Summary of last session

-Chapter 4. Solving Systems of Linear Equations

- **4.4 Norms and the Analysis of Errors**

  Vector norms, Matrix Norms, Condition Number

- **4.6 Solution of Equations by Iterative Methods**

  Jacobi method, Gauss-Seidel method,
  General iteration methods, Richardson method.

Chapter 6

Approximating Functions

6.0 Introduction

For many engineering problems, we need to analyze or interpret some discrete data obtained from experiments or some observations. We need to find the regularities, or derivatives, or intergations, or roots and so on. Therefore we need to construct some approximate functions from the data. These approximate functions should be simple in form and good at representing the data.

In this chapter, several methods of how to construct such fuctions are discussed. Several different sub-problems will be considered. They differ according to the type of functions being represented, whether it is known at relatively few points or at many (or all) points.

We will discuss four methods mainly. They are **Polynomial Inter – polation (Lagrange and Newton forms (Divided Differences) and Hermite interpolation) and Least - Squares Approximation.**

6.1 Polynomial Interpolation

-method of undetermined coefficients

Assuming that the function $y = f(x)$ is defined in the interval $[a, b]$ and the values of $y_i = f(x_i)$ at $x_i (i = 0, 1, 2, \cdots, n)$ $(x_0, x_1, \cdots, x_n \in [a, b])$ are given. If there exists a simple function p that makes

$$p(x_i) = y_i \qquad (0 \le i \le n) \qquad (1)$$

then p is called the **interpolation function** of f, and $x_0, x_1, \cdots, x_n$ are the **interpolation nodes**, $[a, b]$ is the **interpolation interval**, (1) is the **interpolation conditions** and the methods of finding function p is the **interpolation methods**. When the function p

$$p = p_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \qquad (2)$$

is a polynomial of degree at most n where $a_0, \cdots, a_n$ are coefficients of the polynomial, and the method is called **polymonial interpolation**.

6.1 Polynomial Interpolation

-method of undetermined coefficients

The coefficients in (2) satisfy the following linear equation system:

$$\begin{cases} a_0 + a_1 x_0 + a_2 x_0^2 + \mathrm{L} + a_n x_0^n = y_0 \\ a_0 + a_1 x_1 + a_2 x_1^2 + \mathrm{L} + a_n x_1^n = y_1 \\ \qquad\qquad \mathrm{M} \\ a_0 + a_1 x_n + a_2 x_n^2 + \mathrm{L} + a_n x_n^n = y_n \end{cases} \qquad (3)$$

To prove the existence and uniqueness of function p, we need to show that the solution of (3) exists and is unique. The necessary condition for this is

$$V_n(x_0, x_1, \mathrm{L}, x_n) = \begin{vmatrix} 1 & x_0 & x_0^2 & \mathrm{L} & x_0^n \\ 1 & x_1 & x_1^2 & \mathrm{L} & x_1^n \\ \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{L} & \mathrm{M} \\ 1 & x_n & x_n^2 & \mathrm{L} & x_n^n \end{vmatrix} = \prod_{i=1}^{n} \prod_{j=0}^{i-1} (x_i - x_j)$$

which is called Vandermonde determinant, is not zero. As the nodes are distinct, $V_n \neq 0$. Therefore the solution exists and is unique.

6.1 Polynomial Interpolation

-method of undetermined coefficients

The error of this interpolation can be analyzed as follows: Assuming that the derivations $f', f'', \cdots, f^{(n+1)}$ exists in the interval, the error $R_n(x)$ (also called remaining item) of the interpolation can be calculated by

$$R_n(x) = f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega_{n+1}(x) \qquad (4)$$

where $\xi \in (a,b)$ and depends on x, and $\omega_{n+1}(x) = \prod_{i=0}^{n}(x - x_i)$.

Generally, the value of ξ is unknown. However if we can determine the upper bound of $f^{(n+1)}$, i.e. $\max_{a<x<b}|f^{(n+1)}| = M_{n+1}$, then the error term is limited by

$$|R_n(x)| \le \frac{M_{n+1}}{(n+1)!}|\omega_{n+1}(x)|$$

6.1 Polynomial Interpolation

-Lagrange interpolation method

The caculation for the coefficients $[a_0, \mathsf{L}\ , a_n]$ by solving the equation system (3) can be very time consuming. We introduce following methods. We have many ways to give the polynomial p for given data.

Firstly, let's consider a two-point case, x_0, x_1 $(n=1)$. We have conditions

$$p_1(x_0) = y_0 = a_0 + a_1 x_0$$
$$p_1(x_1) = y_1 = a_0 + a_1 x_1$$

Solving the equation system, we get

$$a_0 = \frac{y_1 x_0 - y_0 x_1}{x_0 - x_1}, \quad a_1 = \frac{y_0 - y_1}{x_0 - x_1}$$

So the polynomial is

$$p_1(x) = \frac{y_1 x_0 - y_0 x_1}{x_0 - x_1} + \frac{y_0 - y_1}{x_0 - x_1} x$$

This is actually using a straight line which pass through the two points (x_0, y_0) and (x_1, y_1) to approximate the function $y = f(x)$.

6.1 Polynomial Interpolation

-Lagrange interpolation method

If we write the interpolation function p_1 in the sum of two linear functions, we have

$$p_1(x) = y_0\left(\frac{x - x_1}{x_0 - x_1}\right) + y_1\left(\frac{x - x_0}{x_1 - x_0}\right)$$

let

$$l_0(x) = \frac{x - x_1}{x_0 - x_1}, \text{ and } \quad l_1(x) = \frac{x - x_0}{x_1 - x_0}$$

Obviously,

$$l_0(x_0) = 1, \quad l_0(x_1) = 0, \quad l_1(x_0) = 0, \quad l_1(x_1) = 1.$$

This indicates that at the *i*th-interpolation point $l_i(x_i) \ \ (i = 0,1) = 1$ and $l_i(x) = 0$ elsewhere. Similarly, we consider $n+1$ nodes, and construct a function $l_i(x) \ \ (i = 0,1,\text{L} \ ,n)$ of which the degree is not greater than n to satisfy

$$l_i(x_j) = \begin{cases} 0, & j \neq i \\ 1, & j = i \end{cases} \qquad (5)$$

6.1 Polynomial Interpolation

-Lagrange interpolation method

and using the values of y at the interpolation nodes as the coefficients for corresponding $l_i(x)$ to generate the polynomial (linear sum of these terms):

$$p_n(x) = \sum_{i=0}^{n} y_i l_i(x)$$

According to (5), $l_i(x)$ should have n roots, and $l_i(x) = 0$ at every node. Therefore, it must have the form:

$$l_i(x) = \frac{(x-x_0)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)} = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x-x_j}{x_i-x_j} \quad (i=0,1,\cdots,n)$$

so

$$p_n(x) = \sum_{i=0}^{n} y_i l_i(x) = \sum_{i=0}^{n} y_i \left(\prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x-x_j}{x_i-x_j}\right) \qquad (6)$$

(6) is the Lagrange polynomial formula. $l_i(x)$ is called the base function of the interpolation. We often use $L_n(x)$ instead of $p_n(x)$.

6.1 Polynomial Interpolation

-Lagrange interpolation method

Example 1 Consider a function $y = 2^x$ and the following five nodes

x	-2	-1	0	1	2
$y=2^x$	0.25	0.5	1	2	4

(i) take $x_0 = 0, x_1 = 1$ as interpolation nodes and generate the polynomial $L_1(x)$; use $L_1(x)$ to calculate $2^{0.3}$ and estimate the error.

(ii) take $x_0 = -1,\ x_1 = 0,\ x_2 = 1$ as nodes to obtain $L_2(x)$; use $L_2(x)$ to calculate $2^{0.3}$ and estimate the error.

6.1 Polynomial Interpolation

-Lagrange interpolation method

Solution for (i): by (6), the polynomial $L_1(x)$ is

$$L_1(x) = \frac{x-1}{0-1} \bullet 1 + \frac{x-0}{1-0} \bullet 2 = x+1$$

Therefore $$2^{0.3} \approx L_1(0.3) = 1.3$$

and by (4), the error is

$$|R_1(x)| \le \frac{M_2}{2} |(x-x_0)(x-x_1)|$$

where $M_2 = \max_{0 \le x \le 1} |f''(x)|$. Because $f''(x) = (\ln 2)^2 2^x$, we have

$$\max_{0 \le x \le 1} |f''(x)| = 2(\ln 2)^2 = 0.9069$$

so that

$$|R_1(0.3)| \le \frac{0.9069}{2} |(0.3-0)(0.3-1)| = 0.09522$$

6.1 Polynomial Interpolation

-Lagrange interpolation method

Solution for (ii): similarly by (6), the polynomial $L_2(x)$ is

$$L_2(x) = \frac{(x-0)(x-1)}{(-1-0)(-1-1)} \bullet 0.5 + \frac{(x+1)(x-1)}{(0+1)(0-1)} \bullet 1 + \frac{(x-0)(x+1)}{(1-0)(1+1)} \bullet 2$$

$$= 0.25x^2 + 0.75x + 1$$

Therefore $2^{0.3} \approx L_2(0.3) = 1.248$

and by (4), the error is

$$|R_2(x)| \leq \frac{M_3}{6} |(x-x_0)(x-x_1)(x-x_2)|$$

where $M_3 = \max_{-1 \leq x \leq 1} |f'''(x)| = 2(\ln 2)^3 = 0.6660$, we have

so that

$$|R_2(0.3)| \leq \frac{0.6660}{6} |(0.3+1)(0.3-0)(0.3-1)| = 0.03030$$

6.2 Polynomial Interpolation

-- Newton interpolation – divided difference

The Lagrange interpolation formula is neat and symmetric in form, and it is easy to get from the base function. It is easy for programming and also convenient for theoretical analysis. But in some situation at which when we need to add some more nodes (points), the formula appears not convenient as we have to change all the base functions $l_i(x)$ $(i = 0,1,\mathsf{L}\ ,n)$ accordingly. This is obviously not pratical. However, if we change the formula to

$$p_1(x) = y_0 + \frac{(y_1 - y_0)}{(x_1 - x_0)}(x - x_0) \qquad (7)$$

the problem may be solved. We then derive another form of interpolation. Before deriving the formula, we first introduce some concepts called **Divided Difference**, and Difference which will be used in the formula.

6.2 Polynomial Interpolation

- Newton interpolation – divided difference

Let $f(x)$ be a function whose values are known at a set of points $x_0, x_1, \cdots, x_n$. Assuming that these points are distinct. We call

$$f[x_i, x_j] = \frac{f(x_i) - f(x_j)}{x_i - x_j} \quad (i \neq j)$$

the first order divided difference at points x_i, x_j. The higher order divided difference is similar, We define

$$f[x_i, x_j, x_k] = \frac{f[x_i, x_j] - f[x_j, x_k]}{x_i - x_k}$$

as the second order divided difference at points x_i, x_j, x_k.

Generally, the following expression

$$f[x_0, x_1, \cdots, x_k] = \frac{f[x_0, x_1, \cdots, x_{k-1}] - f[x_1, x_2, \cdots, x_k]}{x_0 - x_k}$$

is the k-order divided difference at points $x_0, x_1, \cdots, x_k$.

6.2 Polynomial Interpolation

- Newton interpolation – divided difference

The didvided difference has the following natures

(i) The linearity: if $f(x) = a\phi(x) + b\psi(x)$, then for any k, we have

$$f[x_0, x_1, \cdots, x_k] = a\phi[x_0, x_1, \cdots, x_k] + b\psi[x_0, x_1, \cdots, x_k]$$

(ii) k-order divided difference can be expressed by the linear sum of $f(x_0), f(x_1), \cdots, f(x_k)$, i.e.

$$f[x_0, x_1, \cdots, x_k] = \sum_{i=0}^{k} \frac{f(x_i)}{\omega'_{k+1}(x_i)}$$

where $\omega'_{k+1}(x_i) = \prod_{\substack{j=0 \\ j \neq i}}^{k} (x_i - x_j)$.

6.2 Polynomial Interpolation

- Newton interpolation – divided difference

Proof of (ii):

This is obviously true when $k=1$, because

$$f[x_0, x_1] = \frac{f(x_0)}{\omega_2'(x_0)} + \frac{f(x_1)}{\omega_2'(x_1)}$$

where $\omega_2'(x_0) = (x_0 - x_1)$, $\omega_2'(x_1) = (x_1 - x_0)$

$$f[x_0, x_1] = \frac{f(x_0)}{(x_0 - x_1)} + \frac{f(x_1)}{(x_1 - x_0)}$$

$$= \frac{f(x_0) - f(x_1)}{x_0 - x_1}$$

6.2 Polynomial Interpolation

- Newton interpolation – divided difference

When $k = 2$, we have

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

$$= \frac{1}{x_0 - x_2}\left[\frac{f(x_0) - f(x_1)}{x_0 - x_1} - \frac{f(x_1) - f(x_2)}{x_1 - x_2}\right]$$

$$= \frac{f(x_0)}{(x_0 - x_1)(x_0 - x_2)} + \frac{f(x_1)}{(x_1 - x_0)(x_1 - x_2)} + \frac{f(x_2)}{(x_2 - x_0)(x_2 - x_1)}$$

This nature shows that the divided difference is not effected by the order of nodes. This is called the symmetry of the divided difference, i.e.

$$f[x_i, x_j] = f[x_j, x_i]$$

$$f[x_i, x_j, x_k] = f[x_i, x_k, x_j] = f[x_j, x_i, x_k]$$

6.2 Polynomial Interpolation

- Newton interpolation – divided difference

(*iii*) If $f(x)$ is a n-polynomial, then the first-order divided difference of $f(x)$ is an (n-1)-polynomial. In fact, if $f(x)$ is an n-polynomial, the function $P(x) = f(x) - f(x_i)$ is also an n-polynomial, and $P(x_i) = 0$. $P(x)$ can be factorized as

$$P(x) = (x - x_i)P_{n-1}(x)$$

where $P_{n-1}(x)$ is a (n-1)-polynomial. Therefore

$$f[x, x_i] = \frac{f(x) - f(x_i)}{(x - x_i)} = \frac{(x - x_i)P_{n-1}(x)}{(x - x_i)} = P_{n-1}(x)$$

(*iv*) If $f^{(n)}(x)$ exists on $[a, b]$ and $x_i \in [a, b]$ $(i = 0, 1, \text{L}, n)$, then the divided difference is related with $f^{(n)}(x)$ by the following formula

$$f[x_0, \text{L}, x_n] = \frac{f^{(n)}(\xi)}{n!} \qquad \xi \in [a, b]$$

6.2 Polynomial Interpolation

- Newton interpolation – divided difference

Using the concept of the divided difference we can write the formula (7) as

$$p_1(x) = f(x_0) + (x\text{-}x_0) f[x_0, x_1] \qquad (8)$$

Formula (8) is called one-order Newton Interpolation.

In general, if $x \in [a,b]$, by the definition of divided difference, we have

$$f(x) = f(x_0) + (x\text{-}x_0) f[x, x_0]$$

$$f[x, x_0] = f[x_0, x_1] + (x\text{-}x_1) f[x, x_0, x_1]$$

$$f[x, x_0, x_1] = f[x_0, x_1, x_2] + (x\text{-}x_2) f[x, x_0, x_1, x_2]$$

M

$$f[x, x_0, x_1, \cdots, x_{n-1}] = f[x_0, x_1, \cdots, x_n] + (x\text{-}x_n) f[x, x_0, x_1, \cdots, x_n]$$

6.2 Polynomial Interpolation

- Newton interpolation – divided difference

$$\Rightarrow f(x) = f(x_0) + (x-x_0)f[x_0,x_1] + (x-x_0)(x-x_1)f[x_0,x_1,x_2] + \cdots$$
$$+ (x-x_0)(x-x_1)(x-x_2)\cdots(x-x_{n-1})f[x_0,x_1,\cdots,x_n]$$
$$+ (x-x_0)(x-x_1)(x-x_2)\cdots(x-x_n)f[x,x_0,x_1,\cdots,x_n]$$
$$= N_n(x) + R_n(x)$$

where $N_n(x) = f(x_0) + (x-x_0)f[x_0,x_1] + (x-x_0)(x-x_1)f[x_0,x_1,x_2] + \cdots$
$$+ (x-x_0)(x-x_1)(x-x_2)\cdots(x-x_{n-1})f[x_0,x_1,\cdots,x_n]$$

$$R_n(x) = (x-x_0)(x-x_1)(x-x_2)\cdots(x-x_n)f[x,x_0,x_1,\cdots,x_n]$$
$$= \omega_{n+1}(x)f[x,x_0,x_1,\cdots,x_n] \qquad (9)$$

Obviously, $R_n(x_i) = \omega_{n+1}(x_i)f[x_i,x_0,x_1,\cdots,x_n] = 0 \quad (i = 0,1,2,\cdots,n)$

$N_n(x)$ is called the Newton Interpolation polynomial.

6.2 Polynomial Interpolation

- Newton interpolation – divided difference

The advantage of this fomula is that when we add one more node, we need to add one more term only in the formula. It is more convenient for calculation, easy for programming, and it costs less computing effort.

In fact, $N_n(x) = L_n(x)$, they are only different in expression form. Therefore their residual terms are the same.

$$R_n(x) = \omega_{n+1}(x) f[x, x_0, x_1, \cdots, x_n]$$

$$= \frac{f^{(n+1)}(\xi)}{(n+1)!} \omega_{n+1}(x) \quad \xi \in [a, b]$$

The residual term (9) for Newton Interpolation is more general than the residual (4) for the Lagrange interpolation. It allows to calcultate the error of interpolation from nodes values even when we don't know the value of $f^{(n+1)}$.

6.2 Polynomial Interpolation

- **Newton interpolation – divided difference**

Newton interpolation can be calculated using Table 6-1

x_i	$f(x_i)$	$k=1$	$k=2$	…	$k=n$	
x_0	$f(x_0)$					1
x_1	$f(x_1)$	$f[x_0,x_1]$				$x - x_0$
x_2	$f(x_2)$	$f[x_1,x_2]$	$f[x_0,x_1,x_2]$			$(x - x_0)(x - x_1)$
x_3	$f(x_3)$	$f[x_2,x_3]$	$f[x_1,x_2,x_3]$			$(x - x_0)(x - x_1)\ (x - x_2)$
x_4	$f(x_4)$	$f[x_3,x_4]$	$f[x_2,x_3,x_4]$	…		$\prod(x - x_j)$
x_n	$f(x_n)$	$f[x_{n-1}, x_n]$	$f[x_{n-2}, x_{n-1}, x_n]$	…	$f[x_0,x_1, \ldots,x_n]$	

- Newton interpolation – divided difference

Example 2 The function table is given for function $f(x) = Shx$.
Using the Newton interpolation formula to calculate $f(0.596) = Sh(0.596)$.

Table 6-2 Calculations of divided difference for function $f(x) = sh(x)$.

x_i	$f(x_i)$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	
0.40	0.41075						1
0.55	0.57815	1.11600					$x - 0.40$
0.65	0.69675	1.18600	0.28000				$(x - 0.40)(x - 0.55)$
0.80	0.88811	1.27573	0.35893	0.19733			$(x - 0.40)(x - 0.55)$ $(x - 0.65)$
0.90	1.02652	1.38410	0.43348	0.21300	0.03134		$(x - 0.40)(x - 0.55)$ $(x - 0.65)$ $(x - 0.80)$
1.05	1.25382	1.51533	0.52493	0.22863	0.03126	-0.00012	$(x - 0.40)(x - 0.55)$ $(x - 0.65)$ $(x - 0.80)$ $(x - 0.90)$

6.2 Polynomial Interpolation

- Newton interpolation – divided difference

It is observed that the fourth divided difference is almost constant and the fifth divided difference is near zero. Therefore we use fourth Newton interpolation formula to do the calculation.

$N_4(x)=0.41075$

$+1.11600(x-0.40)$

$+0.28000(x-0.40)(x-0.55)$

$+0.19733(x-0.40)(x-0.55)(x-0.65)$

$+0.03134(x-0.40)(x-0.55)(x-0.65)(x-0.80)$

And we have:

$f(0.596)\approx N_4(0.596)=0.63192$

The truncated error is

$$|R_4(x)|=|f[x,x_0,\ldots,x_4]\,\omega_5(0.596)| \approx |f[x_0,\ldots,x_4,x_5]\,\omega_5(0.596)| \le 3.63\times10^{-9}$$

The truncated error is very small and can be ignored.

6.2 Polynomial Interpolation*

- difference and the interpolation for even spaced nodes

In practice, we often meet some situation in which the nodes are even spaced. In such case, the Newton interpolation can be further simplified, and the divided difference between nodes can be expressed using the difference.

Let $f_k = f(x_k)$ be the function, $x_k = x_0 + kh$ $(k = 0,1,...,n)$ are even spaced, and the distance between the nodes is constant. The difference is definded by

$$\Delta f_k = f_{k+1} - f_k \quad (k = 0,1,...,n-1)$$

$$\nabla f_k = f_k - f_{k-1}$$

$$\delta f_k = f(x_k + \frac{h}{2}) - f(x_k - \frac{h}{2})$$

They are first order **upwind difference**, first order **downwind difference** and first order **central difference**, respectively, and the second order difference is defined

$$\Delta^2 f_k = \Delta f_{k+1} - \Delta f_k \quad (k = 0,1,...,n-1)$$

In general, m-order upwind and downwind differences are defined by

$$\Delta^m f_k = \Delta^{m-1} f_{k+1} - \Delta^{m-1} f_k \quad (k = 0,1,...,n-1)$$

$$\nabla^m f_k = \nabla^{m-1} f_k - \nabla^{m-1} f_{k-1} \quad (k = 0,1,...,n-1)$$

6.2 Polynomial Interpolation

- difference and the interpolation for even spaced nodes

It can be proved that the difference is related with the divided difference by the following formula, for upwind difference:

$$f[x_i, x_{i+1}, ..., x_{i+m}] = \frac{1}{m!h^m}\Delta^m f_i, \quad m = 1, ..., n$$

and for downwind difference:

$$f[x_i, x_{i-1}, ..., x_{i-m}] = \frac{1}{m!h^m}\nabla^m f_i, \quad m = 1, ..., n$$

Besides, the difference is related with $f^{(n)}(x)$ by the following formula

$$\Delta^n f_i = h^n f^{(n)}(\xi) \qquad \xi \in (x_i, x_{i+n})$$

(Prove it by yourself!)

6.2 Polynomial Interpolation

- difference and the interpolation for even spaced nodes

For even spaced nodes, and $x = x_0 + th \quad (0 \le t \le 1)$, the Newton Interpolation can be written as

$$N_n(x_0 + th) = f_0 + t\Delta f_0 + \frac{t(t-1)}{2!}\Delta^2 f_0 + \ldots + \frac{t(t-1)\ldots(t-n+1)}{n!}\Delta^n f_0$$

This is the Newton upwind interpolation formula. The truncated error is

$$R_n(x_0 + th) = \frac{t(t-1)\ldots(t-n)}{(n+1)!}h^{n+1}f^{(n+1)}(\xi) \quad \xi \in (x_0, x_n)$$

Similarly, let $x = x_n + th \quad (-1 \le t \le 0)$, the Newton downwind interpolation formula is

$$N_n(x_n + th) = f_n + t\nabla f_n + \frac{t(t+1)}{2!}\nabla^2 f_n + \ldots + \frac{t(t+1)\ldots(t+n-1)}{n!}\nabla^n f_n$$

The remainder of the method is

$$R_n(x_n + th) = \frac{t(t+1)\ldots(t+n)}{(n+1)!}h^{n+1}f^{(n+1)}(\xi) \quad \xi \in (x_0, x_n)$$

6.2 Polynomial Interpolation

- difference and the interpolation for even spaced nodes

Example 3 Let $y = \cos x$, the function table is as follows. Find the $f(0.048)$ using the fourth order Newton difference interpolation.

x	0.0	0.1	0.2	0.3	0.4	0.5	0.6
cosx	1.00000	0.99500	0.98007	0.95534	0.92106	0.87758	0.82534

Solution: according to above function table, we make the following table for difference.

$f_i=f(x_i)$	k=1	k=2	k=3	k=4	k=5	
1.00000						
0.99500	-0.00500					
0.98007	-0.01493	-0.00993				
0.95534	-0.02473	-0.00980	0.00013			
0.92106	-0.03428	-0.00955	0.00025	0.00012		
0.87758	-0.04348	-0.00920	0.00035	0.00010	-0.00002	
0.82534	-0.05224	-0.00876	0.00044	0.00009	-0.00001	

6.2 Polynomial Interpolation

- difference and the interpolation for even spaced nodes

We use the upwind Newton Interpolation formula for the calculation.

$x = 0.048,\ h = 0.1,\ t = \dfrac{x - 0}{h} = 0.48,$ making use of the above table, we have

$$N_4(0.048) = 1.00000 + 0.48 \times (-0.00500) + \frac{0.48 \times (0.48 - 1)}{2} \times (-0.00993)$$
$$+ \frac{0.48 \times (0.48 - 1) \times (0.48 - 2)}{3!} \times 0.00013$$
$$+ \frac{0.48 \times (0.48 - 1) \times (0.48 - 2) \times (0.48 - 3)}{4!} \times 0.00012$$
$$= 0.99885 \approx \cos(0.048)$$

The truncated error is estimated by

$$| R_4(0.048) | \leq \frac{M_5}{5!} | t(t-1)(t-2)(t-3)(t-4) | h^5 \leq 1.5487 \times 10^{-7}$$

where $M_5 = | \sin 0.6 | \leq 0.565.$

6.1 Polynomial Interpolation

-Hermite interpolation method

The term **Hermite interpolation** refers to the interpolation of a function and some of its derivatives at a set of nodes.

Assume that at nodes $a \le x_0 < x_1 < \cdots < x_n \le b, y_j = f(x_j), m_j = f'(x_j)\ (j = 0,1,\mathrm{L}\ ,n)$, we require a polynomial $H(x)$ of least degree that interpolates f and its derivatives f' at these $n+1$ nodes:

$$H(x_j) = y_j, H'(x_j) = m_j\ \ (j = 0,1,\mathrm{L}\ ,n). \qquad (3.1)$$

There are $2n+2$ conditions, which can determine a polynomial of degree no more than $2n+1$, we denote it by $H_{2n+1}(x) = H(x)$:

$$H_{2n+1}(x) = a_0 + a_1 x + \cdots + a_{2n+1} x^{2n+1},$$

and we need to determine $2n+2$ coefficients $a_0,\ a_1, \cdots, a_{2n+1}$. We use the base function method. First we find the $2n+2$ interpolation base function $\alpha_j(x)$ and $\beta_j(x)\ (j = 0,1,\mathrm{L}\ ,n)$, which are polynomials of degree $2n+1$:

$$\begin{cases} \alpha_j(x_k) = \delta_{jk}, \alpha_j'(x_k) = 0 \\ \beta_j(x_k) = 0, \beta_j'(x_k) = \delta_{jk} \end{cases} \quad \delta_{jk} = \begin{cases} 0, & j \ne k \\ 1, & j = k \end{cases} \quad (j,k = 0,1,\mathrm{L}\ ,n). \qquad (3.2)$$

6.1 Polynomial Interpolation

-Hermite interpolation method

Thus the polynomial $H(x)=H_{2n+1}(x)$ satisfying (3.1) can be represented as

$$H_{2n+1}(x)=\sum_{j=0}^{n}\left[y_j\alpha_j(x)+m_j\beta_j(x)\right]. \qquad (3.3)$$

Obviously, $H_{2n+1}(x_k)=y_k, H'_{2n+1}(x_k)=m_k \ (k=0,1,\text{L}\ ,n)$. Now we try to find $\alpha_j(x)$ and $\beta_j(x)$ satisfying (3.2), and we use the Lagrange interpolation base function $l_j(x)$.

Let

$$\alpha_j(x)=(ax+b)\ l_j^{\,2}(x),$$

where

$$l_j(x)=\frac{(x-x_0)\text{L}\ (x-x_{j-1})(x-x_{j+1})\text{L}\ (x-x_n)}{(x_j-x_0)\text{L}\ (x_i-x_{j-1})(x_j-x_{j+1)})\text{L}\ (x_j-x_n)}.$$

According to (3.2),

$$\begin{cases}\alpha_j(x_j)=(ax_j+b)\ l_j^{\,2}(x_j)=1\\ \alpha_j'(x_j)=l_j(x_j)\left[al_j(x_j)+2(ax_j+b)l_j'(x_j)\ \right]=0\end{cases}$$

then,

$$\begin{cases}ax_j+b=1\\ a+2l_j'(x_j)\ =0\end{cases}$$

Thus we obtain

$$a=-2l_j'(x_j),\quad b=1+2x_jl_j'(x_j).$$

6.1 Polynomial Interpolation

-Hermite interpolation method

Since

$$l_j(x)=\frac{(x-x_0)\cdots(x-x_{j-1})(x-x_{j+1})\cdots(x-x_n)}{(x_j-x_0)\cdots(x_j-x_{j-1})(x_j-x_{j+1})\cdots(x_j-x_n)},$$

then we can obtain

$$l_j'(x_j)=\sum_{\substack{k=0\\k\neq j}}^{n}\frac{1}{(x_j-x_k)},$$

thus

$$\alpha_j(x)=\left(1-2(x-x_j)\sum_{\substack{k=0\\k\neq j}}^{n}\frac{1}{(x_j-x_k)}\right)l_j^{\,2}(x). \quad (3.4)$$

Similarly, we can obtain

$$\beta_j(x)=(x-x_j)\,l_j^{\,2}(x). \quad (3.5)$$

Question Whether the polynomial satisfying (3.1) is unique?

It can be proved that if $f^{(2n+2)}(x)$ exists on (a,b), the residual term is

$$R(x)=f(x)-H_{2n+1}(x)=\frac{f^{(2n+2)}(\xi)}{(2n+2)!}\omega_{n+1}^2(x). \quad (3.6)$$

where $\xi\in(a,b)$ and depends on x. (**Exercise**: prove this by yourself.)

6.1 Polynomial Interpolation

-Hermite interpolation method

When $n=1$, the nodes are x_k and x_{k+1}, and the interpolation polynomial $H_3(x)$ satisfies

$$\begin{cases} H_3(x_k)=y_k, H_3(x_{k+1})=y_{k+1} \\ H_3'(x_k)=m_k, H_3'(x_{k+1})=m_{k+1} \end{cases} \tag{3.7}$$

According to (3.4) and (3.5), we can get

$$\begin{cases} \alpha_k(x)=\left(1+2\dfrac{x-x_k}{x_{k+1}-x_k}\right)\left(\dfrac{x-x_{k+1}}{x_k-x_{k+1}}\right)^2 \\ \alpha_{k+1}(x)=\left(1+2\dfrac{x-x_{k+1}}{x_k-x_{k+1}}\right)\left(\dfrac{x-x_k}{x_{k+1}-x_k}\right)^2 \end{cases} \tag{3.8}$$

$$\begin{cases} \beta_k(x)=(x-x_k)\left(\dfrac{x-x_{k+1}}{x_k-x_{k+1}}\right)^2 \\ \beta_{k+1}(x)=(x-x_{k+1})\left(\dfrac{x-x_k}{x_{k+1}-x_k}\right)^2 \end{cases} \tag{3.9}$$

Thus

$$H_3(x)=y_k\alpha_k(x)+y_{k+1}\alpha_{k+1}(x)+m_k\beta_k(x)+m_{k+1}\beta_{k+1}(x), \tag{3.10}$$

the residual term is

$$R_3(x)=f(x)-H_3(x)=\frac{f^{(4)}(\xi)}{4!}(x-x_k)^2(x-x_{k+1})^2, \xi\in(x_k,x_{k+1}).$$

6.1 Polynomial Interpolation

-Hermite interpolation method

Example Find the interpolation polynomial which satisfies $P(x_j)=f(x_j)$ $(j=0,1,2)$ and $P'(x_1)=f'(x_1)$, and give its residual term.

Solution : By the given conditions, an interpolation polynomial of degree no more than 3 can be determined. Because the polynomial passes through three points $(x_0,f(x_0)),(x_1,f(x_1)),(x_2,f(x_2))$, it can be represented as

$$P(x)=f(x_0)+f[x_0,x_1](x-x_0)+f[x_0,x_1,x_2](x-x_0)(x-x_1)+A(x-x_0)(x-x_1)(x-x_2),$$

where A is a coefficient to be determined. By $P'(x_1)=f'(x_1)$, we can obtain

$$A=\frac{f'(x_1)-f[x_0,x_1]-(x_1-x_0)f[x_0,x_1,x_2]}{(x_1-x_0)(x_1-x_2)}.$$

To find the residual term, let

$$R(x)=f(x)-P(x)=k(x)(x-x_0)(x-x_1)^2(x-x_2),$$

where $k(x)$ is a function to be determined. Construct

$$\varphi(t)=f(t)-P(t)-k(x)(t-x_0)(t-x_1)^2(t-x_2).$$

Obviously, $\varphi(x_j)=0$ $(j=0,1,2)$. And $\varphi'(x_1)=0$, $\varphi(x)=0$, thus $\varphi(t)$ has at least 5 zero points on (a,b).

Apply the Rolle theorem repeatedly, $\varphi^{(4)}(t)$ has at least one zero point ξ,

$$\varphi^{(4)}(\xi)=f^{(4)}(\xi)-4!k(x)=0,$$

thus

$$k(x)=\frac{1}{4!}f^{(4)}(\xi),$$

and the residual term

$$R(x)=\frac{1}{4!}f^{(4)}(\xi)(x-x_0)(x-x_1)^2(x-x_2),$$

where ξ is between x_0,x_1,x_2 and x.

Exercises

*Ex*1. For a given function $f(x) = (4x - 7)/(x - 2)$, let $x_0 = 1.7, x_1 = 1.8, x_2 = 1.9, x_3 = 2.1$.

(1) Construct an interpolation polynomial of degree at most two to approximate f(1.75) using points x_0, x_1 *and* x_2;

(2) Construct an interpolation polynomial of degree three to approximate f(1.75) and f(2.000) using points x_0, x_1, x_2 *and* x_3.

*Ex*2. *S*uppose that f(x) has second - order continued derivative in the interval [a, b], and f(a) = f(b) = 0. Prove that

$$\max_{a \le x \le b} |f(x)| \le \frac{1}{8}(b-a)^2 \max_{a \le x \le b} |f''(x)|$$

*Ex*3. Construct the Lagrange interpolating polynomials for the following functions, and find a bound for the absolute error on the interval $[x_0, x_n]$

(1) $f(x) = e^{2x}\cos 3x,\ x_0 = 0,\ x_1 = 0.3,\ x_2 = 0.6,\ n = 2$

(2) $f(x) = \cos x + \sin x,\ x_0 = 0,\ x_1 = 0.25,\ x_2 = 0.5, x_3 = 1.0,\ n = 3$

Exercises

*Ex*4. A table for function sinx is given as follows:

x	0.7	0.9	1.1	1.3	1.5	1.7
sinx	0.6442	0.7833	0.8912	0.9636	0.9975	0.9917

Make a divided difference table accordingly, and construct Newton interpolation polynomials of two-point, three-point and four-point to approximate sin(1.0).

*Ex*5. Using the same function table as in Ex4,

(1) construct a table of upwind difference and a table of downwind difference respectively;

(2) construct two-point, three-point and four-point Newton upwind interpolation formula to approximate sin(0.74), and estimate the relevant truncation errors ;

(3) construct two-point, three-point and four-point Newton downwind interpolation formula to approximate sin(1.6), and estimate the relevant truncation errors ;

Exercises

*Ex*6. Find a polynomial $P(x)$ of degree no more than 3 that assumes $P(0)=0, P'(0)=1, P(1)=1, P'(1)=2.$

*Ex*7. Find a polynomial $P(x)$ of degree no more than 4 that assumes $P(0)=P'(0)=0, P(1)=P'(1)=1, P(2)=1.$