Numerical Analysis

Chapter Six: Approximating Functions

主讲人　朱亚菲
幻灯片制作　朱亚菲

中国海洋大学　信息科学与工程学院

2013 年 12 月 6 日

Contents

Contents

Contents

Introduction

1. Find simple approximate representations for known functions.
2. Interpolation and extrapolation, estimating unknown function values from known values at nearby points.

x	x_0	x_1	x_2	...	x_n
y	y_0	y_1	y_2	...	y_n

- On one hand, interpolation of smooth functions gives accurate approximations.
- On the other hand, we can interpolate and extrapolate using our approximating functions.

Contents

Vandermonde Theory

The most direct approach uses the vanderMonde matrix. We can require our polynomial to be expressed in powers of x:

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

The interpolation conditions, $p(x_i) = y_i$ for $0 \leqslant i \leqslant n$, lead to a system of $n+1$ linear equations for determining $a_0, a_1, \ldots, a_n$. This system has the form

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Vandermonde Theory

- The Vandermonde matrix is nonsingular because the system has a unique solution for any choice of $y_0, y_1, \ldots, y_n$.
- However, the Vandermonde matrix is often ill conditioned, and the coefficients a_i may therefore be inaccurately determined by solving the System.
- Therefore, this approach is not recommended.

Contents

Polynomial Interpolation

THEOREM 1 (Theorem on polynomial Interpolation)

If $x_0, x_1, \ldots, x_n$ are distinct real numbers, then for arbitrary values $y_0, y_1, \ldots, y_n$, there is a unique polynomial p_n of degree at most n such that

$$p_n(x_i) = y_i \quad (0 \leqslant i \leqslant n)$$

Newton Form of the Interpolation Polynomial

Suppose that we hane obtained a polynomial p_{k-1} of degree $\leqslant k-1$ with $p_{k-1}(x_i)=y_i$ for $0\leqslant i\leqslant k-1$. We try to construct p_k in the form

$$p_k(x)=p_{k-1}(x)+c(x-x_0)(x-x_1)\cdots(x-x_{k-1})$$

Note that p_k interpolates the data that p_{k-1} interpolates because

$$p_k(x_i)=p_{k-1}(x_i)=y_i \qquad (0\leqslant i\leqslant k-1)$$

Now we determine the unknown coefficient c from the condition $p_k(x_k)=y_k$. This leads to the equation

$$p_{k-1}(x_k)+c(x_k-x_0)(x_k-x_1)\cdots(x_k-x_{k-1})=y_k$$

If $x_0,x_1,\ldots,x_k$ are prescribed, then for arbitrary values $y_0,y_1,\ldots,y_k$, there is a polynomial p of degree at most k having the form

$$p_k(x)=c_0+c_1(x-x_0)+c_2(x-x_0)(x-x_1)+\ldots+c_k(x-x_0)\ldots(x-x_{k-1})$$

Newton Form of the Interpolation Polynomial

To obtain $u = p_k(t)$ for a prescribed value of t, assuming that the coefficients $c_0, c_1, \ldots, c_k$ are known, an efficient method called **nested multiplication** or **Horner's algorithm** is used.

$u \leftarrow c_k$
for $i = k - 1$ **to** 0 **step** -1 **do**
  $u \leftarrow (t - x_i)u + c_i$
end do

Newton Form of the Interpolation Polynomial

An algorithm to compute $c_0, c_1, \ldots, c_n$ from the table of values for $x_0, x_1, \ldots, x_n$ and $y_0, y_1, \ldots, y_n$ follows:

$c_0 \leftarrow y_0$
for $k = 1$ **to** n **do**
  $d \leftarrow x_k - x_{k-1}$
  $u \leftarrow c_{k-1}$
  for $i = k - 2$ **to** 0 **step -1 do**
    $u \leftarrow u(x_k - x_i) + c_i$
    $d \leftarrow d(x_k - x_i)$
  end do
  $c_k \leftarrow (y_k - u)/d$
end do

Newton Form of the Interpolation Polynomial

```
function c=Coefficient(n,x,y)
c(1)=y(1);
for k=2:n+1
    d=x(k)-x(k-1);
    u=c(k-1);
    for i=k-2:-1:1
        u=u*(x(k)-x(i))+c(i);
        d=d*(x(k)-x(i));
    end
    c(k)=(y(k)-u)/d;
end
```

Divided Differences

For $q_j(x) = (x - x_0)(x - x_1)(x - x_2) \cdots (x - x_{j-1})$
The Newton form is $p(x) = \sum_{j=0}^{n} c_j q_j(x)$
The interpolation conditions $p(x_i) = f(x_i)$ give rise to a linear system of equations for the determination of the unknown coefficients, c_j:

$$\sum_{j=0}^{n} c_j q_j(x_i) = f(x_i) \quad (0 \leqslant i \leqslant n)$$

In this system of equations, the coefficient matrix is an $(n+1) \times (n+1)$ matrix A whose elements are

$$a_{ij} = q_j(x_i) \quad (0 \leqslant i, j \leqslant n)$$

Divided Differences

For example, consider the case of three nodes with

$$\begin{aligned} p_2(x) &= c_0 q_0(x) + c_1 q_1(x) + c_2 q_2(x) \\ &= c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) \end{aligned}$$

Setting $x = x_0, x = x_1$, and $x = x_2$, we have a lower triangular system

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & (x_1 - x_0) & 0 \\ 1 & (x_2 - x_0) & (x_2 - x_0)(x_2 - x_1) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \end{bmatrix}$$

We can see that c_0 depends on only $f(x_0)$, that c_1 depends on $f(x_0)$ and $f(x_1)$, and so on. Thus, c_n depends on f at $x_0, x_1, \dots, x_n$.

$$c_n = f[x_0, x_1, \dots, x_n]$$

Divided Differences

THEOREM 2 (Theorem on Higher-Order Divided Differences)

Divided differences satisfy the equation

$$f[x_0, x_1, \ldots, x_n] = \frac{f[x_1, x_2, \ldots, x_n] - f[x_0, x_1, \ldots, x_{n-1}]}{x_n - x_0}$$

Divided Differences

In the formulas, $x_0, x_1, x_2, \ldots$ can be interpreted as independent variables. Because of that, we also have equations such as

$$f[x_i, x_{i+1}, \ldots, x_{i+j}] = \frac{f[x_{i+1}, x_{i+2}, \ldots, x_{i+j}] - f[x_i, x_{i+1}, \ldots, x_{i+j-1}]}{x_{i+j} - x_i}$$

If a table of function values $(x_i, f(x_i))$ is given, we can construct from it a table of divided differences. This is customarily laid out in the following form, where differences of orders 0,1,2, and 3 are shown in each successive column:

$$\begin{array}{cc|lll}
x_0 & f[x_0] & f[x_0, x_1] & f[x_0, x_1, x_2] & f[x_0, x_1, x_2, x_3] \\
x_1 & f[x_1] & f[x_1, x_2] & f[x_1, x_2, x_3] & \\
x_2 & f[x_2] & f[x_2, x_3] & & \\
x_3 & f[x_3] & & &
\end{array}$$

Algorithm for Divided Differences

An algorithm for computing a divided difference table can be very efficient and is recommended as the best means for producing an interpolating polynomial.

$$
\begin{array}{cc|cccccc}
x_0 & c_{00} & c_{01} & c_{02} & c_{03} & \cdots & c_{0,n-1} & c_{0,n} \\
x_1 & c_{10} & c_{11} & c_{12} & c_{13} & \cdots & c_{1,n-1} & \\
x_2 & c_{20} & c_{21} & c_{22} & c_{23} & & & \\
\vdots & \vdots & \vdots & \vdots & & & & \\
\vdots & \vdots & \vdots & & & & & \\
x_{n-1} & c_{n-1,0} & c_{n-1,1} & & & & & \\
x_n & c_{n0} & & & & & &
\end{array}
$$

Algorithm for Divided Differences

It is clear that we have set $c_{ij} = f[x_i, x_{i+1}, \ldots, x_{i+j}]$
An algorithm is obtained from a direct translation of Equation

$$f[x_i, x_{i+1}, \ldots, x_{i+j}] = \frac{f[x_{i+1}, x_{i+2}, \ldots, x_{i+j}] - f[x_i, x_{i+1}, \ldots, x_{i+j-1}]}{x_{i+j} - x_i}$$

for $j = 1$ **to** n **do**
    for $i = 0$ **to** $n - j$ **do**
        $c_{ij} \leftarrow (c_{i+1,j-1} - ci, j - 1)/(xi + j - x_i)$
    end do
end do

Algorithm for Divided Differences

Another algorithm can be designed that uses less storage space in the computer.

for $i = 0$ **to** n **do**
$d_i \leftarrow f(x_i)$ **end do for** $j = 1$ **to** n **do**
for $i = n$ **to** j **step -1 do**
$d_i \leftarrow (d_i - di - 1)/(x_i - x_{i-j})$
end do
end do

Algorithm for Divided Differences

```
function c=dividedDifferences(n,c,x)
for j=2:n
    for i=1:(n-j+1)
        c(i,j)=(c(i+1,j-1)-c(i,j-1))/(x(i+j-1)-x(i));
    end
end
```

Algorithm for Divided Differences

```
function d=lessStorage(n,x,y)
for i=1:n
    d(i)=y(i);
end
for j=2:n
    for i=n:-1:j
        d(i)=(d(i)-d(i-1))/(x(i)-x(i-j+1));
    end
end
```

Contents

Lagrange Form of the Interpolation Polynomial

The alternative method will express p in the form

$$p(x) = y_0 l_0(x) + y_1 l_1(x) + \ldots + y_n l_n(x) = \sum_{k=0}^{n} y_k l_k(x)$$

Here $l_0, l_1, \ldots, l_n$ are polynomials that depend on the nodes $x_0, x_1, \ldots, x_n$ but not on the ordinates $y_0, y_1, \ldots, y_n$.
Since all the ordinates could be 0 except for a 1 occupying the ith position, we see that $\delta_{ij} = l_i(x_j)$.

Lagrange Form of the Interpolation Polynomial

Let us consider l_0. It is to be a polynomial of degree n that takes the value 0 at $x_1, x_2, \ldots, x_n$ and the value 1 at x_0. Clearly, l_0 must be of the form

$$l_0(x) = c(x - x_1)(x - x_2)\ldots(x - x_n) = c\prod_{j=1}^{n}(x - x_j)$$

The value of c is obtained by putting $x = x_0$, so that

$$1 = c\prod_{j=1}^{n}(x_0 - x_j) \quad \text{and } c = \prod_{j=1}^{n}(x_0 - x_j)^{-1}$$

Hence, we have

$$l_0(x) = \prod_{j=1}^{n}\frac{x - x_j}{x_0 - x_j}$$

Lagrange Form of the Interpolation Polynomial

Each l_i is obtained by similar reasoning, and the general formula is then

$$l_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \quad (0 \leqslant i \leqslant n)$$

For the set of nodes $x_0, x_1, \ldots, x_n$, these polynomials are known as the **cardinal functions**.

Comparisons

the Newton form	the Lagrange form
If more data points are added to the interpolation problem, the coefficients already computed will not have to be changed.	a single set of fixed x_i nodes with many different y_i values associated with them

Contents

Contents

Hermite Interpolation

The term **Hermite Interpolation** refers to the interpolation of a function and some of its derivatives at a set of nodes.
In a Hermite problem, it is assumed that whenever a derivative $p^{(j)}(x_i)$ is to be prescribed (at a node x_i), $p^{(j-1)}(x_i)$, $p^{(j-2)}(x_i)$, ..., $p'(x_i)$, and $p(x_i)$ will also be prescribed. We choose our notation so that at node x_i, k_i interpolatory conditions are prescribed. Let the nodes be $x_0, x_1, \ldots, x_n$, and suppose that at node x_i these interpolation conditions are given:

$$p^{(j)}(x_i) = c_{ij} \qquad (0 \leqslant j \leqslant k_i - 1, 0 \leqslant i \leqslant n)$$

The total number of conditions on p is denoted by $m + 1$, and therefore

$$m + 1 = k_0 + k_1 + \cdots + k_n$$

Hermite Interpolation

THEOREM 3 (Theorem on Hermite Interpolation)

There exists a unique polynomial p in $\prod_m$ fulfilling the Hermite interpolation conditions in Equation $p^{(j)}(x_i) = c_{ij}$

Contents

Newton Divided Difference Method

We begin with a simple case in which a quadratic polynomial p is sought, taking prescribed values:

$$p(x_0) = c_{00} \qquad p'(x_0) = c_{01} \qquad p(x_1) = c_{10}$$

We write the divided difference table in this form:

$$\begin{array}{cc|cc} x_0 & c_{00} & c_{01} & ? \\ x_0 & c_{00} & ? & \\ x_1 & c_{10} & & \end{array}$$

$$f[x_0, x_0, \ldots, x_0] = \frac{1}{k!} f^{(k)}(x_0)$$

Contents

Lagrange Form

- Let the nodes be $x_0, x_1, \ldots, x_n$ and assume that at each node a function value and the first derivative have been prescribed. The polynomial p that we seek must satisfy these equations:

$$p(x_i) = c_{i0} \qquad p'(x_i) = c_{i1} \qquad (0 \leqslant i \leqslant n)$$

In analogy with the Lagrange formula,

$$p(x) = \sum_{i=0}^{n} c_{i0} A_i(x) + \sum_{i=0}^{n} c_{i1} B_i(x)$$

in which A_i and B_i are to be polynomial.

$$\begin{cases} A_i(x_j) = \delta_{ij} \\ A_i'(x_j) = 0 \end{cases} \begin{cases} B_i(x_j) = 0 \\ B_i'(x_j) = \delta_{ij} \end{cases}$$

Lagrange Form

- With the aid of the functions

$$l_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \qquad (0 \leqslant i \leqslant n)$$

A_i and B_i can be defined as follows:

$$\begin{cases} A_i(x) = [1 - 2(x - x_i) l_i'(x_i)] l_i^2(x) & (0 \leqslant i \leqslant n) \\ B_i(x) = (x - x_i) l_i^2(x) & (0 \leqslant i \leqslant n) \end{cases}$$