

CS537-002

**Introduction to Numerical
Methods**

Lecture 3

**Interpolation and Numerical
Differentiation**

Professor Jun Zhang

Department of Computer Science
University of Kentucky
Lexington, KY 40206-0046

September 18, 2008

Polynomial Interpolation

Given a set of discrete values, how can we estimate other values between these data

$$\begin{array}{c|c|c|c|c} x & x_0 & x_1 & \cdots & x_n \\ \hline y & y_0 & y_1 & \cdots & y_n \end{array}$$

The method what we will use is called *polynomial interpolation*.

We assume the data we had are from the evaluation of a *smooth* function. We may be able to use a polynomial $p(x)$ to approximate this function, at least locally.

A condition: the polynomial $p(x)$ takes the given values at the given points (nodes), i.e., $p(x_i) = y_i$ with $0 \leq i \leq n$. The polynomial is said to *interpolate* the *table*, since we do not know the *function*.

Order of Interpolating Polynomial

A polynomial of degree **0**, a constant function, interpolates one set of data

If we have two sets of data, we can have an interpolating polynomial of degree **1**, a linear function

$$\begin{aligned} p(x) &= \left(\frac{x - x_1}{x_0 - x_1} \right) y_0 + \left(\frac{x - x_0}{x_1 - x_0} \right) y_1 \\ &= y_0 + \left(\frac{y_1 - y_0}{x_1 - x_0} \right) (x - x_0) \end{aligned}$$

Review carefully if the *condition* is satisfied

Interpolating polynomials can be written in several forms, the most well known ones are the Lagrange form and Newton form. Each has some advantages

Lagrange Form

For a set of fixed nodes x_0, x_1, \dots, x_n , the *cardinal functions*, l_0, l_1, \dots, l_n , are defined as

$$l_i(x_j) = \delta_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}$$

We can interpolate any function $f(x)$ by the *Lagrange form* of the interpolating polynomial of degree $\leq n$

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

Note that $l_i(x)$ is of order n , so $p_n(x)$ is of order $\leq n$, and

$$p_n(x_j) = \sum_{i=0}^n l_i(x_j) f(x_i) = l_j(x_j) f(x_j) = f(x_j)$$

The (point exact) *condition* is satisfied

Cardinal Functions

The cardinal function is

$$l_i(\mathbf{x}) = \prod_{j \neq i, j=0}^n \left(\frac{\mathbf{x} - \mathbf{x}_j}{\mathbf{x}_i - \mathbf{x}_j} \right) \quad (0 \leq i \leq n)$$

What it looks like

$$l_i(\mathbf{x}) = \left(\frac{\mathbf{x} - \mathbf{x}_0}{\mathbf{x}_i - \mathbf{x}_0} \right) \left(\frac{\mathbf{x} - \mathbf{x}_1}{\mathbf{x}_i - \mathbf{x}_1} \right) \dots$$
$$\left(\frac{\mathbf{x} - \mathbf{x}_{i-1}}{\mathbf{x}_i - \mathbf{x}_{i-1}} \right) \left(\frac{\mathbf{x} - \mathbf{x}_{i+1}}{\mathbf{x}_i - \mathbf{x}_{i+1}} \right) \dots$$
$$\left(\frac{\mathbf{x} - \mathbf{x}_n}{\mathbf{x}_i - \mathbf{x}_n} \right)$$

Note that $l_i(\mathbf{x}_j) = \mathbf{0}$, for $i \neq j$

and $l_i(\mathbf{x}_i) = \mathbf{1}$

Step by Step Construction

For any table of data, we can construct a Lagrange interpolating polynomial. Its evaluation is a little bit costly, but we can always do that. The existence of the interpolating polynomial is guaranteed

Can we construct the interpolating polynomial step by step, or if we discover some new data, can we add those data to make the interpolation more accurate?

We can use Newton form of the interpolating polynomial

Let $p_k(x)$ be an interpolating polynomial for the data set $\{(x_i, y_i)\}$ with $0 \leq i \leq k$ such that $p_k(x_i) = y_i$

Newton Form - Cont.

We want to add another data (x_{k+1}, y_{k+1}) to have a new interpolating polynomial $p_{k+1}(x)$ such that $p_{k+1}(x_i) = y_i$ for $0 \leq i \leq (k + 1)$.

Let

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

Where c is an undetermined constant

Since $p_{k+1}(x_{k+1}) = y_{k+1}$, we have

$$\begin{aligned} p_k(x_{k+1}) + c(x_{k+1} - x_0)(x_{k+1} - x_1) \\ \cdots (x_{k+1} - x_k) = y_{k+1} \end{aligned}$$

We can solve this equation for c , with the condition that x_0, x_1, \dots, x_{k+1} are all distinct

$$c = \frac{y_{k+1} - p_k(x_{k+1})}{(x_{k+1} - x_0)(x_{k+1} - x_1) \cdots (x_{k+1} - x_k)}$$

Uniqueness of Polynomial

Is the interpolating polynomial unique?

If p and q are interpolating polynomials for the data set $\{(x_i, y_i)\}$ for $0 \leq i \leq n$ such that $p(x_i) = q(x_i) = y_i$

Then the polynomial $r(x) = p(x) - q(x)$ of degree at most n is zero at x_0, x_1, \dots, x_n . Note that a polynomial of degree n can have at most n roots, we must have $r(x) = 0$, or $p - q = 0$. Hence $p = q$

The interpolating polynomial is unique

It may be written in different forms

An Example

Find the interpolating polynomial for this table

x	0	1	-1
y	-5	-3	-15

Lagrange form

$$l_0(x) = \frac{(x-1)(x+1)}{(0-1)(0+1)} = -(x-1)(x+1)$$

$$l_1(x) = \frac{(x-0)(x+1)}{(1-0)(1+1)} = \frac{1}{2}x(x+1)$$

$$l_2(x) = \frac{(x-0)(x-1)}{(-1-0)(-1-1)} = \frac{1}{2}x(x-1)$$

The interpolating polynomial is

$$p_2(x) = 5(x-1)(x+1) - \frac{3}{2}x(x+1) - \frac{15}{2}x(x-1)$$

Newton Form

The zeroth order polynomial is

$$p_0(x) = -5$$

Let the 1st order interpolating polynomial be

$$p_1(x) = p_0 + c(x - x_0) = -5 + c(x - 0)$$

We want $p_1(x_1) = -3$, hence $-5 + c(1 - 0) = -3$, we have $c = 2$, it follows that

$$p_1(x) = -5 + 2x$$

Let the 2nd order interpolating polynomial be

$$p_2(x) = p_1(x) + c(x - x_0)(x - x_1)$$

Put $p_2(-1) = -15$, i.e., $-5 + 2(-1) + c(-1 - 0)(-1 - 1) = -15$. We have $c = -4$.

The Newton form of the interpolating polynomial is

$$p_2(x) = -5 + 2x - 4x(x - 1)$$

Nested Form

For easy programming and efficient computation, we can write Newton form of the interpolating polynomial in **nested form**

$$\begin{aligned} p(x) = & a_0 + a_1[(x - x_0)] + a_2[(x - x_0)(x - x_1)] \\ & + a_3[(x - x_0)(x - x_1)(x - x_2)] + \cdots \\ & + a_n[(x - x_0)(x - x_1)\cdots(x - x_{n-1})] \end{aligned}$$

Or, using standard product notations as

$$p(x) = a_0 + \sum_{i=1}^n a_i \left[\prod_{j=1}^{i-1} (x - x_j) \right]$$

Using successive factorization, the nested form is

$$\begin{aligned} p(x) = & a_0 + (x - x_0)(a_1 + (x - x_1)(a_2 + \cdots \\ & + (x - x_{n-1})a_n)) \cdots) \\ = & (\cdots((a_n(x - x_{n-1}) + a_{n-1})(x - x_{n-2}) \\ & + a_{n-2}) \cdots)(x - x_0) + a_0 \end{aligned}$$

Computation Procedure

To evaluate $p(x)$ for a given x , we start from the innermost parentheses, forming successively some intermediate quantities

$$\begin{aligned}v_0 &= a_n \\v_1 &= v_0(x - x_{n-1}) + a_{n-1} \\v_2 &= v_1(x - x_{n-2}) + a_{n-2} \\&\vdots \\v_i &= v_{i-1}(x - x_{n-i}) + a_{n-i} \\&\vdots \\v_n &= v_{n-1}(x - x_0) + a_0\end{aligned}$$

A pseudocode is

```
real array  $(a_i)_{0:n}$ ,  $(x_i)_{0:n}$ 
integer  $i, n$ 
real  $x, v$ 
 $v \leftarrow a_n$ 
for  $i = n-1$  to  $0$  step  $-1$  do
     $v \leftarrow v(x - x_i) + a_i$ 
end for
```

Divided Difference

The coefficients a_i in Newton form of the interpolating need to be computed. A notation is introduced for facilitating such computation

$$a_k = f[x_0, x_1, \dots, x_k]$$

Which is called the **divided difference of order k** for f

Newton form interpolating polynomial is

$$p_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) \\ + \dots + a_n(x - x_0) \cdots (x - x_{n-1})$$

Or written in a compact form

$$p_n(x) = \sum_{i=0}^n a_i \prod_{j=0}^{i-1} (x - x_j)$$

With the convention

$$\prod_{j=0}^{-1} (x - x_j) = 1$$

Computing Coefficients a_i

We want $p_n(x_i) = f(x_i)$. So we have

$$f(x_0) = a_0$$

$$f(x_1) = a_0 + a_1(x_1 - x_0)$$

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

... ..

The solution of this system is $a_0 = f(x_0)$

$$\begin{aligned} a_1 &= \frac{f(x_1) - a_0}{x_1 - x_0} \\ &= \frac{f(x_1) - f(x_0)}{x_1 - x_0} \end{aligned}$$

The divided difference of order 1 is

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

Note that $f[x_0, x_1, \dots, x_k]$ is the coefficient of x^k in the polynomial p_k of degree $\leq k$

Computing Coefficients – Cont.

$$\begin{aligned} a_2 &= \frac{f(x_2) - a_0 - a_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} \\ &= \frac{f(x_2) - f[x_0] - f[x_0, x_1](x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} \\ &= f[x_0, x_1, x_2] \end{aligned}$$

In general, we have

$$f[x_0, x_1, \dots, x_k] = \frac{f(x_k) - \sum_{i=0}^{k-1} f[x_0, x_1, \dots, x_i] \prod_{j=0}^{i-1} (x_k - x_j)}{\prod_{j=0}^{k-1} (x_k - x_j)}$$

Computational algorithm

- Set $f[x_0] = f(x_0)$
- For $k = 1, 2, \dots, n$, compute $f[x_0, x_1, \dots, x_k]$ using the above equation

Recursive Formula

The divided difference has a recursive formula

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

Proof:

$f[x_0, x_1, \dots, x_k]$ is the coefficient of x^k in the polynomial p_k of degree $\leq k$, which interpolates f at x_0, x_1, \dots, x_k

$f[x_1, x_2, \dots, x_k]$ is the coefficient of x^{k-1} in the polynomial q_{k-1} of degree $\leq (k-1)$, which interpolates f at x_1, x_2, \dots, x_k

$f[x_0, x_1, \dots, x_{k-1}]$ is the coefficient of x^{k-1} in the polynomial p_{k-1} of degree $\leq (k-1)$, which interpolates f at x_0, x_1, \dots, x_{k-1}

Recursive Formula - Proof

We have

$$p_k(x) = q_{k-1}(x) + \frac{x - x_k}{x_k - x_0} [q_{k-1}(x) - p_{k-1}(x)]$$

To prove this identity, it suffices to show that it holds at $(k + 1)$ different points, since the left-hand side and the right-hand side are polynomials of degree $\leq k$. Note that the left-hand side is $p_k(x_i) = f(x_i)$ for $i = 0, 1, \dots, k$

Check the right-hand side at point x_0

$$\begin{aligned} & q_{k-1}(x_0) + \frac{x_0 - x_k}{x_k - x_0} [q_{k-1}(x_0) - p_{k-1}(x_0)] \\ &= q_{k-1}(x_0) - [q_{k-1}(x_0) - p_{k-1}(x_0)] \\ &= p_{k-1}(x_0) = f(x_0) \end{aligned}$$

Formula Proof - Cont.

Check for points $1 \leq i \leq (k - 1)$,

$$\begin{aligned} q_{k-1}(x_i) + \frac{x_i - x_k}{x_k - x_0} [q_{k-1}(x_i) - p_{k-1}(x_i)] \\ = f(x_i) + \frac{x_i - x_k}{x_k - x_0} [f(x_i) - f(x_i)] = f(x_i) \end{aligned}$$

Check the right-hand side at point x^k

$$\begin{aligned} q_{k-1}(x_k) + \frac{x_k - x_k}{x_k - x_0} [q_{k-1}(x_k) - p_{k-1}(x_k)] \\ = q_{k-1}(x_k) = f(x_k) \end{aligned}$$

Hence the said identity holds

We take the coefficients of x^k on both sides, which yields the desired recursive formula

Invariance Theorem

The divided difference $f[x_0, x_1, \dots, x_k]$ is invariant under all permutations of the arguments x_0, x_1, \dots, x_k

This is because $f[x_0, x_1, \dots, x_k]$ is the coefficient of x^k of the polynomial $p_k(x)$ of degree $\leq k$ that interpolates f at x_0, x_1, \dots, x_k . $f[x_1, x_0, \dots, x_k]$ is the coefficient of x^k of the polynomial $p_k(x)$ of degree $\leq k$ that interpolates f at x_1, x_0, \dots, x_k . These two polynomials are the same

The generic recursive formula is

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

Divided Difference Table

We can construct a divided difference table for f to facilitate computation of the coefficients of the interpolating polynomial

x	$f[]$	$f[,]$	$f[, ,]$	$f[, , ,]$
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	$f[x_3]$			

The coefficients along the top diagonal are the ones needed to form the Newton form of the interpolating polynomial

Divided Difference Algorithm

A pseudocode for computing divided difference is

```
real array  $(a_{ij})_{0:n \times 0:n}$ ,  $(x_i)_{0:n}$ 
integer  $i, j, n$ 
for  $i = 0$  to  $n$  do
     $a_{i0} \leftarrow f(x_i)$ 
end for
for  $j = 1$  to  $n$  do
    for  $i = 0$  to  $n - j$  do
         $a_{ij} \leftarrow (a_{i+1,j-1} - a_{i,j-1}) / (x_{i+j} - x_i)$ 
    end for
end for
```

This algorithm computes and stores all components of the divided difference. The coefficients of the Newton interpolating polynomial are stored in the first row of the array $(a_{ij})_{0:n \times 0:n}$, i.e., in $a(0 : n, 0)$

Computing the Coefficients only

If we compute divided difference only for constructing Newton interpolating polynomial, there is no need to store the unnecessary divided difference terms. (But they will be computed, used, and discarded)

```
real array  $(a_i)_{0:n}, (x_i)_{0:n}$ 
integer  $i, j, n$ 
for  $i=0$  to  $n$  do
     $a_i \leftarrow f(x_i)$ 
end for
for  $j=1$  to  $n$  do
    for  $i=n$  to  $j$  step  $-1$  do
         $a_i \leftarrow (a_i - a_{i-1}) / (x_i - x_{i-j})$ 
    end for
end for
```

The two algorithms assume the same computational cost

Memory Allocations

Here we show how the memory is occupied and updated in computing coefficients of Newton interpolating polynomial

1st	$f[x_0]$	$f[x_1]$	$f[x_2]$	$f[x_3]$	$f[x_4]$
2nd					
3rd					
4th					
5th					

Computation must be done backward to avoid erasing needed memory locations

Inverse Interpolation

It is also possible to use a polynomial to approximate the inverse of a function $y = f(x)$. Given a table

y	y_0	y_1	\dots	y_n
x	x_0	x_1	\dots	x_n

An interpolation polynomial

$$p(y) = \sum_{i=0}^n c_i \prod_{j=0}^{i-1} (y - y_j)$$

Can be constructed such that $p(y_i) = x_i$. This interpolating polynomial is useful to find the approximate location of a root of a function $f(x)$. E.g., there is a root in **[4.0,5.0]** for this table

y	-0.579	-0.363	-0.185	-0.034	0.097
x	1.0	2.0	3.0	4.0	5.0

Neville's Algorithm

Neville proposed a different scheme to construct interpolation polynomial step by step. Start with zero degree polynomials $P_i(x) = f(x_i)$, we construct higher degree interpolation polynomials by the recurrence relation

$$S_{ij}(x) = \left(\frac{x - x_{i-j}}{x_i - x_{i-j}} \right) S_{i,j-1}(x) + \left(\frac{x_i - x}{x_i - x_{i-j}} \right) S_{i-1,j-1}(x)$$

With $S_{i0}(x) = P_i(x) = f(x_i)$. The relation table can be written as

x_0	$S_{00}(x)$				
x_1	$S_{10}(x)$	$S_{11}(x)$			
x_2	$S_{20}(x)$	$S_{21}(x)$	$S_{22}(x)$		
x_3	$S_{30}(x)$	$S_{31}(x)$	$S_{32}(x)$	$S_{33}(x)$	
x_4	$S_{40}(x)$	$S_{41}(x)$	$S_{42}(x)$	$S_{43}(x)$	$S_{44}(x)$

Interpolation Property

Redefine constant polynomials as $P_i^{(0)}(x) = y_i$ for $0 \leq i \leq n$,
We can define higher order polynomial as

$$P_i^{(j)}(x) = \left(\frac{x - x_{i-j}}{x_i - x_{i-j}} \right) P_i^{(j-1)}(x) + \left(\frac{x_i - x}{x_i - x_{i-j}} \right) P_{i-1}^{(j-1)}(x)$$

The range of j is $1 \leq j \leq n$ and that of i is $j \leq i \leq n$

The interpolation properties of these polynomials are:

The polynomial $P_i^{(j)}$ defined above interpolate as follows (see p. 153 for a proof)

$$P_i^{(j)}(x_k) = y_k \quad (0 \leq i-j \leq k \leq i \leq n)$$

Higher Dimensional Interpolation

It is possible to define interpolation polynomials of several variables. The **tensor-product interpolation** is used on rectangular domain $[a,b] \times [\alpha,\beta]$. Select n nodes in $[a,b]$ and define the Lagrange polynomials as

$$l_i(x) = \prod_{j \neq i, j=1}^n \frac{x - x_j}{x_i - x_j} \quad (1 \leq i \leq n).$$

Select m nodes in $[\alpha,\beta]$ and define

$$h_i(y) = \prod_{j \neq i, j=1}^m \frac{y - y_j}{y_i - y_j} \quad (1 \leq i \leq m).$$

Then function

$$P(x, y) = \sum_{i=1}^n \sum_{j=1}^m f(x_i, y_j) l_i(x) h_j(y)$$

A two dimensional table with data

$$(x_i, y_j, f(x_i, y_j))$$

Computing First Derivative

The first derivative can be approximated as

$$f'(x) \approx \frac{1}{h}[f(x+h) - f(x)] \quad (1)$$

For accurate approximation, h should be small. Thus $f(x+h)$ and $f(x)$ are close to each other. This may cause loss of significant digits in finite precision computation

Using Taylor's theorem, we have

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(\xi)$$

For ξ between x and $x+h$. It follows that

$$f'(x) = \frac{1}{h}[f(x+h) - f(x)] - \frac{1}{2}hf''(\xi)$$

The approximation error of (1) is $-\frac{1}{2}hf''(\xi)$, or of order $O(h)$. This is a first order (or sided) approximation of first derivative. The error goes to 0 as fast as $h \rightarrow 0$

Higher Order Approximation

It is desirable to have some higher order (faster) approximation schemes

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2!}h^2 f''(x) + \frac{1}{3!}h^3 f'''(x) + \frac{1}{4!}h^4 f^{(4)}(x) + \dots$$

$$f(x-h) = f(x) - hf'(x) + \frac{1}{2!}h^2 f''(x) - \frac{1}{3!}h^3 f'''(x) + \frac{1}{4!}h^4 f^{(4)}(x) - \dots$$

Subtracting these two equations, we have

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{2}{3!}h^3 f'''(x) + \frac{2}{5!}h^5 f^{(5)}(x) + \dots$$

2nd Order Approximation

It follows that

$$f'(x) = \frac{1}{2h} [f(x+h) - f(x-h)] \\ - \frac{h^2}{3!} f'''(x) - \frac{h^4}{5!} f^{(5)}(x) - \dots$$

After dropping the higher order terms, we have a second order approximation formula as

$$f'(x) \approx \frac{1}{2h} [f(x+h) - f(x-h)]$$

The leading truncated terms of this approximation scheme is $-\frac{h^2}{6} f'''(x)$. Hence the approximation is of $O(h^2)$. The approximation error goes to $\mathbf{0}$ as fast as $h^2 \rightarrow \mathbf{0}$. The exact truncation error is

$$-\frac{1}{6} h^2 \left[\frac{f'''(\xi_1) + f'''(\xi_2)}{2} \right] = -\frac{1}{6} h^2 f'''(\xi)$$

Richardson Extrapolation (I)

First derivative can be approximated as

$$f'(x) = \frac{1}{2h} [f(x+h) - f(x-h)] \\ + a_2 h^2 + a_4 h^4 + a_6 h^6 + \dots$$

In which the constants a_2, a_4, \dots depend on the higher order derivatives of f and the value of x . When such information is available, it is possible to construct much more accurate approximation schemes

Define a function

$$\psi(h) = \frac{1}{2h} [f(x+h) - f(x-h)]$$

Which is an approximation to $f'(x)$ with error of order $O(h^2)$. This approximation becomes accurate as $h \rightarrow 0$. So we can study the quantity

$$\lim_{h \rightarrow 0} \psi(h)$$

Richardson Extrapolation (II)

Richardson extrapolation estimates the value of $\psi(0)$ from some computed values of $\psi(h)$ near 0

$$\begin{aligned}\psi(h) &= f'(x) - a_2 h^2 - a_4 h^4 - a_6 h^6 - \dots \\ \psi\left(\frac{h}{2}\right) &= f'(x) - a_2 \left(\frac{h}{2}\right)^2 - a_4 \left(\frac{h}{2}\right)^4 - a_6 \left(\frac{h}{2}\right)^6 - \dots\end{aligned}$$

Multiply the 2nd equation by 4 and subtract it from the 1st equation

$$\psi(h) - 4\psi\left(\frac{h}{2}\right) = -3f'(x) - \frac{3}{4}a_4 h^4 - \frac{15}{16}a_6 h^6 - \dots$$

Hence

$$\phi\left(\frac{h}{2}\right) + \frac{1}{3}\left[\psi\left(\frac{h}{2}\right) - \psi(h)\right] = f'(x) + \frac{a_4}{4}h^4 + \frac{5}{16}a_6 h^6 + \dots$$

$f'(x)$ can be computed as accurate as $O(h^4)$

General Approach

A general Richardson extrapolation form

$$\psi(h) = L - \sum_{k=1}^{\infty} a_{2k} h^{2k}$$

Where we assume that $\psi(h)$ is computable for any $h > 0$ and we want to approximate L as accurately as possible

Choose a special sequence $\frac{h}{2^n}$ define

$$D(n,0) = \psi\left(\frac{h}{2^n}\right) \quad (n \geq 0)$$

Then, we have

$$D(n,0) = L + \sum_{k=1}^{\infty} A(k,0) \left(\frac{h}{2^n}\right)^{2k}$$

With $A(k,0) = -a_{2k}$. $D(n,0)$ is a rough approximate of $L = \lim_{x \rightarrow 0} \psi(x)$

Richardson Theorem

The extrapolation formula is

$$D(n, m) = \frac{4^m}{4^m - 1} D(n, m - 1) - \frac{1}{4^m - 1} D(n - 1, m - 1) \quad (1 \leq m \leq n)$$

Richardson Extrapolation Theorem:

$$D(n, m) = L + \sum_{k=m+1}^{\infty} A(k, m) \left(\frac{h}{2^n} \right)^{2k}$$

For $0 \leq m \leq n$

The proof of this theorem is based on induction on m , see p. 175 of the Book. Proof will be given in class

Not that $D(n, m)$ approximates L at the order of $O(h^{2m})$. The convergence rate is fast

Computational Procedure

Richardson extrapolation computational procedure:

- 1.) write a procedure to compute $\psi(h)$
- 2.) decide on suitable values for n and h
- 3.) for $i = 0, 1, \dots, n$, compute

$$D(i,0) = \psi(h / 2^i)$$

- 4.) for $0 \leq i \leq j \leq n$, compute

$$D(i, j) = D(i, j-1) + \\ (4^j - 1)^{-1} [D(i, j-1) - D(i-1, j-1)]$$

Using Interpolation Polynomial

We can approximate the function $f(x)$ by a polynomial $p_n(x)$ of order n , such that $p_n(x) \approx f(x)$

To compute $f'(x)$, we use the approximation $f'(x) \approx p'_n(x)$

Higher order polynomials are avoided because of oscillation

Let p interpolates f at two points, x_0 and x_1

$$p_1(x) = f(x_0) + f[x_0, x_0](x - x_0)$$

The first derivative of $p_1(x)$ is

$$p'_1(x) = f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \approx f'(x)$$

1st and 2nd Order Approx.

Let $x_0 = x$ and $x_1 = x + h$, we have

$$f'(x) \approx \frac{1}{h}[f(x+h) - f(x)]$$

This is just the $O(h)$ order sided approximation formula

Put $x_0 = x - h$ and $x_1 = x + h$, we have the $O(h^2)$ approximation scheme

$$f'(x) \approx \frac{1}{2h}[f(x+h) - f(x-h)]$$

A three point polynomial interpolation is

$$\begin{aligned} p_2(x) &= f(x_0) + f[x_0, x_1](x - x_0) \\ &\quad + f[x_0, x_1, x_2](x - x_0)(x - x_1) \end{aligned}$$

We have corrected approximation

$$p'_2(x) = f[x_0, x_1] + f[x_0, x_1, x_2](2x - x_0 - x_1)$$

Second Derivative

If we have first derivative, we can use

$$f''(x) = \frac{1}{2h} [f'(x+h) - f'(x-h)]$$

To approximate the second derivative to $O(h^2)$

A direct approximation would be using Taylor expansion

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + 2 \left[\frac{1}{4!} h^4 f^{(4)}(x) + \dots \right]$$

Hence, we have

$$f''(x) \approx \frac{1}{h^2} [f(x+h) - 2f(x) + f(x-h)]$$

This approximation is of $O(h^2)$ accuracy