) = f(x) + h f'(x) + \frac{h^2}{2!} f''(x) + \cdots$$

which can be written in operator notation as,

$$E f(x) = \left[1 + hD + \frac{h^2 D^2}{2!} + \cdots \right] f(x)$$

The term in square brackets is the exponential function and hence

$$\boxed{E = e^{hD}} \tag{5.9}$$

While such a game of discovering relationships between various operators can be played indefinitely, let us turn to developing some useful algorithms from these.

5.3.2 Newton forward difference approximation

Our objective is to construct a polynomial representation for the discrete set of data $\{(x_i, f_i) \mid i = 1, \cdots, m\}$ using an alternate approach from that of section §5.1.2. Assuming that there is a function $f(x)$ represent-

Is such an assumption always valid?

ing the given data, we can express such a function as,

$$f(x) = P_n(x) + R(x)$$

where $P_n(x)$ is the polynomial approximation to $f(x)$ and $R(x)$ is the residual error. Given a set of m data points we know at least one way (section §5.1.2) to can construct a polynomial of degree $(m-1)$. Now let us use the power of operator algebra to develop an alternate way to construct such a polynomial and in the process, also learn something about the residual function $R(x)$. Applying equation (5.7) repeatedly α time on $f(x)$ we get,

$$E^\alpha f(x) = (1+\Delta)^\alpha f(x)$$

Now for integer values of α the right hand side is the binomial expansion while for any real number, it yields an infinite series. Using such an expansion the above equation can be written as,

$$f(x+\alpha h) = \left[1 + \alpha\Delta + \frac{\alpha(\alpha-1)}{2!}\Delta^2 + \frac{\alpha(\alpha-1)(\alpha-2)}{3!}\Delta^3 + \cdots \right.$$
$$\left. \frac{\alpha(\alpha-1)(\alpha-2)\cdots(\alpha-n+1)}{n!}\Delta^n + \cdots\right] f(x) \quad (5.10)$$

Up to this point in our development we have merely used tricks of operator algebra. We will now make the direct connection to the given, discrete set of data $\{(x_i, f_i) \mid i = 1, \cdots, m\}$. Taking x_1 as the reference point, the transformation

$$x = x_1 + \alpha h$$

makes α the new independent variable and for integer values of $\alpha = 0, 1, \cdots (m-1)$ we retrieve the equally spaced data set $\{x_1, x_2, \cdots x_m\}$ and for non-integer (real) values of α we can reach the other values of $x \in (x_1, x_m)$. Splitting equation (5.10) into two parts,

$$f(x_1+\alpha h) = \left[1 + \alpha\Delta + \frac{\alpha(\alpha-1)}{2!}\Delta^2 + \frac{\alpha(\alpha-1)(\alpha-2)}{3!}\Delta^3 + \cdots \right.$$
$$\left. \frac{\alpha(\alpha-1)(\alpha-2)\cdots(\alpha-m+2)}{(m-1)!}\Delta^{m-1}\right] f(x_1) + R(x)$$

we can recognize the terms in the square brackets as a polynomial of degree $(m-1)$ in the transformed variable α. We still need to determine the numbers $\{\Delta f(x_1), \Delta^2 f(x_1), \cdots, \Delta^{(m-1)} f(x_1)\}$. These can be computed and organized as a forward difference table shown in figure 5.3. Since forward differences are needed for constructing the polynomial, it

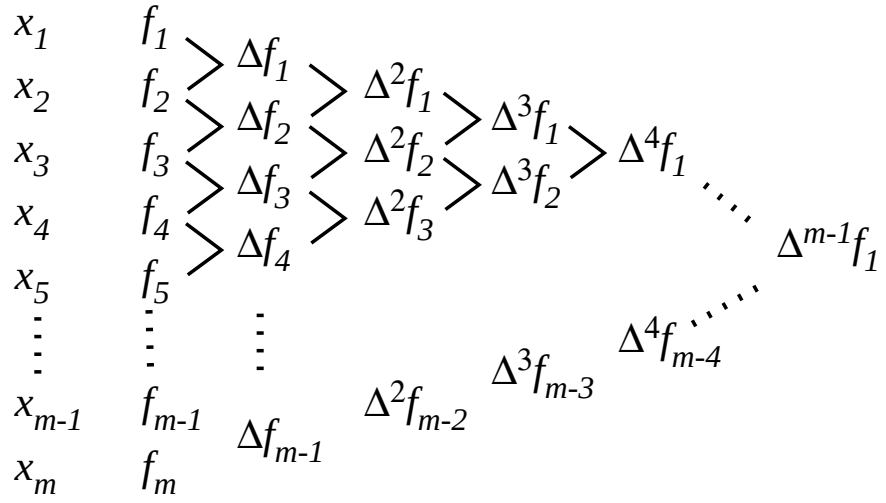


Figure 5.3: Structure of Newton forward difference table for m equally spaced data

is called the Newton forward difference polynomial and it is given by,

$$P_{m-1}(x_1 + \alpha h) = \left[1 + \alpha\Delta + \frac{\alpha(\alpha-1)}{2!}\Delta^2 + \frac{\alpha(\alpha-1)(\alpha-2)}{3!}\Delta^3 + \cdots \right.$$
$$\left. \frac{\alpha(\alpha-1)(\alpha-2)\cdots(\alpha-m+2)}{(m-1)!}\Delta^{m-1}\right] f(x_1) + \mathcal{O}(h^m) \quad (5.11)$$

The polynomial in equation (5.11) will pass through the given data set $\{(x_i, f_i) \mid i = 1, \cdots, m\}$ - *i.e.*, for integer values of $\alpha = 0, 1, \cdots (m-1)$ it will return values of $\{f_1, f_2, \cdots f_m\}$. This implies that the residual function $R(x)$ will have roots at the data points $\{x_i \mid i = 1, \cdots, m\}$. For a polynomial of degree $(m-1)$, shown in equation (5.11), the residual at other values of x is typically represented as $R(x) \approx \mathcal{O}(h^m)$ to suggest that the leading term in the truncated part of the series is of order m.

Example

A set if five ($m = 5$) equally spaced data points and the forward difference table for the data are shown in figure 5.4. For this example, clearly $h = 1$ and $x = x_1 + \alpha$. We can take the reference point as $x_1 = 2$ and construct the following linear, quadratic and cubic polynomials, respectively.

Note that $P_4(2+\alpha) = P_3(2+\alpha)$ for this case! Why?

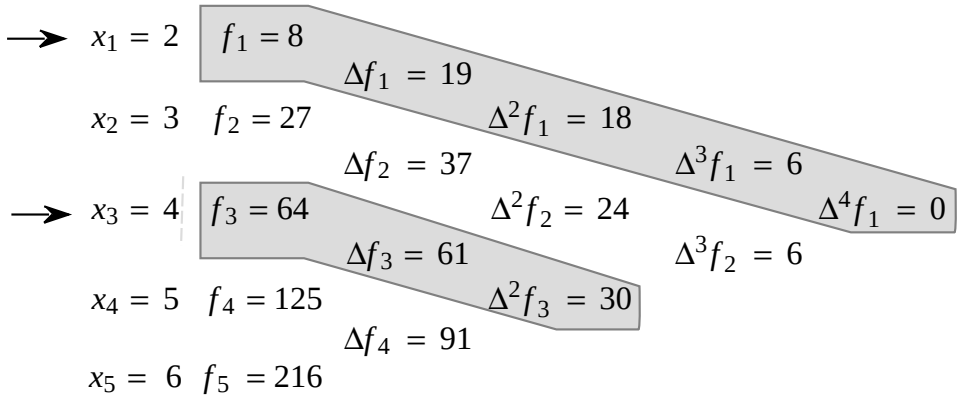


Figure 5.4: Example of a Newton forward difference table

$$
\begin{aligned}
P_1(2+\alpha) &= [1+\alpha\Delta]\, f(x_1) = (8) + \alpha(19) + \mathcal{O}(h^2) \\
P_2(2+\alpha) &= (8) + \alpha(19) + \frac{\alpha(\alpha-1)}{2!}(18) + \mathcal{O}(h^3) \\
P_3(2+\alpha) &= (8) + \alpha(19) + \frac{\alpha(\alpha-1)}{2!}(18) + \frac{\alpha(\alpha-1)(\alpha-2)}{3!}(6) + \mathcal{O}(h^4)
\end{aligned}
$$

You can verify easily that $P_1(2+\alpha)$ passes through $\{x_1, x_2\}$, $P_2(2+\alpha)$ passes through $\{x_1, x_2, x_3\}$ and $P_3(2+\alpha)$ passes through $\{x_1, x_2, x_3, x_4\}$. For finding the interpolated value of $f(x = 3.5)$ for example, first determine the values of α at $x = 3.5$ from the equation $x = x_1 + \alpha h$. It is $\alpha = (3.5 - 2)/1 = 1.5$. Using this value in the cubic polynomial,

$$P_3(2+1.5) = (8) + 1.5(19) + \frac{1.5(0.5)}{2!}(18) + \frac{1.5(0.5)(-0.5)}{3!}(6) = 42.875$$

As another example, by taking $x_3 = 4$ as the reference point we can construct the following quadratic polynomial

$$P_2(4+\alpha) = (64) + \alpha(61) + \frac{\alpha(\alpha-1)}{2!}(30)$$

which will pass through the data set $\{x_3, x_4, x_5\}$. This illustration should show that once the difference talbe is constructed, a variety of polynomials of varying degrees can be constructed quite easily.

5.3.3 Newton backward difference approximation

An equivalent class of polynomials using the *backward difference* operator based on equation (5.8) can be developed. Applying equation (5.8)

repeatedly α times on $f(x)$ we get,

$$E^{\alpha} f(x) = (1 - \nabla)^{-\alpha} f(x)$$

which can be expanded as before to yield,

$$f(x + \alpha h) = \left[1 + \alpha\nabla + \frac{\alpha(\alpha+1)}{2!}\nabla^2 + \frac{\alpha(\alpha+1)(\alpha+2)}{3!}\nabla^3 + \cdots \right.$$
$$\left. \frac{\alpha(\alpha+1)(\alpha+2)\cdots(\alpha+n-1)}{n!}\nabla^n + \cdots\right] f(x) \quad (5.12)$$

As with the Newton forward formula, the above equation (5.12) terminates at a finite number of terms for integer values of α and for non-integer values, it will always be an infinite series which must be truncated, thus sustaining a *trunctaion error*.

In making the precise connection to a given discrete data set $\{(x_i, f_i) \mid i = 0, -1 \cdots, -n\}$, typically the largest value of x (say, x_0) is taken as the reference point. The transformation

$$x = x_0 + \alpha h$$

makes α the new independent variable and for negative integer values of $\alpha = -1, \cdots - n$ we retrieve the equally spaced data set $\{x_{-1}, \cdots x_{-n}\}$ and for non-integer (real) values of α we can reach the other values of $x \in (x_{-n}, x_0)$. Splitting equation (5.12) into two parts,

$$f(x_0 + \alpha h) = \left[1 + \alpha\nabla + \frac{\alpha(\alpha+1)}{2!}\nabla^2 + \frac{\alpha(\alpha+1)(\alpha+2)}{3!}\nabla^3 + \cdots \right.$$
$$\left. \frac{\alpha(\alpha+1)(\alpha+2)\cdots(\alpha+n-1)}{n!}\nabla^n + \cdots\right] f(x_0) + R(x)$$

we can recognize the terms in the square brackets as a polynomial of degree n in the transformed variable α. We still need to determine the numbers $\{\nabla f(x_0), \nabla^2 f(x_0), \cdots, \nabla^n f(x_0)\}$. These can be computed and organized as a backward difference table shown in figure 5.5. Since backward differences are needed for constructing the polynomial, it is called the Newton backward difference polynomial and it is given by,

$$P_n(x_0 + \alpha h) = \left[1 + \alpha\nabla + \frac{\alpha(\alpha+1)}{2!}\nabla^2 + \frac{\alpha(\alpha+1)(\alpha+2)}{3!}\nabla^3 + \cdots \right.$$
$$\left. \frac{\alpha(\alpha+1)(\alpha+2)\cdots(\alpha+n-1)}{n!}\nabla^n\right] f(x_0) + \mathcal{O}(h^{n+1}) \quad (5.13)$$

The polynomial in equation (5.13) will pass through the given data set $\{(x_i, f_i) \mid i = 0, \cdots, -n\}$ - *i.e.*, for integer values of $\alpha = 0, -1, \cdots - n$ it will return values of $\{f_0, f_{-1}, \cdots f_{-n}\}$. At other values of x the residual will be of order $\mathcal{O}(h^{n+1})$.

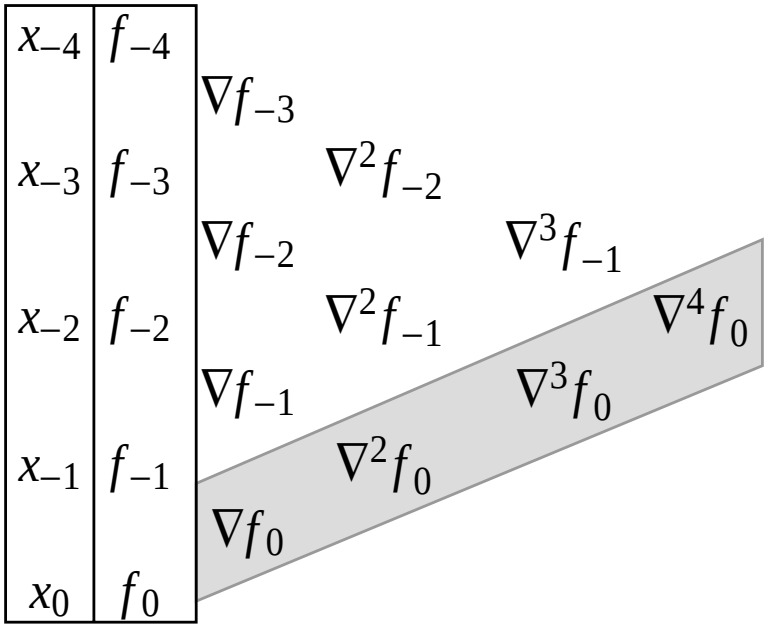


Figure 5.5: Structure of Newton backward difference table for 5 equally spaced data

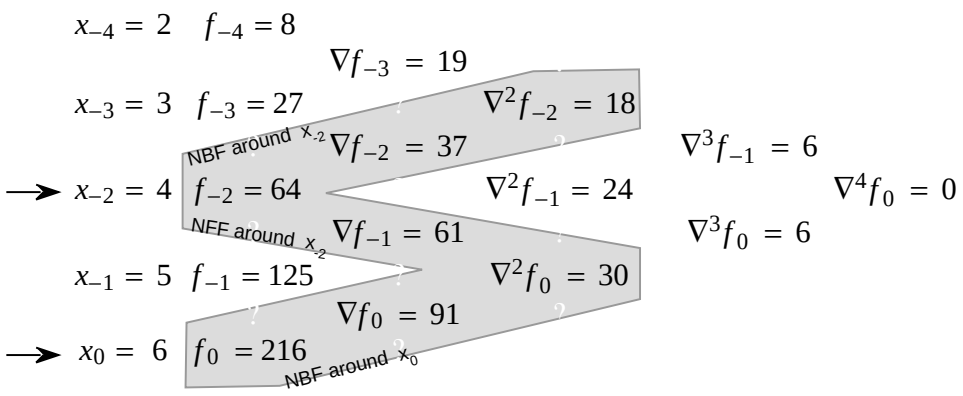


Figure 5.6: Example of a Newton backward difference table

Example

A set if five ($n = 4$) equally spaced data points and the backward difference table for the data are shown in figure 5.6. This is the same example as used in the previous section! It is clear that $h = 1$ and $x = x_0 + \alpha$. In the previous case we constructed a linear, quadratic and cubic polynomials, with $x_3 = 4$ as the reference point. In the present case let us use the same reference point, but it is labelled as $x_{-2} = 4$. A quadratic backward difference polynomial in α is,

$$P_2(4+\alpha) = (64) + \alpha(37) + \frac{\alpha(\alpha+1)}{2!}(18) + \mathcal{O}(h^3)$$

which passes through the points (x_{-2}, f_{-2}), (x_{-3}, f_{-3}) and (x_{-4}, f_{-4}) for $\alpha = 0, -1, -2$, respectively. Recall that the forward difference polynomial around the same point was,

Calculate the interpolated value of $f(4.5)$ from these two polynomials

$$P_2(4+\alpha) = (64) + \alpha(61) + \frac{\alpha(\alpha-1)}{2!}(30)$$

which passes through the three forward point for $\alpha = 0, 1, 2$. Although they are based on the same reference point, these are two different polynomials passing through a different set of data points.

As a final example, let us construct a quadratic backward difference polynomial around $x_0 = 6$. It is,

Is this polynomial different from the NFF, $P_2(4+\alpha)$ constructed above?

$$P_2(6+\alpha) = (216) + \alpha(91) + \frac{\alpha(\alpha+1)}{2!}(30)$$

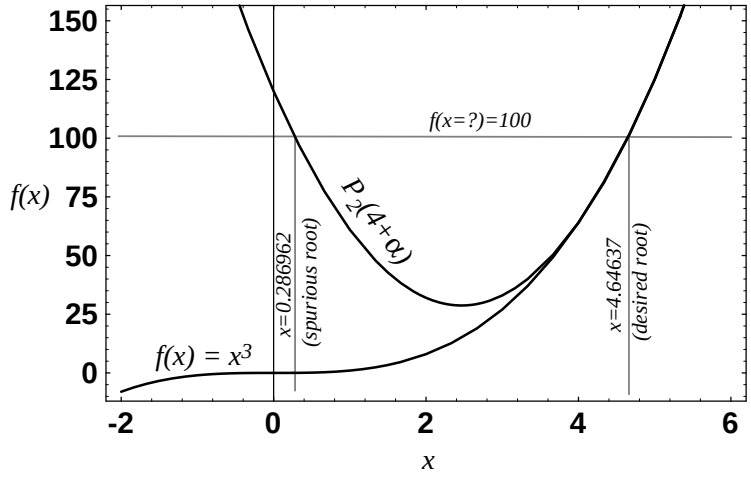


Figure 5.7: Example of a inverse interpolation

5.4 Inverse interpolation

One of the objectives in constructing an interpolating polynomial is to be able to evaluate the function $f(x)$ at values of x other than the ones in the discrete set of given data points (x_i, f_i). The objective of *inverse interpolation* is to determine the independent variable x for a given value of f using a given discrete data set (x_i, f_i). If x_i are equally spaced, we can combined two of the tools (polynomial curve fitting and root finding) to meet this objective, although this must be done with caution.

We illustrate this with the example data shown in figure 5.4. Suppose we wish to find the value of x where $f = 100$. Using the three data points in the neighbourhood of $f = 100$ in figure 5.4 viz. (x_3, x_4, x_5), and using a quadratic polynomial fit, we have,

$$P_2(4+\alpha) = (64) + \alpha(61) + \frac{\alpha(\alpha-1)}{2!}(30)$$

A graph of this polynomial approximation $P_2(4+\alpha)$ and the actual function $f(x) = x^3$ used to generate the data given in figure 5.4 are shown in figure 5.7. It is clear that the polynomial approximation is quite good in the range of $x \in (4,6)$, but becomes a poor approximation for lower values of x. Note, in particular, that if we solve the inverse interpolation problem by setting

$$P_2(4+\alpha) - 100 = 0 \qquad \text{or} \qquad (64) + \alpha(61) + \frac{\alpha(\alpha-1)}{2!}(30) - 100 = 0$$

we will find two roots. One of them at $\alpha = 0.64637$ or $x = 4.64637$ is the desired root while the other at $\alpha = -3.71304$ or $x = 0.286962$ is a spurious one. This problem can become compounded as we use higher degree polynomial in an effort to improve accuracy.

In order to achieve high accuracy, but stay close to the desired root, we can generate an initial guess from a *linear interpolation*, followed by constructing a *fixed point iteration* scheme on the polynomial approximation of the desired accuracy. Convergence is generally fast as shown in Dahlquist and Björck (1974). Suppose we wish to find x corresponding to $f(x) = d$, the desired function value. We first construct a polynomial of degree $(m-1)$ to represent the tabular data.

$$P_{m-1}(x_1 + \alpha h) = \left[1 + \alpha\Delta + \frac{\alpha(\alpha-1)}{2!}\Delta^2 + \frac{\alpha(\alpha-1)(\alpha-2)}{3!}\Delta^3 + \cdots \right.$$
$$\left. \frac{\alpha(\alpha-1)(\alpha-2)\cdots(\alpha-m+2)}{(m-1)!}\Delta^{m-1}\right] f(x_1)$$

Then we let $f(x_1 + \alpha h) = d$ and rearrange the polynomial in the form

$$\alpha_{i+1} = g(\alpha_i) \qquad i = 0, 1, 2 \cdots$$

where $g(\alpha)$ is obtained by rearranging the polynomial,

$$g(\alpha) = \frac{1}{\Delta f_1}\left[d - f_1 - \frac{\alpha(\alpha-1)}{2!}\Delta^2 f_1 - \frac{\alpha(\alpha-1)(\alpha-2)}{3!}\Delta^3 f_1 + \cdots\right]$$

and the initial guess obtained by truncating the polynomial after the linear term,

$$\alpha_0 = \frac{d - f_1}{\Delta f_1}$$

Example

Continuing with the task of finding x where $f(x) = 100$ for the data shown in figure 5.4, the fixed point iterate is,

$$\alpha_{i+1} = [100 - 64 - 15\alpha_i(\alpha_i - 1)]/61$$

and the initial guess is

$$\alpha_0 = \frac{d - f_1}{\Delta f_1} = \frac{100 - 64}{61} = 0.5902$$

The first ten iterates, produced from the m-file given below,

i	α_i
1	.59016393
2	.64964028
3	.64613306
4	.64638815
5	.64636980
6	.64637112
7	.64637102
8	.64637103
9	.64637103
10	.64637103

Table 5.3: Inverse interpolation

```
function a=g(a)
for i=1:10
fprintf(1,'%2i %12.7e\n',i,a);
a=(100 - 64 - 15*a*(a-1))/61;
end
```

are shown in Table 5.3.

5.5 Lagrange polynomials

So far we have examined ways to construct polynomial approximations using equally spaced data in x. For a data set $\{x_i, f_i | i = 0, \cdots n\}$, that contains unequally spaced data in the independent variable x, we can construct Lagrange interpolation formula as follows.

$$P_n(x) = \sum_{i=0}^{n} f_i \delta_i(x) \tag{5.14}$$

where

$$\delta_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

Note that

$$\delta_i(x_j) = \begin{cases} 0 & j \neq i \\ 1 & j = i \end{cases}$$

and each $\delta_i(x)$ is a polynomial of degree n. It is also clear from equation (5.14) that $P_n(x_j) = f_j$ - *i.e.*, the polynomial passes through the data points (x_j, f_j).

An alternate way to construct the Lagrange polynomial is based on introducing the divided difference and constructing a divided difference table. The polynomial itself is written in the form

$$\begin{aligned} P_n(x) &= \sum_{i=0}^{n} a_i \prod_{j=0}^{i} (x - x_{j-1}) \\ &= a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots \\ &\quad + a_n(x - x_0) \cdots (x - x_{n-1}) \end{aligned} \tag{5.15}$$

The advantage of writing it the form shown in equation (5.15) is that the unknown coefficients a_i can be constructed recursively or found directly from the divided difference table. The first divided difference is defined by the equation,

$$f[x_0, x_1] = \frac{f_1 - f_0}{x_1 - x_0}$$

Similarly the second divided difference is defined as,

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$$

With these definitions, we return to the task of finding the coefficients a_i in equation (5.15) For example, the first coefficient a_0 is,

$$P_n(x_0) = a_0 = f[x_0] = f_0$$

The second coefficient, a_1, is obtained from,

$$P_n(x_1) = a_0 + a_1(x_1 - x_0) = f_1$$

which can be rearranged as,

$$a_1 = \frac{f_1 - f_0}{x_1 - x_0} = f[x_0, x_1]$$

The third coefficient is obtained from,

$$P_n(x_2) = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) = f_2$$

The only unknown here is a_2, which after some rearrangement becomes,

$$a_2 = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = f[x_0, x_1, x_2]$$

In general the n-th coefficient is the n-th divided difference.

$$a_n = f[x_0, x_1, \cdots, x_n]$$

$x_0 = 1.0$	$f_0 = 1.000$			
		$f[x_0, x_1] = 3.6400$		
$x_1 = 1.2$	$f_1 = 1.728$		$f[x_0, x_1, x_2] = 3.700$	
		$f[x_1, x_2] = 5.4900$		$f[x_0, x_1, x_2, x_3] = 1.000$
$x_2 = 1.5$	$f_2 = 3.375$		$f[x_1, x_2, x_3] = 4.300$	
		$f[x_2, x_3] = 7.2100$		
$x_3 = 1.6$	$f_3 = 4.096$			

Figure 5.8: Structure of divided difference table for 4 unequally spaced data

Example

Consider the example data and the divided difference table shown in figure 5.8. If we wish to construct a quadratic polynomial passing through $(x_0, f_0), (x_1, f_1), (x_2, f_2)$ for example using equation (5.14), it will be

$$\begin{aligned} P_2(x) &= f_0 \frac{(x - x_1)(x - x_2)}{(x_0 - x1)(x_0 - x_2)} + f_1 \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} + f_2 \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} \\ &= 1.00 \frac{(x - 1.2)(x - 1.5)}{(1 - 1.2)(1 - 1.5)} + 1.728 \frac{(x - 1)(x - 1.5)}{(1.2 - 1)(1.2 - 1.5)} + 3.375 \frac{(x - 1)(x - 1.2)}{(1.5 - 1)(1.5 - 1.2)} \end{aligned}$$

The same polynomial using equation (5.15) and the difference table shown in figure 5.8 will be written as,

$$\begin{aligned} P_2(x) &= f_0 + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ &= 1.000 + 3.64(x - 1) + 3.70(x - 1)(x - 1.2) \end{aligned}$$

Observe that in order to construct a cubic polynomial by adding the additional data point (x_3, f_3) Lagrange polynomial based on equation (5.14) requires a complete reconstruction of the equation, while that based on equation (5.15) is simply,

$$\begin{aligned} P_2(x) &= f_0 + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \\ &\quad f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2) \\ &= 1.000 + 3.64(x - 1) + 3.70(x - 1)(x - 1.2) + \\ &\quad 1(x - 1)(x - 1.2)(x - 1.5) \end{aligned}$$

A MATLAB function that implements that Lagrange interpolation formula shown in equation (5.14) is given in figure 5.9. This function ac-

```
function f=LagrangeP(xt,ft,x)
% (xt,ft) are the table of unequally spaced values
%  x is where interpolated values are required
%  f the interpolated values are returned

m=length(x);
nx=length(xt);
ny=length(ft);

if (nx ~= ny),
  error(' (xt,ft) do not have the same # values')
end

for k=1:m
 sum = 0;
 for i=1:nx
  delt(i)=1;
  for j=1:nx
   if (j ~= i),
     delt(i) = delt(i)*(x(k)-xt(j))/(xt(i)-xt(j));
   end
  end
  sum = sum + ft(i) * delt(i) ;
 end
 f(k)=sum;
end
```

Figure 5.9: MATLAB implementation of Lagrange interpolation polynomial

cepts a table of values (xt, ft), constructs the highest degree Lagrange polynomial that is possible and finally evaluates and returns the interpolated values of the function y at specified values of x.

```
»xt=[1  1.2  1.5 1.6]          % Define xt, unequally spaced
»ft=[1  1.728  3.375 4.096]    % Define ft, the function values
»x=[1.0:0.1:1.6]               % x locations for interpolation
»f=LagrangeP(xt,ft,x)          % interpolated f values.
```

5.6 Numerical differentiation

Having obtained approximate functional representations as outlined in sections §5.3.2 or §5.5, we can proceed to construct algorithms for approximate representations of derivatives.

5.6.1 Approximations for first order derivatives

Consider the Newton forward formula given in equation (5.11)

$$f(x) \approx P_{m-1}(x_1 + \alpha h) = \left[1 + \alpha\Delta + \frac{\alpha(\alpha-1)}{2!}\Delta^2 + \frac{\alpha(\alpha-1)(\alpha-2)}{3!}\Delta^3 + \cdots + \frac{\alpha(\alpha-1)\cdots(\alpha-m+2)}{(m-1)!}\Delta^{m-1}\right] f(x_1) + \mathcal{O}(h^m)$$

that passes through the given data set $\{(x_i, f_i) \mid i = 1, \cdots, m\}$. Note that the independent variable x has been transformed into α using $x = x_1 + \alpha h$, hence $dx/d\alpha = h$. Now, the first derivative is obtained as,

$$f'(x) = \frac{df}{dx} \approx \frac{dP_{m-1}}{dx} = \frac{dP_{m-1}}{d\alpha}\frac{d\alpha}{dx} = \frac{1}{h}\left[\Delta + \frac{\alpha + (\alpha-1)}{2}\Delta^2 + \frac{\{\alpha(\alpha-1) + (\alpha-1)(\alpha-2) + \alpha(\alpha-2)\}}{6}\Delta^3 + \cdots\right] f(x_1) \quad (5.16)$$

Equation (5.16) forms the basis of deriving a class of approximations for first derivatives from a tabular set of data. Note that the equation (5.16) is still a function in α and hence it can be used to evaluate the derivative at any value of $x = x_1 + \alpha h$. Also, the series can be truncated after any number of terms. Thus, a whole class of successively more accurate representations for the first derivative can be constructed from equation (5.16) by truncating the series at higher order terms. For example

evaluating the derivative at the reference point x_1, (*i.e.*, $\alpha = 0$) equation (5.16) reduces to,

$$f'(x_1) = \frac{1}{h}\left[\Delta - \frac{1}{2}\Delta^2 + \frac{1}{3}\Delta^3 - \frac{1}{4}\Delta^4 \cdots \pm \frac{1}{m-1}\Delta^{m-1}\right] f(x_1) + \mathcal{O}(h^{m-1})$$

This equation can also be obtained directly using equation (5.9) as,

$$E = e^{hD} \quad \text{or } hD = \ln E = \ln(1+\Delta)$$

Expanding the logarithmic term we obtain,

$$hD = \Delta - \frac{\Delta^2}{2} + \frac{\Delta^3}{3} - \frac{\Delta^4}{4} + \cdots$$

Operating both sides with $f(x_1)$ (*i.e.*, using x_1 as the reference point), we get,

$$Df(x_1) = f'(x_1) = \frac{1}{h}\left[\Delta - \frac{\Delta^2}{2} + \frac{\Delta^3}{3} - \frac{\Delta^4}{4} + \cdots\right] f(x_1) \qquad (5.17)$$

Now, truncating the series after the first term ($m = 2$),

$$\begin{aligned} f'(x_1) &= \frac{1}{h}[\Delta f(x_1)] + \mathcal{O}(h) \\ &= \frac{1}{h}[f_2 - f_1] + \mathcal{O}(h) \end{aligned}$$

which is a 2-point, first order accurate, forward difference approximation for first derivative at x_1. Truncating the series after the first two terms ($m = 3$),

$$\begin{aligned} f'(x_1) &= \frac{1}{h}\left[\Delta f(x_1) - \frac{1}{2}\Delta^2 f(x_1)\right] + \mathcal{O}(h^2) \\ &= \frac{1}{h}\left[(f_2 - f_1) - \frac{1}{2}(f_1 - 2f_2 + f_3)\right] + \mathcal{O}(h^2) \\ &= \frac{1}{2h}[-3f_1 + 4f_2 - f_3] + \mathcal{O}(h^2) \end{aligned}$$

which is the 3-point, second order accurate, forward difference approximation for the first derivative at x_1. Clearly both are approximate representations of the first derivative at x_1, but the second one is more accurate since the truncation error is of the order h^2.

Note that while, equation (5.17) is evaluated at the reference point on both sides of the equation, the earlier equation (5.16) is a polynomial that is constructed around the reference point x_1, but can be evaluated at

Derivative at x_i	Difference approximation	truncation error
$f'(x_i)$	$(f_{i+1} - f_i)/h$	$\mathcal{O}(h)$
$f'(x_i)$	$(f_i - f_{i-1})/h$	$\mathcal{O}(h)$
$f'(x_i)$	$(-3f_i + 4f_{i+1} - f_{i+2})/2h$	$\mathcal{O}(h^2)$
$f'(x_i)$	$(+3f_i - 4f_{i-1} + f_{i-2})/2h$	$\mathcal{O}(h^2)$
$f'(x_i)$	$(f_{i+1} - f_{i-1})/2h$	$\mathcal{O}(h^2)$
$f'(x_i)$	$(f_{i-2} - 8f_{i-1} + 8f_{i+1} - f_{i+2})/12h$	$\mathcal{O}(h^4)$
$f''(x_i)$	$(f_{i+1} - 2f_i + f_{i-1})/h^2$	$\mathcal{O}(h^2)$
$f''(x_i)$	$(f_{i+2} - 2f_{i+1} + f_i)/h^2$	$\mathcal{O}(h)$
$f''(x_i)$	$(-f_{i-3} + 4f_{i-2} - 5f_{i-1} + 2f_i)/h^2$	$\mathcal{O}(h^2)$
$f''(x_i)$	$(-f_{i+3} + 4f_{i+2} - 5f_{i+1} + 2f_i)/h^2$	$\mathcal{O}(h^2)$

Table 5.4: Summary of difference approximations for derivatives

any other point by choosing appropriate α values. For example, consider the first derivative at $x = x_2$ or $\alpha = 1$. Two term truncation of equation (5.16) yields,

$$f'(x_2) = \frac{1}{h}\left[\Delta + \frac{1}{2}\Delta^2\right] f(x_2) + \mathcal{O}(h^2)$$

or

$$f'(x_2) = \frac{1}{2h}\left[f_3 - f_1\right] + \mathcal{O}(h^2)$$

which is a 3-point, second order accurate, central difference approximation for the first derivative at x_2.

Going through a similar exercise as above with the Newton backward difference formula (5.13), truncating the series at various levels and using diffrent reference points, one can easily develop a whole class of approximations for first order derivatives. Some of the useful ones are summarized in Table 5.4.

5.6.2 Approximations for second order derivatives

The second derivative of the polynomial approximation is obtained by taking the derivative of equation (5.16) one more time - viz.

$$\begin{aligned} f''(x) \approx \frac{d}{d\alpha}\left[\frac{dP_{m-1}}{d\alpha}\frac{d\alpha}{dx}\right]\frac{d\alpha}{dx} = \frac{1}{h^2}\Big[\Delta^2 + \\ \frac{\{\alpha + (\alpha-1) + (\alpha-1) + (\alpha-2) + \alpha + (\alpha-2)\}}{6}\Delta^3 \\ + \cdots\Big] f(x_1) + \mathcal{O}(h^{m-2}) \end{aligned} \tag{5.18}$$

Evaluating at $\alpha = 0$ (*i.e.*, $x = x_1$), we obtain,

$$f''(x_1) = \frac{1}{h^2}\left[\Delta^2 - \Delta^3 + \frac{11}{12}\Delta^4 - -\frac{5}{6}\Delta^5 + \frac{137}{180}\Delta^6 \cdots\right] f(x_1) \quad (5.19)$$

This equation can also be obtained directly using equation (5.9) as,

$$(hD)^2 = (\ln E)^2 = (\ln(1+\Delta))^2$$

Expanding the logarithmic term we obtain,

$$\begin{aligned}(hD)^2 &= \left[\Delta - \frac{\Delta^2}{2} + \frac{\Delta^3}{3} - \frac{\Delta^4}{4} + \frac{\Delta^5}{5} \cdots\right]^2 \\ &= \left[\Delta^2 - \Delta^3 + \frac{11}{12}\Delta^4 - \frac{5}{6}\Delta^5 + \frac{137}{180}\Delta^6 - \frac{7}{10}\Delta^7 + \frac{363}{560}\Delta^8 \cdots\right]\end{aligned}$$

Operating both sides on $f(x_1)$ (*i.e.*, using x_1 as the reference point), we get,

$$D^2 f(x_1) = f''(x_1) = \frac{1}{h^2}\left[\Delta^2 - \Delta^3 + \frac{11}{12}\Delta^4 - \cdots\right] f(x_1)$$

Truncating after one term,

$$f''(x_1) = \frac{1}{h^2}\left[\Delta^2 f(x_1)\right] = \frac{1}{h^2}(f_1 - 2f_2 + f_3) + \mathcal{O}(h)$$

Truncating after two terms,

$$\begin{aligned}f''(x_1) &= \frac{1}{h^2}\left[\Delta^2 f(x_1) - \Delta^3 f(x_1)\right] \\ &= \frac{1}{h^2}(2f_1 - 5f_2 + 4f_3 - f_4) + \mathcal{O}(h^2)\end{aligned}$$

Evaluating equation(5.18) at x_2 (or $\alpha = 1$), we get,

$$f''(x_2) = \frac{1}{h^2}\left[\Delta^2 - 0 \cdot \frac{\delta^3}{6}\right] f(x_1) + \mathcal{O}(h^2)$$

Note that the third order term turns out to be zero and hence this formula turns out to be more accurate. This is a 3-point, second order accurate central difference approximation for the second derivative given as,

$$f''(x_2) = \frac{1}{h^2}\left[\Delta^2 f(x_1)\right] = \frac{1}{h^2}(f_1 - 2f_2 + f_3) + \mathcal{O}(h^2)$$

5.6.3 Taylor series approach

One can derive finite difference approximations from Taylor series expansion also. Consider the following expansions around x_i.

$$f(x_i + h) = f(x_i) + hf'(x_i) + \frac{h^2}{2}f''(x_i) + \frac{h^3}{3!}f'''(x_i) + \cdots$$

$$f(x_i - h) = f(x_i) - hf'(x_i) + \frac{h^2}{2}f''(x_i) - \frac{h^3}{3!}f'''(x_i) + \cdots$$

Subtracting the second from the first equation, and extracting $f'(x_i)$, we get,

$$f'(x_i) = \frac{f(x_i + h) - f(x_i - h)}{2h} - \frac{h^2}{6}f''(x_i) + \cdots$$

or,

$$f'(x_i) = \frac{f_{i+1} - f_{i-1}}{2h} + \mathcal{O}(h^2)$$

which is a central difference formula for the first derivative that we derived in the last section §5.6. Adding the two Taylor series expansions above, we get,

$$f_{i+1} + f_{i-1} = 2f_i + h^2 f''(x_i) + \frac{h^4}{12}f''''(x_i) + \cdots$$

or,

$$f''(x_i) = \frac{1}{h^2}\left[f_{i+1} + f_{i-1} - 2f_i\right] + \mathcal{O}(h^2)$$

which is a central difference formula for the second derivative that we derived in the last section §5.6.

5.7 Numerical integration

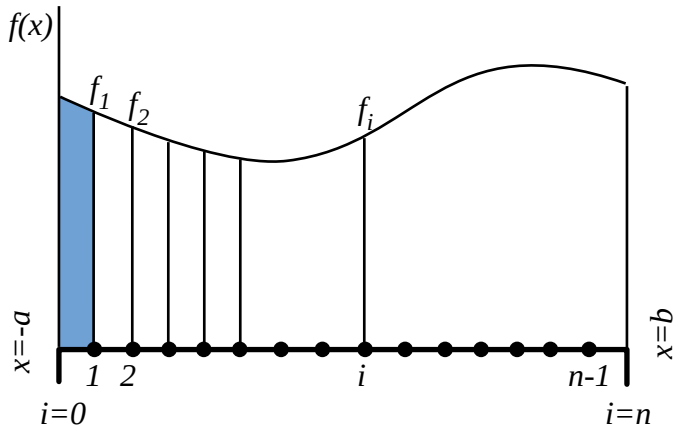
The ability to evaluate definite integrals numerically is useful either (i) when the function is complicated and hence is not easy to integrate analytically or (ii) the data is given in equally spaced, tabular from. In either case the starting point is to use the functional approximation methods seen in earlier sections followed by the integration of the approximate function. By doing this formally, with the Newton forward polynomials, we can develop a class of integration formulas. Consider the integral,

$$\int_a^b f(x)dx \tag{5.20}$$

Since the function can be represented by a polynomial of degree n as,

$$f(x) = P_n(x_0 + \alpha h) + \mathcal{O}(h^{n+1})$$

with an error of order h^{n+1}, we can use this approximation to carry out the integration. We first divide the interval $x \in [a, b]$ into n subdivisions as shown in the sketch below; hence there will be



$(n + 1)$ data points labelled as $\{x_0, x_1, \cdots x_n\}$ and we have

$$h = (b - a)/n, \qquad x = x_0 + \alpha h \qquad dx = h d\alpha$$

As an illustration let us take a first degree polynomial between x_0 and x_1. We have

$$\int_{x_0}^{x_1} f(x)dx \approx \int_0^1 P_1(x_0 + \alpha h)h d\alpha + \int_0^1 \mathcal{O}(h^2)h d\alpha$$

or,

$$\int_{x_0}^{x_1} f(x)dx \approx \int_0^1 [1 + \alpha\Delta]\ f_0\ h\ d\alpha + \mathcal{O}(h^3)$$

which upon, completing the integration becomes,

$$\int_{x_0}^{x_1} f(x)dx \approx \frac{h}{2}[f_0 + f_1] + \underbrace{\mathcal{O}(h^3)}_{\textit{local error}} \tag{5.21}$$

This formula is the well known trapezoidal rule for numerical integration. The geometrical interpretation is that it represents the shaded area under the curve. Note that while numerical differentiation, as developed in equation (5.16), lowers the order of the truncation error by one due to the term $d\alpha/dx = 1/h$ numerical integration increases the order of the

truncation error by one due to the term $dx = hd\alpha$. In the above formula the truncation error is of order $\mathcal{O}(h^3)$. It is called the *local truncation error* since it is the error in integrating over *one* interval $x \in (x_0, x_1)$. To obtain the complete integral over the interval $x \in [a, b]$ we apply equation (5.21) repeatedly over each of the subdivisions as,

$$\int_a^b f(x)dx = \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} f(x)dx = \sum_{i=1}^{n} \frac{h}{2}[f_{i-1} + f_i] + \underbrace{\sum_{i=1}^{n} \mathcal{O}(h^3)}_{\textit{global error}}$$

Recalling that $n = (b-a)/h$ the global or accumulated error becomes of order $\mathcal{O}(h^2)$. Thus the trapezoidal rule has a *local truncation error* of order $\mathcal{O}(h^3)$ and a *global truncation error* of order $\mathcal{O}(h^2)$ and the equation is,

$$\boxed{\int_a^b f(x)dx = \frac{h}{2}\sum_{i=1}^{n}[f_{i-1} + f_i] + \underbrace{\mathcal{O}(h^2)}_{\textit{global error}}} \tag{5.22}$$

By taking an quadratic functional approximation and integrating over the range of $x \in [x_0, x_2]$ we obtain the Simpson's rule.

$$\int_{x_0}^{x_2} f(x)\ dx \approx \int_0^2 P_2(x_0 + \alpha h)\ h\ d\alpha + \int_0^2 \mathcal{O}(h^3)hd\alpha$$

or,

$$\int_{x_0}^{x_2} f(x)\ dx \approx \int_0^2 \left[1 + \alpha\Delta + \frac{\alpha(\alpha-1)}{2}\Delta^2\right] f_0\ h\ d\alpha + \mathcal{O}(h^4)$$

which upon, completing the integration becomes,

$$\int_{x_0}^{x_2} f(x)\ dx \approx \frac{h}{3}[f_0 + 4f_1\ + f_2] + \underbrace{\mathcal{O}(h^4)}_{\textit{local error}} \tag{5.23}$$

Note that the next neglected term in the polynomial $P_2(x_0 + \alpha h)$ that corresponds to order $\mathcal{O}(h^3)$ term viz.

$$\int_0^2 \frac{\alpha(\alpha-1)(\alpha-2))}{3!}\Delta^3 f_0\ h\ d\alpha$$

turns out to be exactly zero, thus making the *local truncation error* in the Simpson's rule to be actually of order $\mathcal{O}(h^5)$ Repeated application of

the Simpson's rule results in,

$$\int_a^b f(x)dx = \frac{h}{3}\left[f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \cdots + f_n\right] + \underbrace{\mathcal{O}(h^4)}_{\textit{global error}} \tag{5.24}$$

Note that in applying Simpson's rule repeatedly over the interval $x \in [a, b]$, we must have an even number of intervals (n even) or equivalently an odd number of points.

5.7.1 Romberg Extrapolation

An idea similar to that used in section §2.8 to accelerate convergence is the notion of extrapolation to improve accuracy of numerical integration. The basic idea is to estimate the *truncation error* by evaluating the integral on two different grid sizes, h_1 and h_2. Let us apply this idea on the trapezoidal rule which has a *global truncation error* of $\mathcal{O}(h^2)$. Let the exact integral be represented as,

$$I = I(h_1) + E(h_1)$$

where $I(h_1)$ is the approximate estimate of the integral using grid size of h_1 and $E(h_1)$ is the error. Similarly we have,

$$I = I(h_2) + E(h_2)$$

But, for trapezoidal rule we have $E(h_i) \propto h_i^2$. Hence

$$\frac{E(h_1)}{E(h_2)} = \frac{h_1^2}{h_2^2}$$

We can combine these equations as,

$$I = I(h_1) + E(h_1) = I(h_2) + E(h_2)$$

or,

$$I(h_1) + E(h_2)\frac{h_1^2}{h_2^2} = I(h_2) + E(h_2)$$

which can be solved to obtain $E(h_2)$ as,

$$E(h_2) = \frac{I(h_1) - I(h_2)}{[1 - (h_1/h_2)^2]}$$

Hence a better estimate for the integral is,

$$I = I(h_2) + \frac{1}{[(h_1/h_2)^2 - 1]}[I(h_2) - I(h_1)]$$

If $h_2 = h_1/2$ then we have,

$$I = I(h_2) + \frac{1}{[2^2 - 1]}[I(h_2) - I(h_1)]$$

Since we have estimated and eliminated the truncation error of order $\mathcal{O}(h^2)$ term, the above equation will have an error of order $\mathcal{O}(h^4)$ which is the next leading term in the Taylor series expansion. By repeated application of the above approach to estimate and eliminate successively higher order terms, we can arrive at the following general formula for Romberg extrapolation.

$$\boxed{I_{j,k} = \frac{4^{k-1} I_{j+1,k-1} - I_{j,k-1}}{4^{k-1} - 1}} \tag{5.25}$$

Example of Romberg extrapolation

Consider the following integral,

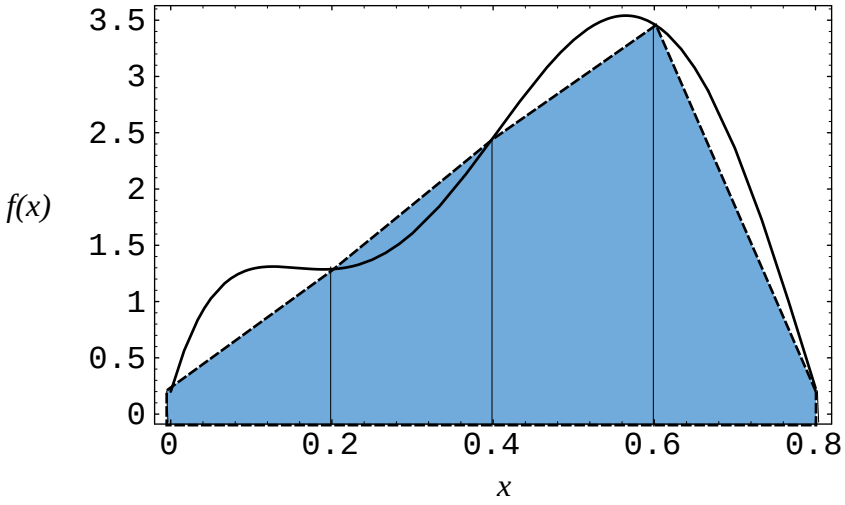
$$\int_0^{0.8} (0.2 + 25x - 200x^2 + 675x^3 - 900x^4 + 400x^5)\, dx = 1.64053334 \tag{5.26}$$

which has the exact value as shown. A sketch of the function $f(x)$ and the Romberg extrapolation results are shown in figure 5.10. It is clear from this example that by combining three rather poor estimates of the integral on grids of $h = 0.8, 0.4$ and 0.2, a result accurate to eight significant digits has been obtained! For example, $I_{2,2}$ is obtained by using $j = 2$ and $k = 2$ which results in,

$$I_{2,2} = \frac{4 \times I_{3,1} - I_{2,1}}{4 - 1} = \frac{4 \times 1.4848 - 1.0688}{3} = 1.6234667$$

Similarly, $I_{1,3}$ is obtained by using $j = 1$ and $k = 3$ which results in,

$$I_{1,3} = \frac{4^2 \times I_{2,2} - I_{1,2}}{4^2 - 1} = \frac{4^2 \times 1.6234667 - 1.3674667}{15} = 1.64053334$$



		$O(h^2)$	$O(h^4)$	$O(h^6)$
		$(k=1)$	$(k=2)$	$(k=3)$
$j=1$	$h=0.8$	0.1728	1.3674667	1.64053334
$j=2$	$h=0.4$	1.0688	1.6234667	1.64053334
$j=3$	$h=0.2$	1.4848	1.6394667	
$j=4$	$h=0.1$	1.6008		

Figure 5.10: Illustration of Romberg extrapolation

```
function f=int_ex(x)
%defines a 5th degree polynomial

m=length(x);
for i=1:m
 f(i) = 0.2 + 25*x(i)  - 200*x(i)^2 + ...
        675*x(i)^3 - 900*x(i)^4 + 400*x(i)^5;
end
```

Figure 5.11: MATLAB implementation of quadrature evaluation

MATLAB example

MATLAB provides a m-file to evaluate definite integrals using adaptive, recursive Simpson's quadrature. You must of course, define the function through a m-file which should accept a vector of input arguments and return the corresponding function values. A m-file that implements equation (5.26) is shown in figure 5.11. After creating such a file work through the following example during an interactive session.

```
»quad('int_ex',0,0.8,1e-5)      % evaluate integral over (0,0.8)
»quad('int_ex',0,0.8,1e-5,1)    % Display results graphically
»quad('int_ex',0,0.8,1e-8)      % Note the warning messges!
```

5.7.2 Gaussian quadratures

5.7.3 Multiple integrals

5.8 Orthogonal functions

5.9 Piecewise continuous functions - splines